Intel SGX Explained

ABSTRACT

Intel's Software Guard Extensions (SGX) is a set of extensions to the Intel architecture that aims to provide integrity and confidentiality guarantees to security sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious.

英特尔软件防护扩展（SGX）是对英特尔架构的扩展，旨在为所有可能具有恶意行为的特许软件（内核，管理程序等）计算机上执行的安全敏感计算提供完整性和机密性保证。

This paper analyzes Intel SGX, based on the 3 papers [14, 79, 139] that introduced it, on the Intel Software Developer's Manual [101] (which supersedes the SGX manuals [95, 99]), on an ISCA 2015 tutorial [103], and on two patents [110, 138]. We use the papers, reference manuals, and tutorial as primary data sources, and only draw on the patents to fill in missing information.

本文在 ISCA 2015 教程的基础上，根据英特尔软件开发人员手册[101]（取代 SGX 手册[95,99]）的 3 篇论文[14,79,139]分析了英特尔 SGX [103]和两项专利[110,138]。我们使用论文，参考手册和教程作为主要数据来源，并仅使用专利填写缺失信息。

This paper does not reflect the information available in two papers [74, 109] that were published after the first version of this paper.

本文不反映本文第一版后发表的两篇论文[74,109]中的信息。

This paper's contributions are a summary of the Intel-specific architectural and Micro-architectural details needed to understand SGX, a detailed and structured presentation of the publicly available information on SGX, series of intelligent guesses about some important but undocumented aspects of SGX, and an analysis of SGX's security properties.

本文的内容是对于了解 SGX 需要的英特尔架构细节和微观架构细节的总结，对 SGX 公开信息的详细和结构化介绍，对 SGX 一些重要但未公开的方面的一系列 intelligent guesses，以及分析 SGX 的安全性能。

1 OVERVIEW

Secure remote computation (Figure 1) is the problem of executing software on a remote computer owned and maintained by an untrusted party, with some integrity and confidentiality guarantees. In the general setting, secure remote computation is an unsolved problem. Fully Homomorphic Encryption [61] solves the problem for a limited family of computations, but has an impractical performance overhead [140].

安全的远程计算（图 1）是由不受信任方拥有和维护的远程计算机上执行软件的问题，具有一些完整性和机密性保证。在一般情况下，安全远程计算是一个未解决的问题。完全同态加密[61]解决了有限计算系列的问题，但具有不切实际的性能开销[140]。

Intel's Software Guard Extensions (SGX) is the latest iteration in a long line of trusted computing (Figure 2) designs, which aim to solve the secure remote computation problem by leveraging trusted hardware in the remote computer. The trusted hardware establishes a secure container, and the remote computation service user uploads the desired computation and data into the secure container. The trusted hardware protects the data's confidentiality and integrity while the computation is being performed on it.

英特尔的软件防护扩展（SGX）是长期可信计算（图 2）设计中的最新一次迭代，

旨在通过利用远程计算机中的可信硬件来解决安全的远程计算问题。受信任的硬件建立一个安全容器，远程计算服务用户将所需的计算和数据上载到安全容器中。可信硬件在计算正在执行时保护数据的机密性和完整性。

SGX relies on *software attestation*, like its predecessors, the TPM [71] and TXT [70]. Attestation (Figure 3) proves to a user that she is communicating with a specific piece of software running in a secure container hosted by the trusted hardware. The proof is a cryptographic signature that certifies the hash of the secure container's contents. It follows that the remote computer's owner can load any software in a secure container, but the remote computation service user will refuse to load her data into a secure container whose contents' hash does not match the expected value.

SGX 依赖软件认证，就像其前身 TPM [71]和 TXT [70]一样。证明（图 3）向用户证明她正在与受信任硬件托管的安全容器中运行的特定软件进行通信。证明是一个加密签名，用于证明安全容器的内容的散列。因此，远程计算机的所有者可以在安全容器中加载任何软件，但远程计算服务用户将拒绝将其数据加载到其内容的散列值与期望值不匹配的安全容器中。

The remote computation service user verifies the *attestation key* used to produce the signature against an *endorsement certificate* created by the trusted hardware's manufacturer. The certificate states that the attestation key is only known to the trusted hardware, and only used for the purpose of attestation.

远程计算服务用户根据由可信硬件制造商创建的 *endorsement certificate* 来验证用于产生签名的 *attestation key*。The certificate 规定 *attestation key* 仅为可信硬件所知，并且仅用于认证目的。

SGX stands out from its predecessors by the amount of code covered by the attestation, which is in the Trusted Computing Base (TCB) for the system using hardware protection. The attestations produced by the original TPM design covered all the software running on a computer, and TXT attestations covered the code inside a VMX [181] virtual machine. In SGX, an *enclave* (secure container) only contains the private data in a computation, and the code that operates on it.

SGX 与其前身使用硬件保护的系统的可信计算基（TCB）中包含的证书代码数量差异较大。原始 TPM 设计产生的证明涵盖了计算机上运行的所有软件，并且 TXT 证明涵盖了 VMX [181]虚拟机内的代码。在 SGX 中，enclave（安全容器）仅包含计算中的 private data 以及对其进行操作的代码。

For example, a cloud service that performs image processing on confidential medical images could be implemented by having users upload encrypted images. The users would send the encryption keys to software running inside an enclave. The enclave would contain the code for decrypting images, the image processing algorithm, and the code for encrypting the results. The code that receives the uploaded encrypted images and stores them would be left outside the enclave.

例如，可以通过让用户上传加密图像来实现对机密医疗图像执行图像处理的云服务。用户会将加密密钥发送给在 enclave 内运行的软件。该 enclave 将包含用于解密图像的代码，图像处理算法和用于加密结果的代码。接收上传的加密图像并存储它们的代码将留在 encalve 之外。

An SGX-enabled processor protects the integrity and confidentiality of the computation inside an enclave by isolating the enclave's code and data from the outside environment,

including the operating system and hypervisor, and hardware devices attached to the system bus. At the same time, the SGX model remains compatible with the traditional software layering in the Intel architecture, where the OS kernel and hypervisor manage the computer's resources.

支持 SGX 的处理器通过隔离 enclave 的代码和数据与外部环境（包括操作系统和管理程序）以及连接到系统总线的硬件设备，来保护 enclave 内计算的完整性和机密性。与此同时，SGX 模型仍然与英特尔架构中的传统软件分层兼容，在该体系结构中，操作系统内核和管理程序管理计算机的资源。

This work discusses the original version of SGX, also referred to as SGX 1. While SGX 2 brings very useful improvements for enclave authors, it is a small incremental improvement from a design and implementation standpoint. After understanding the principles behind SGX 1 and its security properties, the reader should be well equipped to face Intel's reference documentation and learn about the changes brought by SGX 2.

这项工作讨论了 SGX 的原始版本，也被称为 SGX 1。虽然 SGX 2 为 enclave authors 带来了非常有用的改进，但是从设计和实施的角度来看，这是一个小幅度的改进。在了解 SGX 1 及其安全属性背后的原理之后，读者应该能够很好地面对英特尔的参考文档，并了解 SGX2 带来的变化。

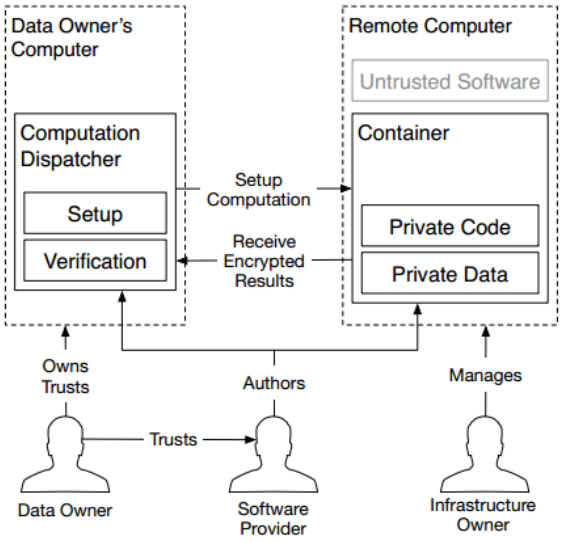


Figure 1: Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation's integrity and confidentiality.

图 1：安全远程计算。用户依赖于不受信任方拥有的远程计算机对其数据执行一些计算。用户对计算的完整性和机密性有一定的保证。

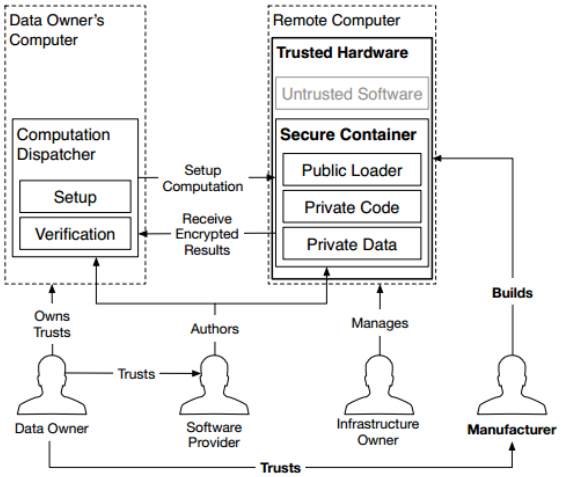


Figure 2: Trusted computing. The user trusts the manufacturer of a piece of hardware in the remote computer, and entrusts her data to a secure container hosted by the secure hardware.

图 2：可信计算。用户信任远程计算机中的硬件制造商，并将其数据委托给安全硬件托管的安全容器。

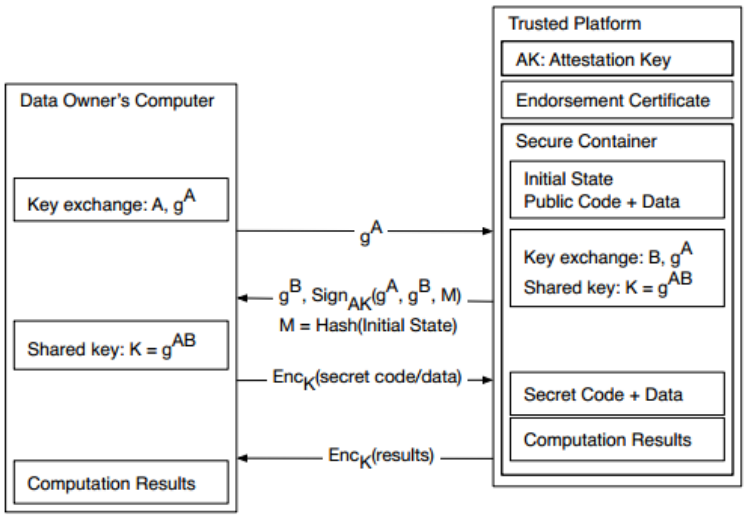


Figure 3: Software attestation proves to a remote computer that it is communicating with a specific secure container hosted by a trusted platform. The proof is an attestation signature produced by the platform's secret attestation key. The signature covers the container's initial state, a challenge nonce produced by the remote computer, and a message produced by the container.

图 3：软件认证向远程计算机证明它正在与受信任平台托管的特定安全容器进行通信。证明是由平台的秘密认证密钥产生的认证签名。签名涵盖了容器的初始状态，远程计算机产生的质询随机数和容器产生的消息。

1.1 SGX Lightning Tour

SGX sets aside a memory region, called the *Processor Reserved Memory* (PRM, *x* 5.1). The CPU protects the PRM from all non-enclave memory accesses, including kernel, hypervisor and SMM (*x* 2.3) accesses, and DMA accesses (*x* 2.9.1) from peripherals.

SGX 留出一个内存区域，称为 *Processor Reserved Memory*（PRM，x 5.1）。CPU 保护 PRM 免受所有非安全区内存访问的影响，包括内核、管理程序和 SMM（x 2.3）访问

以及来自外设的 DMA 访问（x 2.9.1）。

The PRM holds the *Enclave Page Cache* (EPC, *x* 5.1.1), which consists of 4 KB pages that store enclave code and data. The system software, which is untrusted, is in charge of assigning EPC pages to enclaves. The CPU tracks each EPC page's state in the *Enclave Page Cache Metadata* (EPCM, *x* 5.1.2), to ensure that each EPC page belongs to exactly one enclave.

PRM holds Enclave Page Cache（EPC，x 5.1.1），EPC 由 4 KB 页面组成，用于存储 enclave 代码和数据。系统软件不受信任，负责将 EPC 页面分配给 enclave。CPU 追踪 EPC 元数据（EPCM，x 5.1.2）中的每个 EPC 页面的状态，以确保每个 EPC 页面恰好属于一个 enclave。

The initial code and data in an enclave is loaded by untrusted system software. During the loading stage (*x* 5.3), the system software asks the CPU to copy data from unprotected memory (outside PRM) into EPC pages, and assigns the pages to the enclave being setup (*x* 5.1.2). It follows that the initial enclave state is known to the system software.

一个 enclave 中的初始代码和数据由不受信任的系统软件加载。在加载阶段（x 5.3），系统软件要求 CPU 将数据从未受保护的存储器（PRM 外部）复制到 EPC 页面，并将页面分配到正在 setup 的 enclave（x 5.1.2）。因此，系统软件知道最初的 enclave 状态。

After all the enclave's pages are loaded into EPC, the system software asks the CPU to mark the enclave as initialized (*x* 5.3), at which point application software can run the code inside the enclave. After an enclave is initialized, the loading method described above is disabled.

所有 enclave 的页面加载到 EPC 后，系统软件要求 CPU 将 enclave 标记为已初始化（x 5.3），此时应用软件可以在 enclave 内运行代码在 encalve 初始化后，上述的加载方法被禁用。

While an enclave is loaded, its contents is cryptographically hashed by the CPU. When the enclave is initialized, the hash is finalized, and becomes the enclave's *measurement hash* (*x* 5.6).

在加载 enclave 时，其内容由 CPU 加密散列。当 enclave 被初始化时，散列被最终确定，并成为 enclave 的 *measurement hash*（x 5.6）。

A remote party can undergo a *software attestation* process (*x* 5.8) to convince itself that it is communicating with an enclave that has a specific measurement hash, and is running in a secure environment. Execution flow can only enter an enclave via special CPU instructions (*x* 5.4), which are similar to the mechanism for switching from user mode to kernel mode. Enclave execution always happens in protected mode, at ring 3, and uses the address translation set up by the OS kernel and hypervisor.

远程方可以通过软件认证过程（x 5.8）以证明自己它正在与具有特定测量散列的飞地通信，并且正在安全环境中运行。执行流程只能通过特殊的 CPU 指令（x 5.4）进入飞地，这类似于从用户模式切换到内核模式的机制。Enclave 执行始终发生在保护模式下，在环 3，并使用由 OS 内核和管理程序设置的地址转换。

To avoid leaking private data, a CPU that is executing enclave code does not directly service an interrupt, fault (e.g., a page fault) or VM exit. Instead, the CPU first performs an Asynchronous Enclave Exit (*x* 5.4.3) to switch from enclave code to ring 3 code, and then services the interrupt, fault, or VM exit. The CPU performs an AEX by saving the CPU state into a predefined area inside the enclave and transfers control to a pre-specified instruction outside the enclave, replacing CPU registers with synthetic values.

为避免泄漏私密数据，正在执行飞地代码的 CPU 不会直接处理中断，故障（例如页面故障）或 VM exit。相反，CPU 首先执行 Asynchronous Enclave Exit（x 5.4.3），从飞地代码切换到 ring 3 代码，然后为中断，故障或 VM exit 提供服务。CPU 通过将 CPU 状态保存到飞地内的预定义 area 并将控制转移到飞地外的预先指定的指令来执行 AEX，用 synthetic 值替换 CPU 寄存器。

The allocation of EPC pages to enclaves is delegated to the OS kernel (or hypervisor). The OS communicates its allocation decisions to the SGX implementation via special ring 0 CPU instructions (*x* 5.3). The OS can also evict EPC pages into untrusted DRAM and later load them back, using dedicated CPU instructions. SGX uses cryptographic protections to assure the confidentiality, integrity and freshness of the evicted EPC pages while they are stored in untrusted memory.

将 EPC 页面分配给 enclave 被委托给 OS 内核（或管理程序）。操作系统通过特殊的 ring 0 CPU 指令（x 5.3）将其分配决策传达给 SGX implement。操作系统也可以将 EPC 页面 evict 到不可信的 DRAM 中，然后使用专用的 CPU 指令加载它们。SGX 使用加密保护(cryptographic protections)来确保 evicted EPC 页面的机密性、完整性和 fereshness，同时它们存储在不受信任的内存中。

4 RELATED WORK

This section describes the broader picture of trusted hardware projects that SGX belongs to. Table 12 summarizes the security properties of SGX and the other trusted hardware presented here.

本节介绍 SGX 所属的可信硬件项目的更广泛情况。表 12 总结了 SGX 和此处介绍的其他可信硬件的安全属性。

4.1 The IBM 4765 Secure Coprocessor

Secure coprocessors [198] encapsulate an entire computer system, including a CPU, a cryptographic accelerator, caches, DRAM, and an I/O controller within a tamper-resistant environment. The enclosure includes hardware that deters attacks, such as a Faraday cage, as well as an array of sensors that can detect tampering attempts. The secure coprocessor destroys the secrets that it stores when an attack is detected. This approach has good security properties against physical attacks, but tamper-resistant enclosures are very expensive [15], relatively to the cost of a computer system.

Secure coprocessors [198]将整个计算机系统（包括 CPU，cryptographic accelerator，高速缓存，DRAM 和 I/O 控制器）封装在防篡改环境中。The enclosure(围绕; 圈占地)包括阻止攻击的硬件，如 Faraday cage，以及可检测篡改企图的一系列传感器。Secure coprocessors 销毁检测到攻击时存储的秘密。这种方法具有很好的抵御物理攻击的安全属性，但防篡改 enclosures 相对于计算机系统的成本非常昂贵[15]。

The IBM 4758 [172], and its most current-day successor, the IBM 4765 [2] (shown in Figure 57) are representative examples of secure coprocessors. The 4758 was certified to withstand physical attacks to FIPS 140-1 Level 4 [171], and the 4765 meets the rigors of FIPS 140-2 Level 4 [1].

IBM 4758 [172]及其最新的继任者 IBM 4765 [2]（如图 57 所示）是 secure coprocessors 的代表性示例。4758 被认证可承受物理攻击 FIPS 140-1 4 级[171]，4765 满足 FIPS 140-2 4 级[1]的严格要求。

The 4765 relies heavily on physical isolation for its security properties. Its system software

is protected from attacks by the application software by virtue of using a dedicated service processor that is completely separate from the application processor. Special-purpose bus logic prevents the application processor from accessing privileged resources, such as the battery-backed memory that stores the system software's secrets.

4765 在很大程度上依赖于其安全属性的物理隔离。通过使用与应用处理器完全分离的专用服务处理器，其系统软件免受应用软件的攻击。专用总线逻辑可防止应用程序处理器访问特权资源，例如存储系统软件秘密的 battery-backed 内存。

The 4765 implements software attestation. The coprocessor's attestation key is stored in battery-backed memory that is only accessible to the service processor. Upon reset, the service processor executes a first-stage bootloader stored in ROM, which measures and loads the system software. In turn, the system software measures the application code stored in NVRAM and loads it into the DRAM chip accessible to the application processor. The system software provides attestation services to the application loaded inside the coprocessor.

4765 实施软件认证。Coprocessor 的认证密钥存储在只能由服务处理器访问的 battery-backed 内存中。复位后(Upon reset)，服务处理器执行一个存储在 ROM 中的第一级引导加载程序，用于测量和加载系统软件。接着，系统软件测量存储在 NVRAM 中的应用程序代码并将其加载到应用程序处理器可访问的 DRAM 芯片中。系统软件向 coprocessor 中加载的应用程序提供认证服务。

4.2 ARM TrustZone

ARM's TrustZone [13] is a collection of hardware modules that can be used to conceptually partition a system's resources between a *secure world*, which hosts a secure container, and a *normal world*, which runs an untrusted software stack. The TrustZone documentation [18] describes semiconductor intellectual property cores (IP blocks) and ways in which they can be combined to achieve certain security properties, reflecting the fact that ARM is an IP core provider, not a chip manufacturer. Therefore, the mere presence of TrustZone IP blocks in a system is not sufficient to determine whether the system is secure under a specific threat model. Figure 58 illustrates a design for a smartphone *System-on-Chip* (SoC) design that uses TrustZone IP blocks.

ARM 的 TrustZone [13]是一系列硬件模块，可用于从概念上将系统资源划分为 a *secure world*, host a 安全容器，a *normal world,* 运行不可信的软件栈。TrustZone 文档[18]描述了半导体知识产权核心（IP 模块）以及它们可以组合以实现某些安全属性的方式，反映了 ARM 是 IP 核心提供商而不是芯片制造商。因此，仅仅在系统中出现 TrustZone IP 模块并不足以确定系统在特定威胁模型下是否安全。图 58 显示了使用 TrustZone IP 模块的智能手机片上系统（SoC）设计。

TrustZone extends the address lines in the AMBA AXI system bus [17] with one signal that indicates whether an access belongs to the secure or normal (non-secure) world. ARM processor cores that include TrustZone's "Security Extensions" can switch between the normal world and the secure world when executing code. The address in each bus access executed by a core reflects the world in which the core is currently executing.

TrustZone 扩展了 AMBA AXI 系统总线[17]中的地址线，其中一个信号指示访问是属于安全还是 normal（非安全）world。包含 TrustZone 的"安全扩展"的 ARM 处理器内核可以在执行代码时在 normal world 和 secure world 之间切换。内核执行的每个总线访问中的地址反映了内核当前正在执行的 world。

The reset circuitry in a TrustZone processor places it in secure mode, and points it to the

first-stage bootloader stored in on-chip ROM. TrustZone's TCB includes this bootloader, which initializes the platform, sets up the TrustZone hardware to protect the secure container from untrusted software, and loads the normal world's bootloader. The secure container must also implement a monitor that performs the context switches needed to ransition an execution core between the two worlds. The onitor must also handle hardware exceptions, such as nterrupts, and route them to the appropriate world.

TrustZone 处理器中的复位电路将其置于安全模式，并将其指向存储在片内 ROM 中的第一级引导加载程序。TrustZone 的 TCB 包含这个引导加载程序，它初始化平台，设置 TrustZone 硬件以保护安全容器免受不可信软件的攻击，并加载普通世界的引导加载程序。安全容器还必须实现监视器，该监视器执行转换两个世界之间的执行核心所需的上下文切换。监督者还必须处理硬件异常（例如中断），并将其路由到适当的世界。

The TrustZone design gives the secure world's monitor unrestricted access to the normal world, so the monitor can implement inter-process communication (IPC) between the software in the two worlds. Specifically, the monitor can issue bus accesses using both secure and non-secure addresses. In general, the secure world's software can compromise any level in the normal world's software stack. For example, the secure container's software can jump into arbitrary locations in the normal world by flipping a bit in a register. The untrusted software in the normal world can only access the secure world via an instruction that jumps into a well-defined location inside the monitor.

TrustZone 设计为安全的世界监视器提供了对正常世界的无限制访问，因此监视器可以在两个世界中的软件之间实现进程间通信（IPC）。 具体来说，监视器可以使用安全和非安全地址发出总线访问。一般来说，安全世界的软件可能会影响正常世界软件栈的任何级别。例如，安全容器的软件可以通过在寄存器中翻转一点而跳入正常世界的任意位置。正常世界中的不可信软件只能通过跳转到显示器内定义明确的位置的指令访问安全世界。

Conceptually, each TrustZone CPU core provides separate address translation units for the secure and normal worlds. This is implemented by two page table base registers, and by having the page walker use the page table base corresponding to the core's current world. The physical addresses in the page table entries are extended to include the values of the secure bit to be issued on the AXI bus. The secure world is protected from untrusted software by having the CPU core force the secure bit in the address translation result to zero for normal world address translations. As the secure container manages its own page tables, its memory accesses cannot be directly observed by the untrusted OS's page fault handler.

从概念上讲，每个 TrustZone CPU 内核都为 secure and normal worlds 提供单独的地址转换单元。这是通过两个页面表基址寄存器来实现的，并且让页面遍历器使用与核心当前世界相对应的页表基地。页表项中的物理地址被扩展为包含要在 AXI 总线上发布的安全位的值。通过让 CPU 内核将地址转换结果中的安全位强制为零来实现正常的世界地址转换，可以保护 secure world 免受不可信软件的攻击。由于安全容器管理自己的页表，因此不可信的操作系统的页面错误处理程序不能直接观察其内存访问。

TrustZone-aware hardware modules, such as caches, are trusted to use the secure address bit in each bus access to enforce the isolation between worlds. For example, TrustZone's caches store the secure bit in the address tag for each cache line, which effectively provides completely different views of the memory space to the software running in different worlds. This design assumes that memory space is partitioned between the two worlds, so no aliasing can occur.

TrustZone 感知的硬件模块（例如缓存）受信任使用每个总线访问中的安全地址位来强制实现世界之间的隔离。例如，TrustZone 的高速缓存将安全位存储在每个高速缓存行的地址标记中，这有效地为在不同世界中运行的软件提供完全不同的内存空间视图。这种设计假定存储空间在两个世界之间被分割，所以不会出现混叠。

The TrustZone documentation describes two TLB configurations. If many context switches between worlds are expected, the TLB IP blocks can be configured to include the secure bit in the address tag. Alternatively, the secure bit can be omitted from the TLBs, as long as the monitor flushes the TLBs when switching contexts.

TrustZone 文档描述了两种 TLB 配置。如果期望世界之间有许多上下文切换，则可以将 TLB IP 块配置为在地址标记中包含安全位。或者，只要监视器在切换上下文时刷新 TLB，安全位就可以从 TLB 中省略。

The hardware modules that do not consume TrustZone's address bit are expected to be connected to the AXI bus via IP cores that implement simple partitioning techniques. For example, the TrustZone Memory Adapter (TZMA) can be used to partition an on-chip ROM or SRAM into a secure region and a normal region, and the TrustZone Address Space Controller (TZASC) partitions the memory space provided by a DRAM controller into secure and normal regions. A TrustZoneaware DMA controller rejects DMA transfers from the normal world that reference secure world addresses.

不占用 TrustZone 地址位的硬件模块预计将通过实现简单分区技术的 IP 内核连接到 AXI 总线。例如，TrustZone 内存适配器（TZMA）可用于将片上 ROM 或 SRAM 划分为 secure region 和 normal region，TrustZone 地址空间控制器（TZASC）将 DRAM 控制器提供的内存空间划分为 secure and normal regions。TrustZoneaware DMA 控制器拒绝来自 normal world 引用 secure world 地址的 DMA 传输。

It follows that analyzing the security properties of a TrustZone system requires a precise understanding of the behavior and configuration of all the hardware modules that are attached to the AXI bus. For example, the caches described in TrustZone's documentation do not enforce a complete separation between worlds, as they allow a world's memory accesses to evict the other world's cache lines. This exposes the secure container software to cache timing attacks from the untrusted software in the normal world. Unfortunately, hardware manufacturers that license the TrustZone IP cores are reluctant to disclose all the details of their designs, making it impossible for security researchers to reason about TrustZone-based hardware.

因此，分析 TrustZone 系统的安全属性需要准确理解连接到 AXI 总线的所有硬件模块的行为和配置。例如，TrustZone 文档中描述的缓存不能强制在世界之间完全分离，因为它们允许全球内存访问来驱逐其他世界的缓存线。这暴露了安全容器软件缓存来自正常世界中不可信软件的计时攻击。不幸的是，许可 TrustZone IP 内核的硬件制造商不愿意公开他们设计的所有细节，使安全研究人员无法推断基于 TrustZone 的硬件。

The TrustZone components do not have any countermeasures for physical attacks. However, a system that follows the recommendations in the TrustZone documentation will not be exposed to physical attacks, under a threat model that trusts the processor chip package. The AXI bus is designed to connect components in an SoC design, so it cannot be tapped by an attacker. The TrustZone documentation recommends having all the code and data in the secure world stored in on-chip SRAM, which is not subject to physical attacks. However, this approach places significant limits on the secure container's functionality, because on-chip SRAM is many orders of magnitude more expensive than a DRAM chip of the same capacity.

TrustZone 组件没有针对物理攻击的任何对策。但是，遵循 TrustZone 文档中的建议的系统在受信任处理器芯片包的威胁模型下不会受到物理攻击。AXI 总线设计用于连接 SoC 设计中的组件，因此攻击者无法窃听。TrustZone 文档建议将安全领域的所有代码和数据存储在片上 SRAM 中，这些 SRAM 不受物理攻击。然而，这种方法对安全容器的功能造成了很大的限制，因为片上 SRAM 比相同容量的 DRAM 芯片贵很多个数量级。

TrustZone's documentation does not describe any software attestation implementation. However, it does outline a method for implementing secure boot, which comes down to having the first-stage bootloader verify a signature in the second-stage bootloader against a public key whose cryptographic hash is burned into on-chip *One-Time Programmable* (OTP) polysilicon fuses. A hardware measurement root can be built on top of the same components, by storing a per-chip attestation key in the polyfuses, and having the first-stage bootloader measure the second-stage bootloader and store its hash in an on-chip SRAM region allocated to the secure world. The polyfuses would be gated by a TZMA IP block that makes them accessible only to the secure world.

TrustZone 的文档没有描述任何软件认证实施。但是，它的确概述了一种实现安全启动的方法，这归结为让第一阶段启动加载程序在第二阶段启动加载程序中对照公钥加密验证签名，该公钥密码散列烧录到片上一次性可编程（OTP） ）多晶硅熔丝。 硬件测量根可以建立在相同的组件之上，通过在多晶硅熔丝中存储每芯片认证密钥，并且使第一级引导加载程序测量第二级引导加载程序并将其散列存储在片上 SRAM 区域中 分配给安全的世界。 多晶硅将由 TZMA IP 模块进行门控，使其只能在安全的环境下使用。

4.3 The XOM Architecture

The execute-only memory (XOM) architecture [128] introduced the approach of executing sensitive code and data in isolated containers managed by untrusted host software. XOM outlined the mechanisms needed to isolate a container's data from its untrusted software environment, such as saving the register state to a protected memory area before servicing an interrupt.

XOM supports multiple containers by tagging every cache line with the identifier of the container owning it, and ensures isolation by disallowing memory accesses to cache lines that don't match the current container's identifier. The operating system and the untrusted applications are considered to belong to a container with a null identifier.

XOM also introduced the integration of encryption and HMAC functionality in the processor's memory controller to protect container memory from physical attacks on DRAM. The encryption and HMAC functionality is used for all cache line evictions and fetches, and the ECC bits in DRAM chips are repurposed to store HMAC values.

XOM's design cannot guarantee DRAM freshness, so the software in its containers is vulnerable to physical replay attacks. Furthermore, XOM does not protect a container's memory access patterns, meaning that any piece of malicious software can perform cache timing attacks against the software in a container. Last, XOM containers are destroyed when they encounter hardware exceptions, such as page faults, so XOM does not support paging.

XOM predates the attestation scheme described above, and relies on a modified software distribution scheme instead. Each container's contents are encrypted with a symmetric key, which also serves as the container's identity. The symmetric key, in turn, is encrypted with the public key of each CPU that is trusted to run the container. A container's author can be assured that the container is running on trusted software by embedding a secret into the encrypted

container data, and using it to authenticate the container. While conceptually simpler than software attestation, this scheme does not allow the container author to vet the container's software environment.

4.4 The Trusted Platform Module (TPM)

The Trusted Platform Module (TPM) [71] introduced the software attestation model described at the beginning of this section. The TPM design does not require any hardware modifications to the CPU, and instead relies on an auxiliary tamper-resistant chip. The TPM chip is only used to store the attestation key and to perform software attestation. The TPM was widely deployed on commodity computers, because it does not rely on CPU modifications. Unfortunately, the cost of this approach is that the TPM has very weak security guarantees, as explained below.

可信平台模块（TPM）[71]介绍了本节开头描述的软件认证模型。TPM 设计不需要对 CPU 进行任何硬件修改，而是依赖于辅助防篡改芯片。 TPM 芯片仅用于存储认证密钥并执行软件认证。TPM 广泛部署在商用计算机上，因为它不依赖于 CPU 修改。不幸的是，这种方法的成本是 TPM 具有非常弱的安全保证，如下所述。

The TPM design provides one isolation container, covering all the software running on the computer that has the TPM chip. It follows that the measurement included in an attestation signature covers the entire OS kernel and all the kernel modules, such as device drivers. However, commercial computers use a wide diversity of devices, and their system software is updated at an ever-increasing pace, so it is impossible to maintain a list of acceptable measurement hashes corresponding to a piece of trusted software. Due to this issue, the TPM's software attestation is not used in many security systems, despite its wide deployment.

TPM 设计提供了一个隔离容器，涵盖了具有 TPM 芯片的计算机上运行的所有软件。因此，认证签名中包含的 measurement 涵盖了整个操作系统内核和所有内核模块，例如设备驱动程序。然而，商业计算机使用各种各样的设备，并且他们的系统软件以不断增长的速度更新，所以无法维持与可信软件相对应的可接受 measurement hashes 列表。由于这个问题，尽管 TPM 的广泛部署，TPM 的软件认证并未在许多安全系统中使用。

The TPM design is technically not vulnerable to any software attacks, because it trusts all the software on the computer. However, a TPM-based system is vulnerable to an attacker who has physical access to the machine, as the TPM chip does not provide any isolation for the software on the computer. Furthermore, the TPM chip receives the software measurements from the CPU, so TPM-based systems are vulnerable to attackers who can tap the communication bus between the CPU and the TPM.

TPM 设计在技术上不容易受到任何软件攻击，因为它信任计算机上的所有软件。但是，基于 TPM 的系统容易受到具有物理访问权限攻击者的攻击，因为 TPM 芯片不能为计算机上的软件提供任何隔离。此外，TPM 芯片接收来自 CPU 的 software measurements，因此基于 TPM 的系统很容易受到攻击者的攻击，攻击者可以利用 CPU 和 TPM 之间的通信总线。

Last, the TPM's design relies on the software running on the CPU to report its own cryptographic hash. The TPM chip resets the measurements stored in Platform Configuration Registers (PCRs) when the computer is rebooted. Then, the TPM expects the software at each boot stage to cryptographically hash the software at the next stage, and send the hash to the TPM. The TPM updates the PCRs to incorporate the new hashes it receives, as shown in Figure 59. Most importantly, the PCR value at any point reflects all the software hashes received by

the TPM up to that point. This makes it impossible for software that has been measured to "remove" itself from the measurement.

最后，TPM 的设计依赖 CPU 上运行的软件来报告自己的加密散列。 计算机重新启动时，TPM 芯片会重置存储在平台配置寄存器（PCR）中的 measurements。然后，TPM 期望在每个引导阶段的软件在下一阶段加密散列软件，并将散列发送到 TPM。如图 59 所示，TPM 更新 PCRs 以包含它接收到的新哈希值。最重要的是，任何点的 PCR 值都反映了 TPM 收到的所有软件哈希值。这使得测量的软件不能从 measurement.中"remove" itself。

For example, the firmware on most modern computers implements the platform initialization process in the Unified Extensible Firmware Interface (UEFI) specification [180]. Each platform initialization phase is responsible for verifying or measuring the firmware that implements the next phase. The SEC firmware initializes the TPM PCR, and then stores the PEI's measurement into a measurement register. In turn, the PEI implementation measures the DXE firmware and updates the measurement register that stores the PEI hash to account for the DXE hash. When the OS is booted, the hash in the measurement register accounts for all the firmware that was used to boot the computer.

例如，大多数现代计算机上的固件都采用统一可扩展固件接口（UEFI）规范[180]中的平台初始化过程。每个平台初始化阶段负责验证或测量实施下一阶段的固件。 SEC 固件初始化 TPM PCR，然后将 PEI 的测量存储到测量寄存器中。接下来，PEI 实施测量 DXE 固件并更新存储 PEI 哈希的测量寄存器以考虑 DXE 哈希。操作系统启动时，测量寄存器中的散列值 accounts for 所有用于启动计算机的固件。

Unfortunately, the security of the whole measurement scheme hinges on the requirement that the first hash sent to the TPM must reflect the software that runs in the first boot stage. The TPM threat model explicitly acknowledges this issue, and assumes that the firmware responsible for loading the first stage bootloader is securely embedded in the motherboard. However, virtually every TPM-enabled computer stores its firmware in a flash memory chip that can be re-programmed in software (*x* 2.9.1), so the TPM's measurement can be subverted by an attacker who can reflash the computer's firmware [29].

不幸的是，整个测量方案的安全性取决于发送给 TPM 的第一个散列必须反映在第一个启动阶段运行的软件的要求。TPM 威胁模型明确承认这个问题，并假设负责加载第一级引导加载程序的固件安全地嵌入主板中。然而，事实上每个支持 TPM 的计算机都会将其固件存储在一个闪存芯片中，该芯片可以用软件重新编程（x 2.9.1），因此 TPM 的测量可能会被可重新刷新计算机固件的攻击者破坏[29]。

On very recent Intel processors, the attack described above can be defeated by having the initialization microcode (*x* 2.14.4) hash the computer's firmware (specifically, the PEI code in UEFI [180] firwmare) and communicate the hash to the TPM chip. This is marketed as the Measured Boot feature of Intel's Boot Guard [162]. Sadly, most computer manufacturers use Verified Boot (also known as "secure boot") instead of Measured Boot (also known as "trusted boot"). Verified Boot means that the processor's microcode only boots into PEI firmware that contains a signature produced by a key burned into the chip's e-fuses. Verified Boot does not impact the measurements stored on the TPM, so it does not improve the security of software attestation.

在最近的英特尔处理器上，上述攻击可以通过初始化微码（x 2.14.4）将计算机的固件（特别是 UEFI [180] firwmare 中的 PEI 代码）散列并传递给 TPM 芯片来破解。这款

产品作为英特尔 Boot Guard 的测量启动功能销售[162]。可悲的是，大多数计算机制造商使用 Verified Boot（也称为“安全启动”）而不是 Measured Boot（也称为“可信任启动”）。“Verified Boot”意味着处理器的微代码只能引导到 PEI 固件中，该固件包含由烧录到芯片 e-fuses 的密钥产生的签名。Verified Boot 不会影响存储在 TPM 上的测量结果，因此它不会提高软件认证的安全性。

4.5 Intel's Trusted Execution Technology (TXT)

Intel's Trusted Execution Technology (TXT) [70] uses the TPM's software attestation model and auxiliary tamper-resistant chip, but reduces the software inside the secure container to a virtual machine (guest operating system and application) hosted by the CPU's hardware virtualization features (VMX [181]).

英特尔的可信执行技术（TXT）[70]使用 TPM 的软件认证模型和辅助防篡改芯片，但将安全容器内的软件减少为由 CPU 的硬件虚拟化功能托管的虚拟机（客户操作系统和应用程序）VMX [181]）。

TXT isolates the software inside the container from untrusted software by ensuring that the container has exclusive control over the entire computer while it is active. This is accomplished by a secure initialization authenticated code module (SINIT ACM) that effectively performs a warm system reset before starting the container's VM.

TXT 通过确保容器在处于活动状态时对整个计算机具有排他性控制（exclusive control），将容器内的软件与不可信软件隔离开来。这是通过一个安全的初始化认证代码模块（SINIT ACM）完成的，该模块在启动容器的 VM 之前有效地执行 warm 系统重置。

TXT requires a TPM chip with an extended register set. The registers used by the measured boot process described in *x* 4.4 are considered to make up the platform's Static Root of Trust Measurement (SRTM). When a TXT VM is initialized, it updates TPM registers that make up the Dynamic Root of Trust Measurement (DRTM). While the TPM's SRTM registers only reset at the start of a boot cycle, the DRTM registers are reset by the SINIT ACM, every time a TXT VM is launched.

TXT 需要带有扩展寄存器组的 TPM 芯片。由 x 4.4 中描述的测量启动过程（measured boot process）所使用的寄存器被认为构成了平台的可信测量根（SRTM）。当 TXT VM 初始化时，它会更新组成动态可信根测量根（DRTM）的 TPM 寄存器。虽然 TPM 的 SRTM 寄存器仅在引导周期开始时复位，但每次启动 TXT VM 时，DRTM 寄存器都由 SINIT ACM 复位。

TXT does not implement DRAM encryption or HMACs, and therefore is vulnerable to physical DRAM attacks, just like TPM-based designs. Furthermore, early TXT implementations were vulnerable to attacks where a malicious operating system would program a device, such as a network card, to perform DMA transfers to the DRAM region used by a TXT container [188, 191]. In recent Intel CPUs, the memory controller is integrated on the CPU die, so the SINIT ACM can securely set up the memory controller to reject DMA transfers targeting TXT memory. An Intel chipset datasheet [105] documents an "Intel TXT DMA Protected Range" IIO configuration register.

TXT 不实施 DRAM 加密或 HMAC，因此就像基于 TPM 的设计一样易受物理 DRAM 攻击。此外，早期的 TXT 实现容易受到恶意操作系统编程设备（如网卡）执行向 TXT 容器使用的 DRAM 区域的 DMA 传输的攻击[188,191]。在最近的英特尔 CPU 中，内存控制器集成在 CPU 芯片上，因此 SINIT ACM 可以安全地设置内存控制器，拒绝针对 TXT 内存的 DMA 传输。英特尔芯片组数据表[105]记录了“英特尔 TXT DMA 保护范

围”IIO 配置寄存器。

Early TXT implementations did not measure the SINIT ACM. Instead, the microcode implementing the TXT launch instruction verified that the code module contained an RSA signature by a hard-coded Intel key. SINIT ACM signatures cannot be revoked if vulnerabilities are found, so TXT's software attestation had to be revised when SINIT ACM exploits [190] surfaced. Currently, the SINIT ACM's cryptographic hash is included in the attestation measurement.

早期的 TXT 实施并未测量 SINIT ACM。相反，实施 TXT 启动指令的微代码通过硬编码的 Intel 密钥验证代码模块包含 RSA 签名。如果发现漏洞，则不能撤销 SINIT ACM 签名，因此，当 SINIT ACM 利用[190]时，必须修改 TXT 的软件认证。目前，SINIT ACM 的加密散列包含在认证测量（attestation measurement.）中。

Last, the warm reset performed by the SINIT ACM does not include the software running in System Management Mode (SMM). SMM was designed solely for use by firmware, and is stored in a protected memory area (SMRAM) which should not be accessible to non-SMM software. However, the SMM handler was compromised on multiple occasions [44, 49, 164, 186, 189], and an attacker who obtains SMM execution can access the memory used by TXT's container.

最后，由 SINIT ACM 执行的 warm reset 不包括在系统管理模式（SMM）中运行的软件。SMM 专为固件使用而设计，存储在受 SMM 保护的存储区（SMRAM）中，非 SMM 软件无法访问。然而，SMM 处理程序多次遭到破坏[44,49,164,186,189]，并且获得 SMM 执行的攻击者可以访问 TXT 容器使用的内存。

4.6 The Aegis Secure Processor

The Aegis secure processor [174] relies on a security kernel in the operating system to isolate containers, and includes the kernel's cryptographic hash in the measurement reported by the software attestation signature. [176] argued that Physical Unclonable Functions (PUFs) [56] can be used to endow a secure processor with a tamper-resistant private key, which is required for software attestation. PUFs do not have the fabrication process drawbacks of EEPROM, and are significantly more resilient to physical attacks than e-fuses.

Aegis 安全处理器[174]依靠操作系统中的安全内核来隔离容器，并在软件认证签名报告的测量中包含内核的加密散列。[176]认为物理不可克隆函数（PUFs）[56]可用于赋予安全处理器一个防篡改私钥，这是软件认证所必需的。PUFs 不具有 EEPROM 的制造工艺缺陷，并且比 e-fuses 的物理攻击更具弹性。

Aegis relies on a trusted security kernel to isolate each container from the other software on the computer by configuring the page tables used in address translation. The security kernel is a subset of a typical OS kernel, and handles virtual memory management, processes, and hardware exceptions. As the security kernel is a part of the *trusted code base* (TCB), its cryptographic hash is included in the software attestation measurement. The security kernel uses processor features to isolate itself from the untrusted part of the operating system, such as device drivers.

Aegis 依靠可信的安全内核，通过配置地址转换中使用的页面表，将每个容器与计算机上的其他软件隔离。安全内核是典型操作系统内核的一个子集，用于处理虚拟内存管理、进程和硬件异常。由于安全内核是可信代码库（TCB）的一部分，因此其加密散列包含在软件证明测量中。安全内核使用处理器功能将自己与操作系统的不可信任部分（如设备驱动程序）隔离。

The Aegis memory controller encrypts the cache lines in one memory range, and HMACs the cache lines in one other memory range. The two memory ranges can overlap, and are configurable by the security kernel. Thanks to the two ranges, the memory controller can avoid the latency overhead of cryptographic operations for the DRAM outside containers. Aegis was the first secure processor not vulnerable to physical replay attacks, as it uses a Merkle tree construction [57] to guarantee DRAM freshness. The latency overhead of the Merkle tree is greatly reduced by augmenting the L2 cache with the tree nodes for the cache lines.

Aegis 内存控制器在一个 memory range 对 cache lines 行进行加密，并在另一个 memory range 对 cache lines 行计算 HMACs 值。这两个 memory ranges 可以重叠，并且可以由安全内核配置。由于这两个 ranges，内存控制器可以避免容器外 DRAM 的加密操作的延迟开销。Aegis 是第一个不容易受到物理重放攻击的安全处理器，因为它使用 Merkle 树结构[57]来保证 DRAM 的新鲜度。Merkle 树的延迟开销通过增加带 cache lines 的树节点 L2 cache 大大减少。

Aegis' security kernel allows the OS to page out container memory, but verifies the correctness of the paging operations. The security kernel uses the same encryption and Merkle tree algorithms as the memory controller to guarantee the confidentiality and integrity of the container pages that are swapped out from DRAM. The OS is free to page out container memory, so it can learn a container's memory access patterns, at page granularity. Aegis containers are also vulnerable to cache timing attacks.

Aegis 的安全内核允许 OS page out 容器内存，但验证分页操作的正确性。安全内核使用与内存控制器相同的加密和 Merkle 树算法来保证从 DRAM 中交换出来的容器页面的机密性和完整性。操作系统可以自由 page out 容器内存，因此它可以以页面粒度 learn 容器的内存访问模式。Aegis 容器也容易受到缓存时间攻击（cache timing attacks）。

page out 是虚拟内存从物理内存中拷贝到硬盘

4.7 The Bastion Architecture

The Bastion architecture [31] introduced the use of a trusted hypervisor to provide secure containers to applications running inside unmodified, untrusted operating systems. Bastion's hypervisor ensures that the operating system does not interfere with the secure containers. We only describe Bastion's virtualization extensions to architectures that use nested page tables, like Intel's VMX [181].

Bastion 体系结构[31]引入了使用可信的管理程序来为在未修改的不可信操作系统内运行的应用程序提供安全容器。Bastion 的虚拟机管理程序可确保操作系统不会干扰安全容器。我们只描述 Bastion 对使用嵌套页表的架构的虚拟化扩展，如 Intel 的 VMX [181]。

The hypervisor enforces the containers' desired memory mappings in the OS page tables, as follows. Each Bastion container has a Security Segment that lists the virtual addresses and permissions of all the container's pages, and the hypervisor maintains a Module State Table that stores an inverted page map, associating each physical memory page to its container and virtual address. The processor's hardware page walker is modified to invoke the hypervisor on every TLB miss, before updating the TLB with the address translation result. The hypervisor checks that the virtual address used by the translation matches the expected virtual address associated with the physical address in the Module State Table.

管理程序在 OS 页表中强制执行容器所需的内存映射，如下所示。每个 Bastion 容器都有一个安全段，列出所有容器页面的虚拟地址和权限，虚拟机管理程序维护一个模块

状态表，用于存储反向页面映射，将每个物理内存页面与其容器和虚拟地址相关联。 在使用地址转换结果更新 TLB 之前，修改处理器的硬件页面遍历器以在每个 TLB 未命中时调用管理程序。管理程序检查转换使用的虚拟地址是否与模块状态表中与物理地址关联的预期虚拟地址匹配。

DRAM attack Bastion's cache lines are not tagged with container identifiers. Instead, only TLB entries are tagged. The hypervisor's TLB miss handler sets the container identifier for each TLB entry as it is created. Similarly to XOM and Aegis, the secure processor checks the TLB tag against the current container's identifier on every memory access.

DRAM 攻击 Bastion 的缓存行未标记容器标识符。相反，只标记 TLB 条目。管理程序的 TLB 未命中处理程序在创建时为每个 TLB 条目设置容器标识符。与 XOM 和 Aegis 类似，安全处理器在每次访问内存时检查 TLB 标记与当前容器的标识符。

Bastion offers the same protection against physical DRAM attacks as Aegis does, without the restriction that a container's data must be stored inside a continuous DRAM range. This is accomplished by extending cache lines and TLB entries with flags that enable memory encryption and HMACing. The hypervisor's TLB miss handler sets the flags on TLB entries, and the flags are propagated to cache lines on memory writes.

Bastion 提供与 Aegis 相同的物理 DRAM 攻击保护，不受容器数据必须存储在连续 DRAM 范围内的限制。这是通过使用启用内存加密和 HMACing 的标志扩展高速缓存行和 TLB 条目来实现的。管理程序的 TLB 未命中处理程序在 TLB 条目上设置标志，并且标志在存储器写入时传播到高速缓存行。

The Bastion hypervisor allows the untrusted operating system to evict secure container pages. The evicted pages are encrypted, HMACed, and covered by a Merkle tree maintained by the hypervisor. Thus, the hypervisor ensures the confidentiality, authenticity, and freshness of the swapped pages. However, the ability to freely evict container pages allows a malicious OS to learn a container's memory accesses with page granularity. Furthermore, Bastion's threat model excludes cache timing attacks. Bastion does not trust the platform's firmware, and computes the cryptographic hash of the hypervisor after the firmware finishes playing its part in the booting process. The hypervisor's hash is included in the measurement reported by software attestation.

Bastion 虚拟机管理程序允许不受信任的操作系统驱逐安全容器页面。被驱逐的页面被加密，HMAC 化，并由管理程序维护的 Merkle 树覆盖。因此，管理程序确保交换页面的机密性，真实性和新鲜度。但是，自由驱逐容器页面的能力允许恶意操作系统以页面粒度学习容器的内存访问。此外，Bastion 的威胁模型排除了缓存定时攻击。Bastion 不信任平台的固件，并在固件完成在引导过程中发挥作用后计算管理程序的加密哈希值。管理程序的哈希包含在软件证明报告的度量中。

4.8 Intel SGX in Context

Intel's Software Guard Extensions (SGX) [14, 79, 139] implements secure containers for applications without making any modifications to the processor's critical execution path. SGX does not trust any layer in the computer's software stack (firmware, hypervisor, OS). Instead, SGX's TCB consists of the CPU's microcode and a few privileged containers. SGX introduces an approach to solving some of the issues raised by multi-core processors with a shared, coherent last-level cache.

英特尔的 Software Guard Extensions（SGX）[14,79,139]为应用程序实现了安全容器，而无需对处理器的关键执行路径进行任何修改。SGX 不信任计算机软件堆栈中的任何

层（固件，虚拟机管理程序，操作系统）。相反，SGX 的 TCB 由 CPU 的微代码和一些特权容器组成。SGX 引入了一种方法来解决多核处理器带来的一些问题，这些问题具有共享的，一致的最后一级缓存。

SGX does not extend caches or TLBs with container identity bits, and does not require any security checks during normal memory accesses. As suggested in the TrustZone documentation, SGX always ensures that a core's TLBs only contain entries for the container that it is executing, which requires flushing the CPU core's TLBs when context-switching between containers and untrusted software.

SGX 不会使用容器标识位扩展高速缓存或 TLB，并且在正常的内存访问期间不需要任何安全检查。正如 TrustZone 文档中所建议的那样，SGX 始终确保核心的 TLB 仅包含正在执行的容器的条目，这需要在容器和不受信任的软件之间进行上下文切换时刷新 CPU 核心的 TLB。

SGX follows Bastion's approach of having the untrusted OS manage the page tables used by secure containers. The containers' security is preserved by a TLB miss handler that relies on an inverted page map (the EPCM) to reject address translations for memory that does not belong to the current container. Like Bastion, SGX allows the untrusted operating system to evict secure container pages, in a controlled fashion. After the OS initiates a container page eviction, it must prove to the SGX implementation that it also switched the container out of all cores that were executing its code, effectively performing a very coarse-grained TLB shootdown. SGX's microcode ensures the confidentiality, authenticity, and freshness of each container's evicted pages.

SGX 遵循 Bastion 的方法，让不受信任的操作系统管理安全容器使用的页表。 容器的安全性由 TLB 未命中处理程序保留，该处理程序依赖于反向页面映射（EPCM）来拒绝不属于当前容器的内存的地址转换。与 Bastion 一样，SGX 允许不受信任的操作系统以受控方式驱逐安全容器页面。操作系统启动容器页面驱逐后，它必须向 SGX 实现证明它还将容器从正在执行其代码的所有核心中切换出来，从而有效地执行非常粗粒度的 TLB 击落。SGX 的微码确保了每个容器被逐出的页面的机密性，真实性和新鲜度。

Like Bastion's hypervisor. However, SGX relies on a version-based Merkle tree, inspired by Aegis [174], and adds an innovative twist that allows the operating system to dynamically shape the Merkle tree. SGX also shares Bastion's and Aegis' vulnerability to memory access pattern leaks, namely a malicious OS can directly learn a container's memory accesses at page granularity, and any piece of software can perform cache timing attacks.

就像 Bastion 的 hypervisor 一样。 然而，SGX 依赖于基于版本的 Merkle 树，受到 Aegis [174]的启发，并增加了一种创新的扭曲，允许操作系统动态塑造 Merkle 树。 SGX 还分享了 Bastion 和 Aegis 对内存访问模式泄漏的漏洞，即恶意操作系统可以直接以页面粒度学习容器的内存访问，并且任何软件都可以执行缓存定时攻击。

SGX's software attestation is implemented using Intel's Enhanced Privacy ID (EPID) group signature scheme [26], which is too complex for a microcode implementation. Therefore, SGX relies on an assortment of privileged containers that receive direct access to the SGX processor's hardware keys. The privileged containers are signed using an Intel private key whose corresponding public key is hard-coded into the SGX microcode, similarly to TXT's SINIT ACM.

SGX 的软件认证是使用英特尔的增强型隐私 ID（EPID）组签名方案[26]实现的，这对于微码实现而言过于复杂。 因此，SGX 依赖于各种特权容器，这些容器可以直接访

问 SGX 处理器的硬件密钥。 特权容器使用英特尔私钥签名，其相应的公钥被硬编码到 SGX 微码中，类似于 TXT 的 SINIT ACM。

As SGX does not protect against cache timing attacks, the privileged enclave's authors cannot use data dependent memory accesses. For example, cache attacks on the Quoting Enclave, which computes attestation signatures, would provide an attack with a processor's EPID signing key and completely compromise SGX.

由于 SGX 不能防止缓存定时攻击，特权飞地的作者不能使用数据相关的内存访问。例如，计算证明签名的 Quoting Enclave 上的缓存攻击将使用处理器的 EPID 签名密钥提供攻击并完全破坏 SGX。

Intel's documentation states that SGX guarantees DRAM confidentiality, authentication, and freshness by virtue of a Memory Encryption Engine (MEE). The MEE is informally described in an ISCA 2015 tutorial [103], and appears to lack a formal specification. In the absence of further information, we assume that SGX provides the same protection against physical DRAM attacks that Aegis and Bastion provide.

英特尔的文档指出，凭借内存加密引擎（MEE），SGX 可以保证 DRAM 的机密性，身份验证和新鲜度。MEE 在 ISCA 2015 教程[103]中进行了非正式描述，似乎缺乏正式的规范。在没有进一步信息的情况下，我们假设 SGX 提供了与 Aegis 和 Bastion 提供的物理 DRAM 攻击相同的保护。

5 SGX PROGRAMMING MODEL

The central concept of SGX is the *enclave*, a protected environment that contains the code and data pertaining to a security-sensitive computation.

SGX 的核心概念是 *enclave*，一个包含与安全敏感计算有关的代码和数据的受保护环境。

SGX-enabled processors provide trusted computing by isolating each enclave's environment from the untrusted software outside the enclave, and by implementing a software attestation scheme that allows a remote party to authenticate the software running inside an enclave. SGX's isolation mechanisms are intended to protect the confidentiality and integrity of the computation performed inside an enclave from attacks coming from malicious software executing on the same computer, as well as from a limited set of physical attacks.

SGX-enabled 的处理器通过将 enclave 的环境与 enclave 外的不受信任的软件隔离，并通过实施允许远程方认证在 enclave 内运行的软件认证方案来提供可信计算。SGX 的隔离机制旨在保护在 enclave 内执行的计算的机密性和完整性，使其免受来自同一台计算机上执行的恶意软件的攻击以及有限的物理攻击。

This section summarizes the SGX concepts that make up a mental model which is sufficient for programmers to author SGX enclaves and to add SGX support to existing system software. Unless stated otherwise, the information in this section is backed up by Intel's Software Developer Manual (SDM). The following section builds on the concepts introduced here to fill in some of the missing pieces in the manual, and analyzes some of SGX's security properties.

本节总结了 SGX 的概念，这些概念组成了一个 mental 模型，足以让程序员编写 SGX enclave 并为现有系统软件添加 SGX 支持。除非另有说明，本节中的信息由英特尔的软件开发人员手册（SDM）提供备份。以下部分以此处介绍的概念为基础，以填写手册中的一些缺失部分，并分析 SGX 的一些安全属性。

5.1 SGX Physical Memory Organization

The enclaves' code and data is stored in Processor Reserved Memory (PRM), which is a subset of DRAM that cannot be directly accessed by other software, including system software and SMM code. The CPU's integrated memory controllers (2.9.3) also reject DMA transfers targeting the PRM, thus protecting it from access by other peripherals.

Enclave 的代码和数据存储在处理器保护内存（PRM）中，PRM 是 DRAM 的子集，其不能由其他软件直接访问，包括系统软件和 SMM 代码。CPU 的集成内存控制器（2.9.3）也拒绝以 PRM 为目标的 DMA（Direct Memory Access，直接内存存取 ）传输，从而保护其免受其他外设的访问。

The PRM is a continuous range of memory whose bounds are configured using a base and a mask register with the same semantics as a variable memory type range (2.11.4). Therefore, the PRM's size must be an integer power of two, and its start address must be aligned to the same power of two. Due to these restrictions, checking if an address belongs to the PRM can be done very cheaply in hardware, using the circuit outlined in 2.11.4.

PRM 是一个连续范围的内存，其边界使用与变量存储器类型范围（2.11.4）具有相同语义的基址和掩码寄存器进行配置。因此，PRM 的大小必须是 2 的整数次幂，并且其起始地址必须对齐到相同的幂次。 由于这些限制，使用 2.11.4 中概述的电路，检查地址是否属于 PRM 在硬件中代价很低。

The SDM does not describe the PRM and the PRM range registers (PRMRR). These concepts are documented in the SGX manuals [95, 99] and in one of the SGX papers [139]. Therefore, the PRM is a micro-architectural detail that might change in future implementations of SGX. Our security analysis of SGX relies on implementation details surrounding the PRM, and will have to be re-evaluated for SGX future implementations.

SDM 没有描述 PRM 和 PRM 范围寄存器（PRMRR）。这些概念在 SGX 手册[95,99]和 SGX 论文[139]中有记载。因此，PRM 是一种在未来 SGX 的实施中可能会发生变化微观架构细节。我们对 SGX 的安全分析依赖于围绕 PRM 的实施细节，并且必须重新评估 SGX 未来的实施。

5.1.1 The Enclave Page Cache (EPC)

The contents of enclaves and the associated data structures are stored in the Enclave Page Cache (EPC), which is a subset of the PRM, as shown in Figure 60.

Enclave 的内容和相关的数据结构存储在 Enclave Page Cache（EPC）中，EPC 是 PRM 的子集，如图 60 所示。

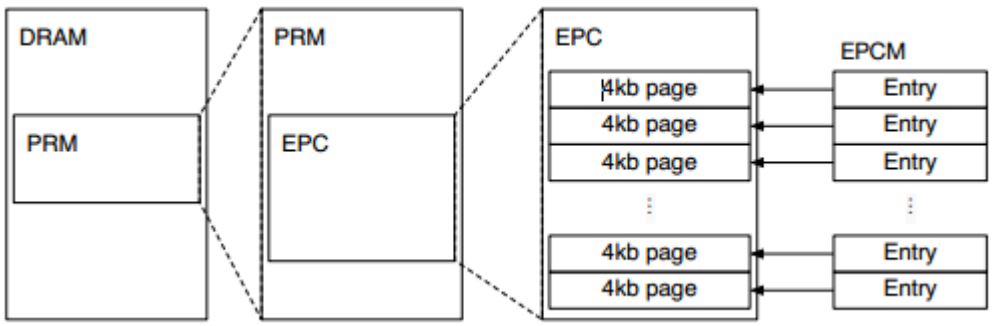


Figure 60: Enclave data is stored into the EPC, which is a subset of the PRM. The PRM is a contiguous range of DRAM that cannot be accessed by system software or peripherals.

图 60：Enclave 数据存储在 EPC 中，EPC 是 PRM 的一个子集。PRM 是连续范围的

DRAM，无法通过系统软件或外设访问。

The SGX design supports having multiple enclaves on a system at the same time, which is a necessity in multi-process environments. This is achieved by having the EPC split into 4 KB pages that can be assigned to different enclaves. The EPC uses the same page size as the architecture's address translation feature (2.5). This is not a coincidence, as future sections will reveal that the SGX implementation is tightly coupled with the address translation implementation.

SGX 设计支持在一个系统中同时拥有多个 enclaves，这在多进程环境中是必不可少的。这是通过将 EPC 分成 4 KB 页面来实现的，可以将其分配到不同的 enclaves。EPC 使用与体系结构的地址转换功能（2.5）相同的页面大小。这并非巧合，因为未来部分将揭示 SGX 实施与地址转换实施密切相关。

The EPC is managed by the same system software that manages the rest of the computer's physical memory. The system software, which can be a hypervisor or an OS kernel, uses SGX instructions to allocate unused pages to enclaves, and to free previously allocated EPC pages. The system software is expected to expose enclave creation and management services to application software.

EPC 由管理计算机其余物理内存的相同系统软件管理。系统软件可以是管理程序或操作系统内核，它使用 SGX 指令将未使用的页面分配给 enclave，并释放先前分配的 EPC 页面。系统软件向应用软件提供 enclave 创建和管理服务。

Non-enclave software cannot directly access the EPC, as it is contained in the PRM. This restriction plays a key role in SGX's enclave isolation guarantees, but creates an obstacle when the system software needs to load the initial code and data into a newly created enclave. The SGX design solves this problem by having the instructions that allocate an EPC page to an enclave also initialize the page. Most EPC pages are initialized by copying data from a non-PRM memory page.

非 enclave 软件不能直接访问 EPC，因为它包含在 PRM 中。此限制在 SGX 的 enclave 隔离保证中起着关键作用，但是当系统软件需要将初始代码和数据加载到新创建的 enclave 时会产生障碍。SGX 设计通过将 EPC 页面分配给 enclave 的指令并初始化页面来解决这个问题。大多数 EPC 页面是通过从非 PRM 存储器页面复制数据来初始化的。

5.1.2 The Enclave Page Cache Map (EPCM)

The SGX design expects the system software to allocate the EPC pages to enclaves. However, as the system software is not trusted, SGX processors check the correctness of the system software's allocation decisions, and refuse to perform any action that would compromise SGX's security guarantees. For example, if the system software attempts to allocate the same EPC page to two enclaves, the SGX instruction used to perform the allocation will fail.

SGX 设计期望系统软件将 EPC 页面分配给安全区。但是，由于系统软件不可信，SGX 处理器会检查系统软件分配决策的正确性，并拒绝执行任何可能危及 SGX 安全保证的行为。例如，如果系统软件试图将相同的 EPC 页分配给两个 enclaves，则用于执行分配的 SGX 指令将失效。

In order to perform its security checks, SGX records some information about the system software's allocation decisions for each EPC page in the Enclave Page Cache Map (EPCM). The EPCM is an array with one entry per EPC page, so computing the address of a page's EPCM entry only requires a bitwise shift operation and an addition.

为了执行安全检查，SGX 在 Enclave Page Cache Map（EPCM）中记录了系统软件对于每个 EPC 页面的分配决定的信息。EPCM 是一个为每个 EPC 页面保留一个条目（entry）的阵列（array），因此计算页面的 EPCM 条目的地址只需要逐位移位操作和一个加法操作。

The EPCM's contents is only used by SGX's security checks. Under normal operation, the EPCM does not generate any software-visible behavior, and enclave authors and system software developers can mostly ignore it. Therefore, the SDM only describes the EPCM at a very high level, listing the information contained within and noting that the EPCM is "trusted memory". The SDM does not disclose the storage medium or memory layout used by the EPCM.

EPCM 的内容仅供 SGX 的安全检查使用。在正常运行情况下，EPCM 不会产生任何软件可见行为，并且 enclave authors 和系统软件开发人员可以忽略它。 因此，SDM 仅在高层次上描述 EPCM，列出其中包含的信息并注意到 EPCM 是"可信内存"。 SDM 未披露 EPCM 使用的存储介质或存储器布局。

The EPCM uses the information in Table 13 to track the ownership of each EPC page. We defer a full discussion of the EPCM to a later section, because its contents is intimately coupled with all of SGX's features, which will be described over the next few sections.

EPCM 使用表 13 中的信息来跟踪每个 EPC 页面的所有权（ownership）。我们将 EPCM 的全面讨论推迟到后面的部分，因为它的内容与 SGX 的所有功能密切相关，将在接下来的几节中进行介绍。

Field	Bits	Description
VALID	1	0 for un-allocated EPC pages
PT	8	page type
ENCLAVESECS		identifies the enclave owning the page

Table 13: The fields in an EPCM entry that track the ownership of pages.

The SGX instructions that allocate an EPC page set the VALID bit of the corresponding EPCM entry to 1, and refuse to operate on EPC pages whose VALID bit is already set.

分配 EPC 页的 SGX 指令将相应 EPCM 条目的 VALID 位设置为 1，并拒绝在 VALID 位已设置的 EPC 页上的操作。

The instruction used to allocate an EPC page also determines the page's intended usage, which is recorded in the page type (PT) field of the corresponding EPCM entry. The pages that store an enclave's code and data are considered to have a regular type (PT REG in the SDM). The pages dedicated to the storage of SGX's supporting data structures are tagged with special types. For example, the PT SECS type identifies pages that hold SGX Enclave Control Structures, which will be described in the following section. The other EPC page types will be described in future sections.

用于分配 EPC 页面的指令还决定页面的预期用途，该用途记录在相应 EPCM 条目的页面类型（PT）字段中。存储 enclave 代码和数据的页面被认为具有常规类型（SDM 中的 PT REG）。专门用于存储 SGX 支持数据结构的页面被标记为特殊类型。例如，PT SECS 类型标识持有 SGX Enclave 控制结构的页面，这将在以下部分进行描述。其他 EPC 页面类型将在以后的章节中介绍。

Last, a page's EPCM entry also identifies the enclave that owns the EPC page. This

information is used by the mechanisms that enforce SGX's isolation guarantees to prevent an enclave from accessing another enclave's private information. As the EPCM identifies a single owning enclave for each EPC page, it is impossible for enclaves to communicate via shared memory using EPC pages. Fortunately, enclaves can share untrusted non-EPC memory, as will be discussed in 5.2.3.

最后，页面的 EPCM 条目还标识拥有 EPC 页面的 enclave。强制 SGX 的隔离保证机制使用此信息来防止安全区访问其他安全区的私人信息。由于 EPCM 为每个 EPC 页面标识一个单独的 owning enclave，因此 enclave 间不可能通过使用 EPC 页面的共享内存进行间通信。幸运的是，enclave 可以共享不可信的非 EPC 内存，这将在 5.2.3 节中讨论。

5.1.3 The SGX Enclave Control Structure (SECS)

SGX stores per-enclave metadata in a SGX Enclave Control Structure (SECS) associated with each enclave. Each SECS is stored in a dedicated EPC page with the page type PT SECS. These pages are not intended to be mapped into any enclave's address space, and are exclusively used by the CPU's SGX implementation.

SGX 将每个 enclave 元数据存储在与每个 enclave 相关的 SGX Enclave Control Structure（SECS）中。每个 SECS 都存储在页面类型为 PT SECS 的专用 EPC 页面中。这些页面不映射到任何 enclave 的地址空间，并且仅由 CPU 的 SGX 实现使用。

An enclave's identity is almost synonymous to its SECS. The first step in bringing an enclave to life allocates an EPC page to serve as the enclave's SECS, and the last step in destroying an enclave de-allocates the page holding its SECS. The EPCM entry field identifying the enclave that owns an EPC page points to the enclave's SECS. The system software uses the virtual address of an enclave's SECS to identify the enclave when invoking SGX instructions.

Enclave 的身份 synonymous to（同义）其 SECS。让 enclave 生效的第一步是分配 EPC 页面作为 enclave 的 SECS，而销毁安全区的最后一步是 de-allocate 持有 SECS 的页面。识别拥有 EPC 页面的安全区的 EPCM entry 字段指向安全区的 SECS。当调用 SGX 指令时，系统软件使用 enclave 的 SECS 的虚拟地址来识别 enclave。

All SGX instructions take virtual addresses as their inputs. Given that SGX instructions use SECS addresses to identify enclaves, the system software must create entries in its page tables pointing to the SECS of the enclaves it manages. However, the system software cannot access any SECS page, as these pages are stored in the PRM. SECS pages are not intended to be mapped inside their enclaves' virtual address spaces, and SGX-enabled processors explicitly prevent enclave code from accessing SECS pages.

所有 SGX 指令都将虚拟地址作为其输入。鉴于 SGX 指令使用 SECS 地址来识别 enclave，系统软件必须在其页表中创建指向其管理的 enclave 的 SECS 条目。但是，系统软件无法访问任何 SECS 页面，因为这些页面存储在 PRM 中。SECS 页面不打算映射到它们的 enclave 的虚拟地址空间中，并且 SGX 激活的的处理器明确阻止 enclave 代码访问 SECS 页面。

This seemingly arbitrary limitation is in place so that the SGX implementation can store sensitive information in the SECS, and be able to assume that no potentially malicious software will access that information. For example, the SDM states that each enclave's measurement is stored in its SECS. If software would be able to modify an enclave's measurement, SGX's software attestation scheme would provide no security assurances.

这种看似任意的限制已经到位，因此 SGX 实施可以将敏感信息存储在 SECS 中，

并且能够假定没有潜在恶意软件将访问该信息。例如，SDM 规定每个 enclave 的 measurement 都存储在其 SECS 中。如果软件能够修改 enclave 的 measurement，SGX 的软件认证计划将不提供安全保证。

The SECS is strongly coupled with many of SGX's features. Therefore, the pieces of information that make up the SECS will be gradually introduced as the different aspects of SGX are described.

SECS 与 SGX 的许多功能紧密结合。因此，构成 SECS 的信息将随着 SGX 的不同方面的描述而逐渐引入。

5.2 The Memory Layout of an SGX Enclave

SGX was designed to minimize the effort required to convert application code to take advantage of enclaves. History suggests this is a wise decision, as a large factor in the continue dominance of the Intel architecture is its ability to maintain backward compatibility. To this end, SGX enclaves were designed to be conceptually similar to the leading software modularization construct, dynamically loaded libraries, which are packaged as *.so* files on Unix, and *.dll* files on Windows.

SGX 旨在最大限度地减少转换应用程序代码以利用飞地所需的工作量。 历史表明这是一个明智的决定，因为英特尔架构继续占据主导地位的一个重要因素是它能够保持向后兼容性。 为此，SGX 飞地的设计在概念上类似于领先的软件模块化构造，动态加载的库，在 Unix 上打包为.so 文件，在 Windows 上打包为.dll 文件。

For simplicity, we describe the interaction between enclaves and non-enclave software assuming that each enclave is used by exactly one application process, which we shall refer to as the enclave's *host process*. We do note, however, that the SGX design does not explicitly prohibit multiple application processes from sharing an enclave.

为简单起见，我们描述了飞地和非飞地软件之间的相互作用，假设每个飞地仅由一个应用程序进程使用，我们将其称为飞地的主进程。 但是，我们注意到 SGX 设计没有明确禁止多个申请流程共享飞地。

5.2.1 The Enclave Linear Address Range (ELRANGE)

Each enclave designates an area in its virtual address space, called the **enclave linear address range** (ELRANGE), which is used to map the code and the sensitive data stored in the enclave's EPC pages. The virtual address space outside ELRANGE is mapped to access non-EPC memory via the same virtual addresses as the enclave's host process, as shown in Figure 61.

每个安区都在其虚拟地址空间中指定一个区域，称为包围线性地址范围（ELRANGE），用于映射存储在飞地 EPC 页面中的代码和敏感数据。 ELRANGE 外部的虚拟地址空间映射为通过与安全区主机进程相同的虚拟地址访问非 EPC 内存，如图 61 所示。

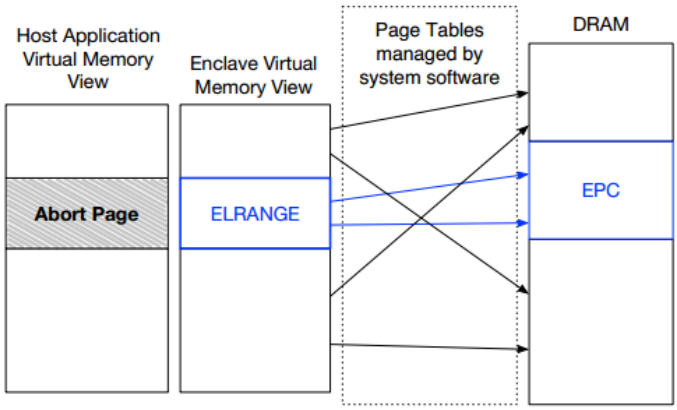


Figure 61*:* An enclave's EPC pages are accessed using a dedicated region in the enclave's virtual address space, called ELRANGE. The rest of the virtual address space is used to access the memory of the host process. The memory mappings are established using the page tables managed by system software.

图 61：使用安全区虚拟地址空间中的专用区域（称为 ELRANGE）访问安全区的 EPC 页面。 虚拟地址空间的其余部分用于访问主机进程的内存。 使用由系统软件管理的页表来建立存储器映射。

The SGX design guarantees that the enclave's memory accesses inside ELRANGE obey the virtual memory abstraction (2.5.1), while memory accesses outside ELRANGE receive no guarantees. Therefore, enclaves must store all their code and private data inside ELRANGE, and must consider the memory outside ELRANGE to be an untrusted interface to the outside world.

SGX 设计保证了 ELRANGE 内部的内存访问遵循虚拟内存抽象（2.5.1），而 ELRANGE 外部的内存访问不受保证。 因此，安全区必须将所有代码和私有数据存储在 ELRANGE 中，并且必须将 ELRANGE 外部的内存视为外部世界的不可信任接口。

The word "linear" in ELRANGE references the linear addresses produced by the vestigial segmentation feature (2.7) in the 64-bit Intel architecture. For most purposes, "linear" can be treated as a synonym for "virtual".

ELRANGE 中的"线性"一词引用了 64 位英特尔架构中残留分段功能（2.7）产生的线性地址。 对于大多数目的，"线性"可以被视为"虚拟"的同义词。

ELRANGE is specified using a base (the BASEADDR field) and a size (the SIZE) in the enclave's SECS (x 5.1.3). ELRANGE must meet the same constraints as a variable memory type range (2.11.4) and as the PRM range (5.1), namely the size must be a power of 2, and the base must be aligned to the size. These restrictions are in place so that the SGX implementation can inexpensively check whether an address belongs to an enclave's ELRANGE, in either hardware (x 2.11.4) or software.

使用基础（BASEADDR 字段）和安全区 SECS（x 5.1.3）中的大小（SIZE）指定 ELRANGE。 ELRANGE 必须满足与可变存储器类型范围（2.11.4）和 PRM 范围（5.1）相同的约束，即大小必须是 2 的幂，并且基数必须与大小对齐。 这些限制已经到位，因此 SGX 实现可以在硬件（x 2.11.4）或软件中低成本地检查地址是否属于安全区的 ELRANGE。

When an enclave represents a dynamic library, it is natural to set ELRANGE to the memory range reserved for the library by the loader. The ability to access non-enclave memory

from enclave code makes it easy to reuse existing library code that expects to work with pointers to memory buffers managed by code in the host process.

当一个飞地代表一个动态库时，很自然地将 ELRANGE 设置为加载器为库保留的内存范围。 从安全区代码访问非安全区内存的能力使得重用现有的库代码变得容易，这些代码期望使用指向由主机进程中的代码管理的内存缓冲区的指针。

Non-enclave software cannot access PRM memory. A memory access that resolves inside the PRM results in an aborted transaction, which is undefined at an architectural level, On current processors, aborted writes are ignored, and aborted reads return a value whose bits are all set to 1. This comes into play in the scenario described above, where an enclave is loaded into a host application process as a dynamically loaded library. The system software maps the enclave's code and data in ELRANGE into EPC pages. If application software attempts to access memory inside ELRANGE, it will experience the abort transaction semantics. The current semantics do not cause the application to crash (e.g., due to a Page Fault), but also guarantee that the host application will not be able to tamper with the enclave or read its private information.

非飞地软件无法访问 PRM 内存。在 PRM 内部解析的内存访问会导致中止事务，在体系结构级别上未定义，在当前处理器上，忽略中止写入，并且中止读取返回其位全部设置为 1 的值。上面描述的场景，其中安塞区作为动态加载的库加载到主机应用程序进程中。 系统软件将 ELRANGE 中的飞地代码和数据映射到 EPC 页面。 如果应用程序软件试图访问 ELRANGE 内部的内存，它将体验中止事务语义。当前语义不会导致应用程序崩溃（例如，由于页面错误），但也保证主机应用程序将无法篡改安全区或读取其私有信息。

5.2.2 SGX Enclave Attributes

The execution environment of an enclave is heavily influenced by the value of the ATTRIBUTES field in the enclave's SECS (x 5.1.3). The rest of this work will refer to the field's sub-fields, shown in Table 14, as enclave attributes.

Field	Bits	Description
DEBUG	1	Opts into enclave debugging features.
XFRM	64	The value of XCR0 (§ 2.6) while this enclave's code is executed.
MODE64BIT	1	Set for 64-bit enclaves.

Table 14: An enclave's attributes are the sub-fields in the ATTRIBUTES field of the enclave's SECS. This table shows a subset of the attributes defined in the SGX documentation.

表 14：飞地的属性是飞地 SECS 的 ATTRIBUTES 字段中的子字段。此表显示了 SGX 文档中定义的属性的子集。

The most important attribute, from a security perspective, is the DEBUG flag. When this flag is set, it enables the use of SGX's debugging features for this enclave. These debugging features include the ability to read and modify most of the enclave's memory. Therefore, DEBUG should only be set in a development environment, as it causes the enclave to lose all the SGX security guarantees.

从安全角度来看，最重要的属性是 DEBUG 标志。 设置此标志后，它可以为此安全

区使用SGX的调试功能。这些调试功能包括读取和修改大部分飞地内存的功能。因此，DEBUG 只应设置在开发环境中，因为它会导致飞地失去所有 SGX 安全保证。

SGX guarantees that enclave code will always run with the XCR0 register (x 2.6) set to the value indicated by extended features request mask (XFRM). Enclave authors are expected to use XFRM to specify the set of architectural extensions enabled by the compiler used to produce the enclave's code. Having XFRM be explicitly specified allows Intel to design new architectural extensions that change the semantics of existing instructions, such as Memory Protection Extensions (MPX), without having to worry about the security implications on enclave code that was developed without an awareness of the new features.

SGX 保证飞地代码始终在 XCR0 寄存器（x 2.6）设置为扩展功能请求掩码（XFRM）指示的值的情况下运行。Enclave 作者应该使用 XFRM 来指定用于生成飞地代码的编译器启用的架构扩展集。明确指定 XFRM 允许英特尔设计新的体系结构扩展，以更改现有指令的语义，例如内存保护扩展（MPX），而不必担心在不了解新功能的情况下开发的安全区代码的安全隐患。

The MODE64BIT flag is set to true for enclaves that use the 64-bit Intel architecture. From a security standpoint, this flag should not even exist, as supporting a secondary architecture adds unnecessary complexity to the SGX implementation, and increases the probability that security vulnerabilities will creep in. It is very likely that the 32-bit architecture support was included due to Intel's strategy of offering extensive backwards compatibility, which has paid off quite well so far.

对于使用 64 位 Intel 架构的安全区，MODE64BIT 标志设置为 true。从安全角度来看，这个标志甚至不应该存在，因为支持辅助架构会给 SGX 实现增加不必要的复杂性，并增加安全漏洞蔓延的可能性。很可能 32 位架构支持包括在内 英特尔提供广泛的向后兼容性的策略，到目前为止已经取得了相当好的成绩。

In the interest of mental sanity, this work does not analyze the behavior of SGX for enclaves whose MODE64BIT flag is cleared. However, a security researcher who wishes to find vulnerabilities in SGX might study this area.

为了精神健全，这项工作不分析 SGX 对其 MODE64BIT 标志被清除的飞地的行为。但是，希望在 SGX 中发现漏洞的安全研究人员可能会研究这个领域。

Last, the INIT flag is always false when the enclave's SECS is created. The flag is set to true at a certain point in the enclave lifecycle, which will be summarized in 5.3.

最后，当创建安全区的 SECS 时，INIT 标志始终为 false。该标志在安全区生命周期中的某个点设置为 true，将在 5.3 中进行汇总。

5.2.3 Address Translation for SGX Enclaves

Under SGX, the operating system and hypervisor are still in full control of the page tables and EPTs, and each enclave's code uses the same address translation process and page tables (2.5) as its host application. This minimizes the amount of changes required to add SGX support to existing system software. At the same time, having the page tables managed by untrusted system software opens SGX up to the address translation attacks described in 3.7. As future sections will reveal, a good amount of the complexity in SGX's design can be attributed to the need to prevent these attacks.

在 SGX 下，操作系统和管理程序仍然完全控制页表和 EPT，每个飞地的代码使用相同的地址转换过程和页表（2.5 节）作为其主机应用程序。这可以最大限度地减少为

现有系统软件添加 SGX 支持所需的更改量。同时，让不受信任的系统软件管理的页表打开 SGX 直到 3.7 节中描述的地址转换攻击。正如未来部分将揭示的那样，SGX 设计中的大量复杂性可归因于防止这些攻击的需要。

SGX's active memory mapping attacks defense mechanisms revolve around ensuring that each EPC page can only be mapped at a specific virtual address (2.7). When an EPC page is allocated, its intended virtual address is recorded in the EPCM entry for the page, in the ADDRESS field.

SGX 的主动内存映射攻击防御机制围绕着确保每个 EPC 页面只能映射到特定的虚拟地址（2.7 节）。分配 EPC 页面时，其预期的虚拟地址将记录在 ADDRESS 字段的页面的 EPCM 条目中。

When an address translation (2.5) result is the physical address of an EPC page, the CPU ensures that the virtual address given to the address translation process matches the expected virtual address recorded in the page's EPCM entry. SGX also protects against some passive memory mapping attacks and fault injection attacks by ensuring that the access permissions of each EPC page always match the enclave author's intentions. The access permissions for each EPC page are specified when the page is allocated, and recorded in the readable (R), writable (W), and executable (X) fields in the page's EPCM entry, shown in Table 15.

当地址转换（2.5 节）结果是 EPC 页面的物理地址时，CPU 确保给予地址转换过程的虚拟地址与页面的 EPCM 条目中记录的预期虚拟地址匹配。SGX 还通过确保每个 EPC 页面的访问权限始终与 enclave author 的意图相匹配来防止一些被动内存映射攻击和故障注入攻击。分配页面时指定每个 EPC 页面的访问权限，并记录在页面的 EPCM 条目中的可读（R），可写（W）和可执行（X）字段中，如表 15 所示。

Field	Bits	Description
ADDRESS	48	the virtual address used to access this page
R	1	allow reads by enclave code
W	1	allow writes by enclave code
X	1	allow execution of code inside the page, inside enclave

Table 15: The fields in an EPCM entry that indicate the enclave's intended virtual memory layout.

表 15：EPCM 条目中的字段，指示安全区的预期虚拟内存 layout。

When an address translation (2.5) resolves into an EPC page, the corresponding EPCM entry's fields override the access permission attributes (2.5.3) specified in the page tables. For example, the W field in the EPCM entry overrides the writable (W) attribute, and the X field overrides the disable execution (XD) attribute.

当地址转换（2.5 节）解析为 EPC 页面时，相应的 EPCM 条目的字段将覆盖页表中指定的访问权限属性（2.5.3 节）。例如，EPCM 条目中的 W 字段将覆盖可写（W）属性，X 字段将覆盖禁用执行（XD）属性。

It follows that an enclave author must include memory layout information along with the enclave, in such a way that the system software loading the enclave will know the expected virtual memory address and access permissions for each enclave page. In return, the SGX design guarantees to the enclave authors that the system software, which manages the page tables and EPT, will not be able to set up an enclave's virtual address space in a manner that is

inconsistent with the author's expectations.

因此，enclave author 必须在飞地中包括存储器 layout 信息，这样加载飞地的系统软件将知道每个飞地页面的预期虚拟存储器地址和访问权限。作为回报，SGX 设计向 enclave author 确保管理页面表和 EPT 的系统软件将无法以与 author 期望不一致的方式设置飞地的虚拟地址空间。

The .so and .dll file formats, which are SGX's intended enclave delivery vehicles, already have provisions for specifying the virtual addresses that a software module was designed to use, as well as the desired access permissions for each of the module's memory areas.

.so 和.dll 文件格式是 SGX 预期的飞地交付工具，已经有规定软件模块设计使用的虚拟地址，以及每个模块内存区域所需的访问权限。

Last, a SGX-enabled CPU will ensure that the virtual memory inside ELRANGE (5.2.1) is mapped to EPC pages. This prevents the system software from carrying out an address translation attack where it maps the enclave's entire virtual address space to DRAM pages outside the PRM, which do not trigger any of the checks above, and can be directly accessed by the system software.

最后，启用 SGX 的 CPU 将确保 ELRANGE（5.2.1）内的虚拟内存映射到 EPC 页面。这可以防止系统软件执行地址转换攻击，其中它将飞地的整个虚拟地址空间映射到 PRM 外部的 DRAM 页面，这些 DRAM 页面不会触发上述任何检查，并且可以由系统软件直接访问。

5.2.4 The Thread Control Structure (TCS)

The SGX design fully embraces multi-core processors. It is possible for multiple logical processors (2.9.3) to concurrently execute the same enclave's code at the same time, via different threads.

SGX 设计完全采用多核处理器。多个逻辑处理器（2.9.3 节）可以通过不同的线程同时执行相同的飞地代码。

The SGX implementation uses a Thread Control Structure (TCS) for each logical processor that executes an enclave's code. It follows that an enclave's author must provision at least as many TCS instances as the maximum number of concurrent threads that the enclave is intended to support.

SGX 实现为执行飞地代码的每个逻辑处理器使用线程控制结构（TCS）。因此，enclave's author 必须至少提供与飞地旨在支持的最大并发线程数一样多的 TCS 实例。

Each TCS is stored in a dedicated EPC page whose EPCM entry type is PT TCS. The SDM describes the first few fields in the TCS. These fields are considered to belong to the architectural part of the structure, and therefore are guaranteed to have the same semantics on all the processors that support SGX. The rest of the TCS is not documented.

每个 TCS 存储在 EPCM 条目类型为 PT TCS 的专用 EPC 页面中。SDM 描述了 TCS 中的前几个字段。这些字段被认为属于结构的体系结构部分，因此保证在支持 SGX 的所有处理器上具有相同的语义。TCS 的其余部分未记录在案。

The contents of an EPC page that holds a TCS cannot be directly accessed, even by the code of the enclave that owns the TCS. This restriction is similar to the restriction on accessing EPC pages holding SECS instances. However, the architectural fields in a TCS can be read by enclave debugging instructions.

即使拥有 TCS 的飞地代码，也无法直接访问包含 TCS 的 EPC 页面的内容。此限制

类似于访问持有 SECS 实例的 EPC 页面的限制。但是，可以通过飞地调试指令读取 TCS 中的架构字段。

The architectural fields in the TCS lay out the context switches (2.6) performed by a logical processor when it transitions between executing non-enclave and enclave code.

TCS 中的架构字段布置了逻辑处理器在执行非飞地和飞地代码之间转换时执行的上下文切换（2.6）。

For example, the OENTRY field specifies the value loaded in the instruction pointer (RIP) when the TCS is used to start executing enclave code, so the enclave author has strict control over the entry points available to enclave's host application. Furthermore, the OFSBASGX and OFSBASGX fields specify the base addresses loaded in the FS and GS segment registers (2.7), which typically point to Thread Local Storage (TLS).

例如，当 TCS 用于开始执行安全区代码时，OENTRY 字段指定在指令指针（RIP）中加载的值，因此 enclave author 可以严格控制安全区主机应用程序可用的 entry points。此外，OFSBASGX 和 OFSBASGX 字段指定 FS 和 GS 段寄存器（2.7 节）中加载的基址，通常指向线程本地存储（TLS）。

5.2.5 The State Save Area (SSA)

When the processor encounters a hardware exception (2.8.2), such as an interrupt (2.12), while executing the code inside an enclave, it performs a privilege level switch (2.8.2) and invokes a hardware exception handler provided by the system software. Before executing the exception handler, however, the processor needs a secure area to store the enclave code's execution context (2.6), so that the information in the execution context is not revealed to the untrusted system software.

当处理器遇到硬件异常（2.8.2），例如中断（2.12）时，在执行安全区内的代码时，它执行特权级别切换（2.8.2）并调用系统提供的硬件异常处理程序 软件。 但是，在执行异常处理程序之前，处理器需要一个安全区域来存储安全区代码的执行上下文（2.6），以便执行上下文中的信息不会泄露给不受信任的系统软件。

In the SGX design, the area used to store an enclave thread's execution context while a hardware exception is handled is called a State Save Area (SSA), illustrated in Figure 62. Each TCS references a contiguous sequence of SSAs. The offset of the SSA array (OSSA) field specifies the location of the first SSA in the enclave's virtual address space. The number of SSAs (NSSA) field indicates the number of available SSAs.

在 SGX 设计中，处理硬件异常时用于存储飞地线程执行上下文的区域称为状态保存区域（SSA），如图 62 所示。每个 TCS 引用一个连续的 SSA 序列。 SSA 阵列（OSSA）字段的偏移量指定了飞地的虚拟地址空间中第一个 SSA 的位置。SSA（NSSA）字段的数量表示可用 SSA 的数量。

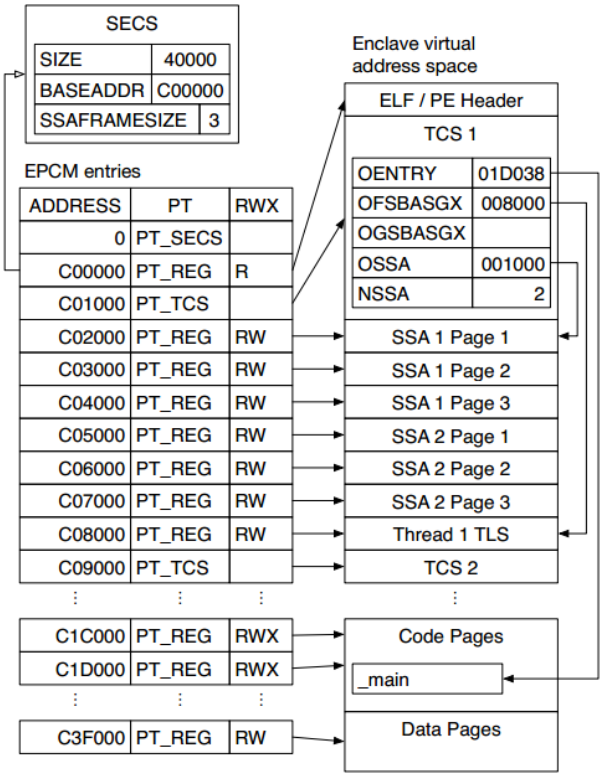


Figure 62: A possible layout of an enclave's virtual address space. Each enclave has a SECS, and one TCS per supported concurrent thread. Each TCS points to a sequence of SSAs, and specifies initial values for RIP and for the base addresses of FS and GS.

图 62：安全区虚拟地址空间的可能布局。每个安全区都有一个 SECS，每个支持的并发线程有一个 TCS。每个 TCS 指向一系列 SSA，并指定 RIP 的初始值以及 FS 和 GS 的基址。

Each SSA starts at the beginning of an EPC page, and uses up the number of EPC pages that is specified in the SSAFRAMESIZE field of the enclave's SECS. These alignment and size restrictions most likely simplify the SGX implementation by reducing the number of special cases that it needs to handle.

每个 SSA 从 EPC 页面的开头开始，并且耗尽了飞地 SECS 的 SSAFRAMESIZE 字段中指定 EPC 页面的数量。这些对齐和大小限制最有可能通过减少需要处理的特殊情况的数量来简化 SGX 实现。

An enclave thread's execution context consists of the general-purpose registers (GPRs) and the result of the XSAVE instruction (x 2.6). Therefore, the size of the execution context depends on the requested-feature bitmap (RFBM) used by to XSAVE. All the code in an enclave uses the same RFBM, which is declared in the XFRM enclave attribute (5.2.2). The number of EPC pages reserved for each SSA, specified in SSAFRAMESIZE, must7 be large enough to fit the XSAVE output for the feature bitmap specified by XFRM.

安全区线程的执行上下文由通用寄存器（GPR）和 XSAVE 指令（x 2.6）的结果组成。因此，执行上下文的大小取决于 XSAVE 使用的请求特征位图（RFBM）。安全区中的所有代码都使用相同的 RFBM，它在 XFRM 安全区属性（5.2.2）中声明。在 SSAFRAMESIZE 中指定的为每个 SSA 保留的 EPC 页面数量必须足够大，以适合 XFRM 指定的特征位图的 XSAVE 输出。

SSAs are stored in regular EPC pages, whose EPCM page type is PT REG. Therefore,

the SSA contents is accessible to enclave software. The SSA layout is architectural, and is completely documented in the SDM. This opens up possibilities for an enclave exception handler that is invoked by the host application after a hardware exception occurs, and acts upon the information in a SSA.

SSA 存储在常规 EPC 页面中，其 EPCM 页面类型为 PT REG。因此，安全区软件可以访问 SSA 内容。SSA 布局是体系结构，完全记录在 SDM 中。这为在发生硬件异常之后由主机应用程序调用的安全区异常处理程序开辟了可能性，并且对 SSA 中的信息起作用。

5.3 The Life Cycle of an SGX Enclave

An enclave's life cycle is deeply intertwined with resource management, specifically the allocation of EPC pages. Therefore, the instructions that transition between different life cycle states can only be executed by the system software. The system software is expected to expose the SGX instructions described below as enclave loading and teardown services. The following subsections describe the major steps in an enclave's lifecycle, which is illustrated by Figure 63.

Enclave 的生命周期与资源管理密切相关，尤其是 EPC 页面的分配。因此，不同生命周期状态之间转换的指令只能由系统软件执行。系统软件将揭示下文所述的 SGX 指令作为 enclave 加载和拆卸(loading and teardown)服务。以下小节介绍 enclave 生命周期中的主要步骤，如图 63 所示。

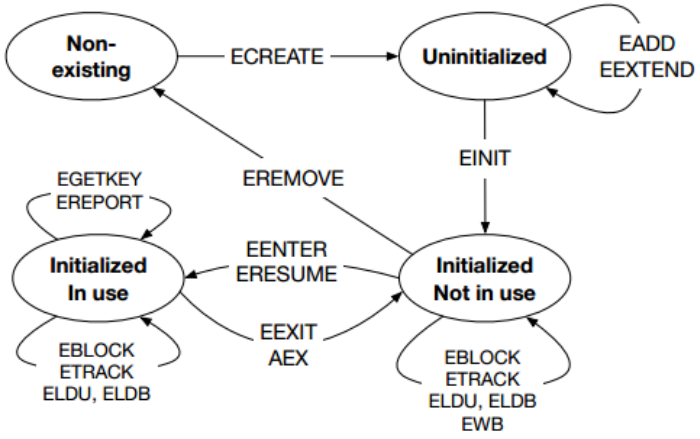


Figure 63: The SGX enclave life cycle management instructions and state transition diagram.

SGX 安全区生命周期管理指令和状态转换图

5.3.1 Creation

An enclave is born when the system software issues the ECREATE instruction, which turns a free EPC page into the SECS (5.1.3) for the new enclave.

当系统软件执行 ECREATE 指令时，enclave 就诞生了，ECREATE 指令将一个空 EPC 页面变成新 enclave 的 SECS（5.1.3）。

ECREATE initializes the newly created SECS using the information in a non-EPC page owned by the system software. This page specifies the values for all the SECS fields defined in the SDM, such as BASEADDR and SIZE, using an architectural layout that is guaranteed to be preserved by future implementations.

ECREATE 使用系统软件拥有的非 EPC 页面中的信息初始化新创建的 SECS。此页面使用保证未来实现保留的体系结构布局，为 SDM 中定义的所有 SECS 字段（例如

BASEADDR 和 SIZE）指定值。

While is very likely that the actual SECS layout used by initial SGX implementations matches the architectural layout quite closely, future implementations are free to deviate from this layout, as long as they maintain the ability to initialize the SECS using the architectural layout. Software cannot access an EPC page that holds a SECS, so it cannot become dependent on an internal SECS layout. This is a stronger version of the encapsulation used in the Virtual Machine Control Structure (VMCS, 2.8.3).

虽然初始 SGX 实现所使用的实际 SECS 布局很可能与体系结构布局相当匹配，但只要它们保持使用体系结构布局初始化 SECS 的能力，未来实现就可以自由地偏离该布局。软件无法访问持有 SECS 的 EPC 页面，因此它不能依赖于 SECS 内部布局。这是虚拟机控制结构（VMCS，2.8.3）中使用的封装的更强版本。

ECREATE validates the information used to initialize the SECS, and results in a page fault (#PF, 2.8.2) or general protection fault (#GP, 2.8.2) if the information is not valid. For example, if the SIZE field is not a power of two, ECREATE results in #GP. This validation, combined with the fact that the SECS is not accessible by software, simplifies the implementation of the other SGX instructions, which can assume that the information inside the SECS is valid.

ECREATE 验证用于初始化 SECS 的信息，如果信息无效，则会导致页面错误（#PF，2.8.2）或一般性保护错误（#GP，2.8.2）。例如，如果 SIZE 字段不是 2 的幂，ECREATE 将导致#GP。这种验证以及 SECS 无法通过软件访问的事实简化了其他 SGX 指令的实施，这些指令可以假定 SECS 内部的信息是有效的。

Last, ECREATE initializes the enclave's INIT attribute (sub-field of the ATTRIBUTES field in the enclave's SECS, 5.2.2) to the false value. The enclave's code cannot be executed until the INIT attribute is set to true, which happens in the initialization stage that will be described in 5.3.3.

最后，ECREATE 将 enclave 的 INIT 属性（enclave SECS 中的 ATTRIBUTES 字段的子字段，5.2.2）初始化为 false 值。在 INIT 属性设置为 true 之前，不能执行 enclave 的代码，这将在 5.3.3 中描述的初始化阶段发生。

5.3.2 Loading

ECREATE marks the newly created SECS as uninitialized. While an enclave's SECS is in this state, the system software can use EADD instructions to load the initial code and data into the enclave. EADD is used to create both TCS pages (5.2.4) and regular pages.

ECREATE 将新创建的 SECS 标记为未初始化。当 enclave 的 SECS 处于此状态时，系统软件可以使用 EADD 指令将初始代码和数据加载到 enclave 中。EADD 用于创建 TCS 页面（5.2.4）和常规页面。

EADD reads its input data from a Page Information (PAGEINFO) structure, illustrated in Figure 64. The structure's contents are only used to communicate information to the SGX implementation, so it is entirely architectural and documented in the SDM.

EADD 从页面信息（PAGEINFO）结构中读取输入数据，如图 64 所示。PAGEINFO 结构的内容仅用于向 SGX 实现传递信息，因此它是完全体系结构的（entirely architectural）并在 SDM 中记录。

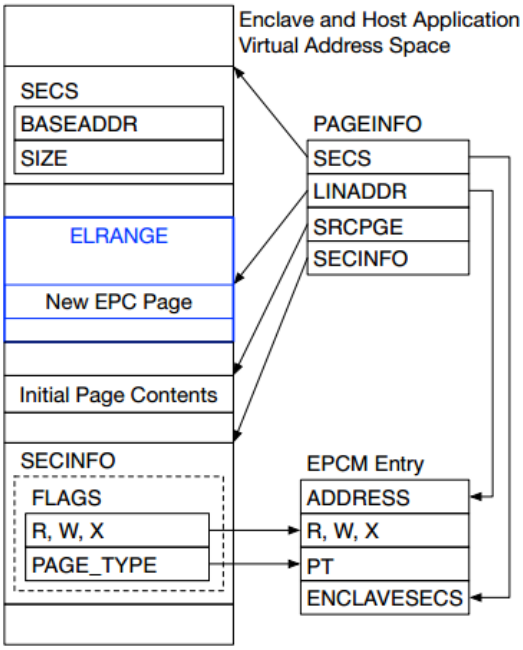


Figure 64: The PAGEINFO structure supplies input data to SGX instructions such as EADD.

Currently, the PAGEINFO structure contains the virtual address of the EPC page that will be allocated (LINADDR), the virtual address of the non-EPC page whose contents will be copied into the newly allocated EPC page (SRCPGE), a virtual address that resolves to the SECS of the enclave that will own the page (SECS), and values for some of the fields of the EPCM entry associated with the newly allocated EPC page (SECINFO).

目前，PAGEINFO 结构包含将分配的 EPC 页面的虚拟地址（LINADDR），其内容将被复制到新分配的 EPC 页面（SRCPGE）中的非 EPC 页面的虚拟地址，一个解析为拥有该页面（SECS）的 enclave 的 SECS 的虚拟地址，以及与新分配的 EPC 页面（SECINFO）相关联的 EPCM 条目的一些字段的值。

The SECINFO field in the PAGEINFO structure is actually a virtual memory address, and points to a Security Information (SECINFO) structure, some of which is also illustrated in Figure 64. The SECINFO structure contains the newly allocated EPC page's access permissions (R, W, X) and its EPCM page type (PT REG or PT TCS). Like PAGEINFO, the SECINFO structure is solely used to communicate data to the SGX implementation, so its contents are also entirely architectural. However, most of the structure's 64 bytes are reserved for future use.

PAGEINFO 结构中的 SECINFO 字段实际上是一个虚拟内存地址，并指向一个安全信息（SECINFO）结构，其中一些结构也在图 64 中说明。SECINFO 结构包含新分配的 EPC 页面的访问权限（R，W，X）及其 EPCM 页面类型（PT REG 或 PT TCS）。与 PAGEINFO 一样，SECINFO 结构仅用于将数据传递给 SGX 实现，因此其内容也是完全架构性的。但是，该结构的大部分 64 字节都保留供将来使用。

Both the PAGEINFO and the SECINFO structures are prepared by the system software that invokes the EADD instruction, and therefore must be contained in non-EPC pages. Both structures must be aligned to their sizes – PAGEINFO is 32 bytes long, so each PAGEINFO instance must be 32-byte aligned, while SECINFO has 64 bytes, and therefore each SECINFO instance must be 64-byte aligned. The alignment requirements likely simplify the SGX implementation by reducing the number of special cases that must be handled.

PAGEINFO 和 SECINFO 结构均由调用 EADD 指令的系统软件准备，因此必须包含在非 EPC 页面中。两个结构都必须对齐它们的大小--PAGEINFO 长度为 32 个字节，因此每个 PAGEINFO 实例必须是 32 字节对齐的，而 SECINFO 有 64 个字节，因此每个 SECINFO 实例必须是 64 字节对齐的。对齐要求可能通过减少必须处理的特殊情况的数量来简化 SGX 实施。

EADD validates its inputs before modifying the newly allocated EPC page or its EPCM entry. Most importantly, attempting to EADD a page to an enclave whose SECS is in the initialized state will result in a #GP. Furthermore, attempting to EADD an EPC page that is already allocated (the VALID field in its EPCM entry is 1) results in a #PF. EADD also ensures that the page's virtual address falls within the enclave's ELRANGE, and that all the reserved fields in SECINFO are set to zero.

EADD 在修改新分配的 EPC 页面或其 EPCM 条目之前验证其输入。最重要的是，尝试 EADD 页面到 SECS 处于初始化状态的 enclave 将导致#GP。此外，尝试 EADD 已分配的 EPC 页面（其 EPCM 条目中的 VALID 字段为 1）会导致#PF。EADD 还确保页面的虚拟地址落在 enclave 的 ELRANGE 内，并且 SECINFO 中的所有保留字段都设置为零。

While loading an enclave, the system software will also use the EEXTEND instruction, which updates the enclave's measurement used in the software attestation process. Software attestation is discussed in 5.8.

在加载 enclave 时，系统软件还将使用更新软件认证过程中使用的 enclave measurement 值 EEXTEND 指令。软件认证在 5.8 中讨论。

5.3.3 Initialization

After loading the initial code and data pages into the enclave, the system software must use a Launch Enclave (LE) to obtain an EINIT Token Structure, via an under-documented process that will be described in more detail in 5.9.1. The token is then provided to the EINIT instruction, which marks the enclave's SECS as initialized.

在将初始代码和数据页面加载到安全区后，系统软件必须使用启动区域（LE）获取 EINIT 令牌结构，通过 5.9.1 中详细描述的未记录过程。然后将令牌提供给 EINIT 指令，该指令将安全区的 SECS 标记为已初始化。

The LE is a privileged enclave provided by Intel, and is a prerequisite for the use of enclaves authored by parties other than Intel. The LE is an SGX enclave, so it must be created, loaded and initialized using the processes described in this section. However, the LE is cryptographically signed (3.1.3) with a special Intel key that is hard-coded into the SGX implementation, and that causes EINIT to initialize the LE without checking for a valid EINIT Token Structure.

LE 是英特尔提供的特权安全区，也是使用英特尔之外其他各方创作的安全区的先决条件。 LE 是 SGX 安全区，因此必须使用本节中描述的流程创建，加载和初始化它。然而，LE 使用特殊的英特尔密钥进行加密签名（3.1.3），该特殊的英特尔密钥被硬编码到 SGX 实现中，并导致 EINIT 初始化 LE 而不检查有效的 EINIT 令牌结构。

When EINIT completes successfully, it sets the enclave's INIT attribute to true. This opens the way for ring 3 (2.3) application software to execute the enclave's code, using the SGX instructions described in 5.4. On the other hand, once INIT is set to true, EADD cannot be invoked on that enclave anymore, so the system software must load all the pages that make up the enclave's initial state before executing the EINIT instruction.

当 EINIT 成功完成时，它将安全区的 INIT 属性设置为 true。 这为环 3（2.3）应用软件使用 5.4 中描述的 SGX 指令执行安全区的代码打开了方法。 另一方面，一旦 INIT 设置为真，EADD 就不能在该安全区上被调用，因此系统软件必须在执行 EINIT 指令之前加载构成安全区初始状态的所有页面。

5.3.4 Teardown

After the enclave has done the computation it was designed to perform, the system software executes the EREMOVE instruction to de-allocate the EPC pages used by the enclave.

在 enclave 完成设计执行的计算后，系统软件执行 EREMOVE 指令以取消分配给该 enclave 使用的 EPC 页面。

EREMOVE marks an EPC page as available by setting the VALID field of the page's EPCM entry to 0 (zero). Before freeing up the page, EREMOVE makes sure that there is no logical processor executing code inside the enclave that owns the page to be removed.

通过将页面的 EPCM 条目的 VALID 字段设置为 0（零），EREMOVE 将 EPC 页面标记为可用。在释放页面之前，EREMOVE 确保没有逻辑处理器在拥有要删除页面的 enclave 内执行代码。

An enclave is completely destroyed when the EPC page holding its SECS is freed. EREMOVE refuses to de-allocate a SECS page if it is referenced by any other EPCM entry's ENCLAVESECS field, so an enclave's SECS page can only be de-allocated after all the enclave's pages have been de-allocated.

当持有其 SECS 的 EPC 页面被释放时，enclave 被完全销毁。 如果 EREMOVE 被任何其他 EPCM 条目的 ENCLAVESECS 字段引用，则 EREMOVE 拒绝销毁 SECS 页面，因此，只有在所有 enclave 的页面被 de-allocated 后，enclave 的 SECS 页面才能被 de-allocated。

5.4 The Life Cycle of an SGX Thread

Between the time when an enclave is initialized (5.3.3) and the time when it is torn down (5.3.4), the enclave's code can be executed by any application process that has the enclave's EPC pages mapped into its virtual address space.

在 enclave 初始化（5.3.3 节）和销毁时间（5.3.4 节）之间，enclave 的代码可以由任何具有 enclave 的 EPC 页面映射到其虚拟地址空间的应用程序执行。

When executing the code inside an enclave, a logical processor is said to be in enclave mode, and the code that it executes can access the regular (PT REG, 5.1.2) EPC pages that belong to the currently executing enclave. When a logical process is outside enclave mode, it bounces any memory accesses inside the Processor Reserved Memory range (PRM, 5.1), which includes the EPC.

当在 enclave 内执行代码时，逻辑处理器被认为处于 enclave 模式，并且它执行的代码可以访问属于当前正在执行 enclave 的常规（PT REG，5.1.2）EPC 页面。当逻辑进程处于 enclave 模式之外时，它会 bounces 包含 EPC 页面的 PRM range（PRM，5.1 节）内的任何内存访问。

Each logical processor that executes enclave code uses a Thread Control Structure (TCS, 5.2.4). When a TCS is used by a logical processor, it is said to be busy, and it cannot be used by any other logical processor. Figure 65 illustrates the instructions used by a host process to execute enclave code and their interactions with the TCS that they target.

每个执行 enclave 代码的逻辑处理器都使用线程控制结构（TCS，5.2.4 节）。当逻辑处理器使用 TCS 时，视作 busy，并且它不能被任何其他逻辑处理器使用。图 65 说明主

进程用于执行 enclave 代码及其与它们与目标 TCS 的交互的指令。

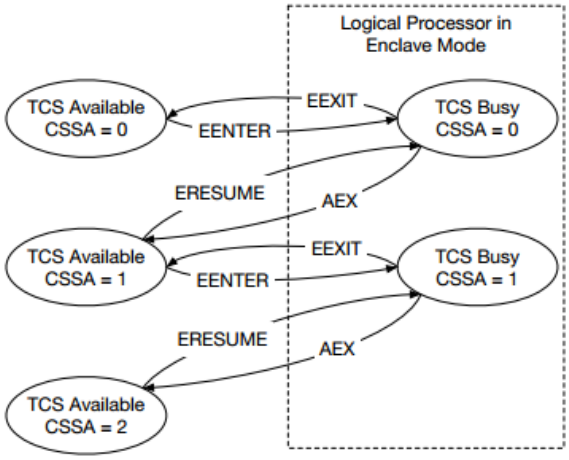


Figure 65: The stages of the life cycle of an SGX Thread Control Structure (TCS) that has two State Save Areas (SSAs).

Figure 65:具有两个状态保存区（SSA）的 SGX 线程控制结构（TCS）的生命周期阶段。

Assuming that no hardware exception occurs, an enclave's host process uses the EENTER instruction, described in 5.4.1, to execute enclave code. When the enclave code finishes performing its task, it uses the EEXIT instruction, covered in 5.4.2, to return the execution control to the host process that invoked the enclave.

假设没有硬件异常发生，enclave 的主进程使用 5.4.1 中描述的 EENTER 指令来执行 enclave 代码。当 enclave 代码完成其任务时，它使用 5.4.2 中介绍的 EEXIT 指令将执行控制返回给调用该 enclave 的主进程。

If a hardware exception occurs while a logical processor is in enclave mode, the processor is taken out of enclave mode using an Asynchronous Enclave Exit (AEX), summarized in 5.4.3, before the system software's exception handler is invoked. After the system software's handler is invoked, the enclave's host process can use the ERESUME instruction, described in 5.4.4, to reenter the enclave and resume the computation that it was performing.

如果在逻辑处理器处于 enclave 模式时发生硬件异常，则在调用系统软件的异常处理程序之前，处理器将使用 5.4.3 节概述的 Asynchronous Enclave Exit（AEX）离开 enclave 模式。系统软件的处理程序被调用后，enclave 的主进程可以使用 5.4.4 节中描述的 ERESUME 指令重新进入 enclave 并恢复其执行的计算。

5.4.1 Synchronous Enclave Entry

At a high level, EENTER performs a controlled jump into enclave code, while performing the processor configuration that is needed by SGX's security guarantees. Going through all the configuration steps is a tedious exercise, but it a necessary prerequisite to understanding how all data structures used by SGX work together. For this reason, EENTER and its siblings are described in much more detail than the other SGX instructions.

At a high level，EENTER 执行受控跳转到 enclave 代码，同时执行 SGX 安全保证所需的处理器配置。完成所有配置步骤是一项单调乏味的工作，但它是理解 SGX 使用的所有数据结构如何协同工作的必要先决条件。出于这个原因，EENTER 及其 siblings 的描述比其他 SGX 指令更详细。

EENTER, illustrated in Figure 66 can only be executed by unprivileged application software running at ring 3 (2.3), and results in an undefined instruction (#UD) fault if is

executed by system software.

图 66 所示的 EENTER 只能由在 ring 3（2.3）运行的非特权应用软件执行，并且如果由系统软件执行则会导致未定义指令（#UD）错误。

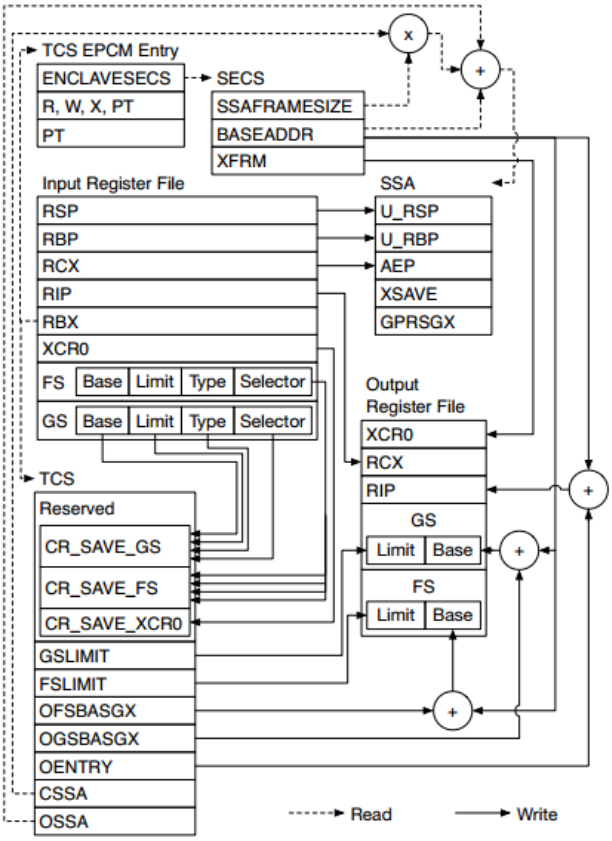


Figure 66: Data flow diagram for a subset of the logic in EENTER. The figure omits the logic for disabling debugging features, such as hardware breakpoints and performance monitoring events.

图 66：ENTER 中逻辑子集的数据流图。该图省略了用于禁用调试功能的逻辑，例如硬件断点和性能监视事件。

EENTER switches the logical processor to enclave mode, but does not perform a privilege level switch (2.8.2). Therefore, enclave code always executes at ring 3, with the same privileges as the application code that calls it. This makes it possible for an infrastructure owner to allow user-supplied software to create and use enclaves, while having the assurance that the OS kernel and hypervisor can still protect the infrastructure from buggy or malicious software.

EENTER 将逻辑处理器切换到 enclave 模式，但不执行特权级别切换（2.8.2）。因此，enclave 代码总是在 ring 3 执行，具有与调用它的应用程序代码相同的特权。这使基础设施所有者可以允许用户提供的软件创建和使用 enclave，同时确保 OS 内核和管理程序仍然可以保护基础设施免受漏洞或恶意软件的侵害。

EENTER takes the virtual address of a TCS as its input, and requires that the TCS is available (not busy), and that at least one State Save Area (SSA, 5.2.5) is available in the TCS. The latter check is implemented by making sure that the current SSA index (CSSA) field in the TCS is less than the number of SSAs (NSSA) field. The SSA indicated by the CSSA, which shall be called the current SSA, is used in the event that a hardware exception occurs while enclave code is executed.

EENTER 将 TCS 的虚拟地址作为输入，并要求 TCS 可用（not busy），并且 TCS 中

至少有一个状态保存区域（SSA，5.2.5）可用。接下来检查通过确保 TCS 中的当前 SSA 索引（CSSA）字段小于 SSA（NSSA）字段的数量来实现。如果在执行 enclave 代码时发生硬件异常，CSSA 调用的 SSA 将被称为当前 SSA。

EENTER transitions the logical processor into enclave mode, and sets the instruction pointer (RIP) to the value indicated by the entry point offset (OENTRY) field in the TCS that it receives. EENTER is used by an untrusted caller to execute code in a protected environment, and therefore has the same security considerations as SYSCALL (2.8), which is used to call into system software. Setting RIP to the value indicated by OENTRY guarantees to the enclave author that the enclave code will only be invoked at well defined points, and prevents a malicious host application from bypassing any security checks that the enclave author may perform.

EENTER 将逻辑处理器转换为 enclave 模式，并将指令指针（RIP）设置为由其接收的 TCS 中的入口点偏移（OENTRY）字段指示的值。EENTER 由不受信任的调用者用于在受保护的环境中执行代码，因此具有与 SYSCALL（2.8）相同的安全性考虑因素，SYSCALL 用于调用系统软件。将 RIP 设置为 OENTRY 指示的值可以向 enclave author 保证仅在明确定义的点处调用飞地代码，并防止恶意主机应用程序绕过 enclave author 可能执行的任何安全检查。

EENTER also sets XCR0 (2.6), the register that controls which extended architectural features are in use, to the value of the XFRM enclave attribute (5.2.2). Ensuring that XCR0 is set according to the enclave author's intentions prevents a malicious operating system from bypassing an enclave's security by enabling architectural features that the enclave is not prepared to handle.

EENTER 还将 XCR0（2.6 节）（控制使用哪些扩展架构特征的寄存器）设置为 XFRM 飞地属性（5.2.2 节）的值。确保根据飞地作者的意图设置 XCR0 可防止恶意操作系统通过启用飞地不准备处理的架构特征来绕过飞地的安全性。

Furthermore, EENTER loads the bases of the segment registers（2.7) FS and GS using values specified in the TCS. The segments' selectors and types are hard-coded to safe values for ring 3 data segments. This aspect of the SGX design makes it easy to implement per-thread Thread Local Storage (TLS). For 64-bit enclaves, this is a convenience feature rather than a security measure, as enclave code can securely load new bases into FS and GS using the WRFSBASE and WRGSBASE instructions.

此外，EENTER 使用 TCS 中指定的值加载段寄存器（2.7 节）FS 和 GS 的基数。段的选择器和类型被硬编码为 ring 3 数据段的安全值。SGX 这一设计使得容易实现单线程线程本地存储（TLS）。对于 64 位飞地，这是一个便利功能而不是安全措施，因为飞地代码可以使用 WRFSBASE 和 WRGSBASE 指令将新基础安全地加载到 FS 和 GS 中。

The EENTER implementation backs up the old values of the registers that it modifies, so they can be restored when the enclave finishes its computation. Just like SYSCALL, EEENTER saves the address of the following instruction in the RCX register.

EENTER 实现备份它修改的寄存器的旧值，因此当安全区完成计算时可以恢复它们。与 SYSCALL 一样，EEENTER 将以下指令的地址保存在 RCX 寄存器中。

Interestingly, the SDM states that the old values of the XCR0, FS, and GS registers are saved in new registers dedicated to the SGX implementation. However, given that they will only be used on an enclave exit, we expect that the registers are saved in DRAM, in the reserved area in the TCS.

有趣的是，SDM 声明 XCR0，FS 和 GS 寄存器的旧值保存在 SGX 实现的专用新寄存器中。但是，鉴于它们仅用于 enclave exit，我们希望寄存器保存在 DRAM 中，TCS 的保留区域（reserved area）中。

Like SYSCALL, EENTER does not modify the stack pointer register (RSP). To avoid any security exploits, enclave code should set RSP to point to a stack area that is entirely contained in EPC pages. Multi-threaded enclaves can easily implement per-thread stack areas by setting up each thread's TLS area to include a pointer to the thread's stack, and by setting RSP to the value obtained by reading the TLS area at which the FS or GS segment points.

与 SYSCALL 一样，EENTER 不会修改堆栈指针寄存器（RSP）。为避免任何安全漏洞，飞地代码应将 RSP 设置为指向完全包含在 EPC 页面中的堆栈区域。通过设置每个线程的 TLS 区域以包括指向线程堆栈的指针，并通过将 RSP 设置为通过读取 FS 或 GS 段指向的 TLS 区域获得的值，多线程飞地可以轻松实现单线程堆栈区域。

Last, when EENTER enters enclave mode, it suspends some of the processor's debugging features, such as hardware breakpoints and Precise Event Based Sampling (PEBS). Conceptually, a debugger attached to the host process sees the enclave's execution as one single processor instruction.

最后，当 EENTER 进入安全区模式时，它会暂停某些处理器的调试功能，例如硬件断点和基于精确事件的采样（PEBS）。从概念上讲，附加到主机进程的调试器将飞地执行视为一个单处理器指令。

5.4.2 Synchronous Enclave Exit

EEXIT can only be executed while the logical processor is in enclave mode, and results in a (#UD) if executed in any other circumstances. In a nutshell, the instruction returns the processor to ring 3 outside enclave mode and restores the registers saved by EENTER, which were described above.

EEXIT 只能在逻辑处理器处于飞地模式时执行，如果在任何模式下执行，则会产生（#UD）错误。简而言之，该指令将处理器返回到飞地模式外的 ring 3 并恢复由 EENTER 保存的寄存器，如上所述。

Unlike SYSRET, EEXIT sets RIP to the value read from RBX, after exiting enclave mode. This is inconsistent with EENTER, which saves the RIP value to RCX. Unless this inconsistency stems from an error in the SDM, enclave code must be sure to note the difference.

与 SYSRET 不同，EEXIT 在退出安全区模式后将 RIP 设置为从 RBX 读取的值。这与 EENTER 不一致，后者将 RIP 值保存到 RCX。除非这种不一致源于 SDM 中的错误，否则飞地代码必须注意区别。

The SDM explicitly states that EEXIT does not modify most registers, so enclave authors must make sure to clear any secrets stored in the processor's registers before returning control to the host process. Furthermore, enclave software will most likely cause a fault in its caller if it doesn't restore the stack pointer RSP and the stack frame base pointer RBP to the values that they had when EENTER was called.

SDM 明确指出 EEXIT 不会修改大多数寄存器，因此 enclave author 必须确保在将控制权返回给主机进程之前清除存储在处理器寄存器中的任何秘密。此外，如果没有将堆栈指针 RSP 和堆栈帧基指针 RBP 恢复到调用 EENTER 时的值，则飞地软件很可能会导致其调用者出错。

It may seem unfortunate that enclave code can induce faults in its caller. For better or for worse, this perfectly matches the case where an application calls into a dynamically loaded

module. More specifically, the module's code is also responsible for preserving stack-related registers, and a buggy module might jump anywhere in the application code of the host process.

可能看起来很不幸的是，飞地代码会在其调用者中引发错误。无论好坏，这完全匹配应用程序调用动态加载模块的情况。更具体地说，模块的代码还负责保留与堆栈相关的寄存器，并且有缺陷的模块可能会跳转到主机进程的应用程序代码中的任何位置。

This section describes the EENTER behavior for 64-bit enclaves. The EENTER implementation for 32-bit enclaves is significantly more complex, due to the extra special cases introduced by the full-fledged segmentation model that is still present in the 32-bit Intel architecture. As stated in the introduction, we are not interested in such legacy aspects.

本节介绍 64 位飞地的 EENTER 行为。由于 32 位英特尔架构中仍然存在的完整分段模型引入了额外的特殊情况，因此 32 位安全区的 EENTER 实现要复杂得多。 、如引言中所述，我们对此类遗留方面不感兴趣。

5.4.3 Asynchronous Enclave Exit (AEX)

If a hardware exception, like a fault (2.8.2) or an interrupt (2.12), occurs while a logical processor is executing an enclave's code, the processor performs an Asynchronous Enclave Exit (AEX) before invoking the system software's exception handler, as shown in Figure 67.

如果在逻辑处理器执行安全区代码时发生硬件异常，如故障（2.8.2）或中断（2.12），则处理器在调用系统软件的异常处理程序之前执行异步磁盘退出（AEX），如 如图 67 所示。

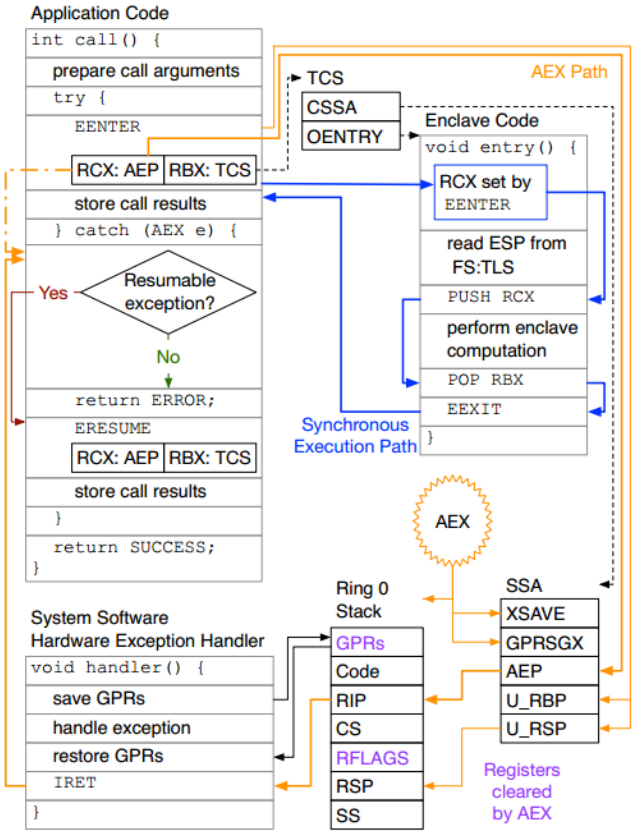


Figure 67: If a hardware exception occurs during enclave execution, the synchronous execution path is aborted, and an Asynchronous Enclave Exit (AEX) occurs instead.

图 67：如果在飞地执行期间发生硬件异常，则中止同步执行路径，并产生生异步飞地 Exit（AEX）。

The AEX saves the enclave code's execution context (2.6), restores the state saved by EENTER, and sets up the processor registers so that the system software's hardware exception handler will return to an asynchronous exit handler in the enclave's host process. The exit handler is expected to use the ERESUME instruction to resume the enclave computation that was interrupted by the hardware exception.

AEX 保存了飞地代码的执行上下文（2.6），恢复了 EENTER 保存的状态，并设置了处理器寄存器，以便系统软件的硬件异常处理程序返回到飞地主机进程中的异步退出处理程序。退出处理程序应使用 ERESUME 指令来恢复由硬件异常中断的飞地计算。

Asides from the behavior described in 5.4.1, EENTER also writes some information to the current SSA, which is only used if an AEX occurs. As shown in Figure 66, EENTER stores the stack pointer register RSP and the stack frame base pointer register RBP into the U RSP and U RBP fields in the current SSA. Last, EENTER stores the value in RCX in the Asynchronous Exit handler Pointer (AEP) field in the current SSA.

除了 5.4.1 中描述的行为之外，EENTER 还将一些信息写入当前 SSA，仅在发生 AEX 时使用。 如图 66 所示，EENTER 将堆栈指针寄存器 RSP 和堆栈帧基址指针寄存器 RBP 存储到当前 SSA 中的 U RSP 和 U RBP 字段中。最后，EENTER 将 RCX 中的值存储在当前 SSA 的异步退出处理程序指针（AEP）字段中。

When a hardware exception occurs in enclave mode, the SGX implementation performs a sequence of steps that takes the logical processor out of enclave mode and invokes the hardware exception handler in the system software. Conceptually, the SGX implementation first performs an AEX to take the logical processor out of enclave mode, and then the hardware exception is handled using the standard Intel architecture's behavior described in 2.8.2. Actual Intel processors may interleave the AEX implementation with the exception handling implementation. However, for simplicity, this work describes AEX as a separate process that is performed before any exception handling steps are taken.

当在飞地模式下发生硬件异常时，SGX 实现执行一系列步骤，使逻辑处理器退出安全区模式并调用系统软件中的硬件异常处理程序。从概念上讲，SGX 实现首先执行 AEX 以使逻辑处理器退出飞地模式，然后使用 2.8.2 中描述的标准 Intel 架构的行为来处理硬件异常。实际的英特尔处理器可以将 AEX 实现与异常处理实现交错。但是，为简单起见，此工作将 AEX 描述为在执行任何异常处理步骤之前执行的单独过程。

In the Intel architecture, if a hardware exception occurs, the application code's execution context can be read and modified by the system software's exception handler (2.8.2). This is acceptable when the system software is trusted by the application software. However, under SGX's threat model, the system software is not trusted by enclaves. Therefore, the AEX step erases any secrets that may exist in the execution state by resetting all its registers to predefined values.

在 Intel 架构中，如果发生硬件异常，可以通过系统软件的异常处理程序（2.8.2）读取和修改应用程序代码的执行上下文。当系统软件受应用程序软件信任时，这是可以接受的。但是，根据 SGX 的威胁模型，飞地不信任系统软件。因此，AEX 步骤通过将其所有寄存器重置为预定义值来擦除执行状态中可能存在的任何秘密。

Before the enclave's execution state is reset, it is backed up inside the current SSA. Specifically, an AEX backs up the general purpose registers (GPRs, 2.6) in the GPRSGX area in the SSA, and then performs an XSAVE (2.6) using the requested-feature bitmap (RFBM) specified in the XFRM field in the enclave's SECS. As each SSA is entirely stored in EPC

pages allocated to the enclave, the system software cannot read or tamper with the backed up execution state. When an SSA receives the enclave's execution state, it is marked as used by incrementing the CSSA field in the current TCS.

在重置飞地执行状态之前，它将在当前 SSA 内备份。具体来说，AEX 备份 SSA 中 GPRSGX 区域中的通用寄存器（GPRs，2.6 节），然后使用安全区 SECS 中 XFRM 字段中指定的请求特征位图（RFBM）执行 XSAVE（2.6 节）。由于每个 SSA 完全存储在分配给安全区的 EPC 页面中，因此系统软件无法读取或篡改备份的执行状态。当 SSA 收到飞地的执行状态时，通过递增当前 TCS 中的 CSSA 字段将其标记为已使用。

After clearing the execution context, the AEX process sets RSP and RBP to the values saved by EENTER in the current SSA, and sets RIP to the value in the current SSA's AEP field. This way, when the system software's hardware exception handler completes, the processor will execute the asynchronous exit handler code in the enclave's host process. The SGX design makes it easy to set up the asynchronous handler code as an exception handler in the routine that contains the EENTER instruction, because the RSP and RBP registers will have the same values as they had when EENTER was executed.

清除执行上下文后，AEX 进程将 RSP 和 RBP 设置为 EENTER 在当前 SSA 中保存的值，并将 RIP 设置为当前 SSA 的 AEP 字段中的值。这样，当系统软件的硬件异常处理程序完成时，处理器将在安全区的主机进程中执行异步退出处理程序代码。SGX 设计可以很容易地将异步处理程序代码设置为包含 EENTER 指令的例程中的异常处理程序，因为 RSP 和 RBP 寄存器将具有与执行 EENTER 时相同的值。

Many of the actions taken by AEX to get the logical processor outside of enclave mode match EEXIT. The segment registers FS and GS are restored to the values saved by EENTER, and all the debugging facilities that were suppressed by EENTER are restored to their previous states.

AEX 为使逻辑处理器在飞地模式之外采取的许多行动与 EEXIT 相匹配。段寄存器 FS 和 GS 将恢复为 EENTER 保存的值，并且 EENTER 抑制（suppressed）的所有调试工具都将恢复到之前的状态。

5.4.4 Recovering from an Asynchronous Exit

When a hardware exception occurs inside enclave mode, the processor performs an AEX before invoking the exception's handler set up by the system software. The AEX sets up the execution context in such a way that when the system software finishes processing the exception, it returns into an asynchronous exit handler in the enclave's host process. The asynchronous exception handler usually executes the ERESUME instruction, which causes the logical processor to go back into enclave mode and continue the computation that was interrupted by the hardware exception.

当在安全区模式内发生硬件异常时，处理器在调用系统软件设置的异常处理程序之前执行 AEX。AEX 以这样的方式设置执行上下文：当系统软件完成处理异常时，它返回到飞地主机进程中的异步退出处理程序。异步异常处理程序通常执行 ERESUME 指令，该指令使逻辑处理器返回到飞地模式并继续由硬件异常中断的计算。

ERESUME shares much of its functionality with EENTER. This is best illustrated by the similarity between Figures 68 and 67.

RESUME 与 REENTER 共享其大部分功能。图 68 和 67 之间的相似性最好地说明了这一点。

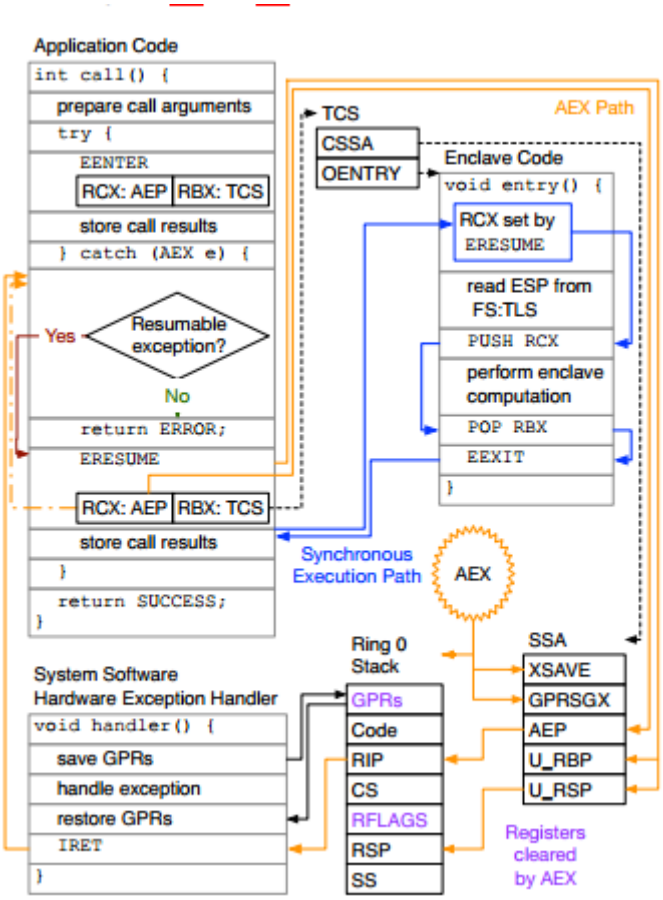


Figure 68: If a hardware exception occurs during enclave execution, the synchronous execution path is aborted, and an Asynchronous Enclave Exit (AEX) occurs instead.

图 68：如果在安全区执行期间发生硬件异常，则中止同步执行路径，同时产生异步飞地 Exit（AEX）。

EENTER and ERESUME receive the same inputs, namely a pointer to a TCS, described in 5.4.1, and an AEP, described in 5.4.3. The most common application design will pair each EENTER instance with an asynchronous exit handler that invokes ERESUME with exactly the same arguments.

EENTER 和 ERESUME 接收相同的输入，即 5.4.1 中描述的指向 TCS 的指针，以及 5.4.3 中描述的 AEP。最常见的应用程序设计将 pair EENTER 实例与异步退出处理程序，该处理程序使用完全相同的参数调用 ERESUME。

The main difference between ERESUME and EENTER is that the former uses an SSA that was “filled out” by an AEX (5.4.3), whereas the latter uses an empty SSA. Therefore, ERESUME results in a #GP fault if the CSSA field in the provided TCS is 0 (zero), whereas EENTER fails if CSSA is greater than or equal to NSSA.

ERESUME 和 EENTER 之间的主要区别在于前者使用的是由 AEX（5.4.3）“填充”的 SSA，而后者使用空的 SSA。因此，如果提供的 TCS 中的 CSSA 字段为 0（零），则 ERESUME 会导致#GP 错误，而如果 CSSA 大于或等于 NSSA，则 EENTER 会失败。

When successful, ERESUME decrements the CSSA field of the TCS, and restores the execution context backed up in the SSA pointed to by the CSSA field in the TCS. Specifically, the ERESUME implementation restores the GPRs (2.6) from the GPRSGX field in the SSA,

and performs an XRSTOR (2.6) to load the execution state associated with the extended architectural features used by the enclave.

成功后，ERESUME 会递减 TCS 的 CSSA 字段，并恢复在 TCS 中 CSSA 字段指向的 SSA 中备份的执行上下文。具体地，ERESUME 实现从 SSA 中的 GPRSGX 字段恢复 GPR（2.6），并执行 XRSTOR（2.6）以加载与飞地使用的扩展架构特征相关联的执行状态。

ERESUME shares the following behavior with EENTER (5.4.1). Both instructions write the U RSP, U RBP, and AEP fields in the current SSA. Both instructions follow the same process for backing up XCR0 and the FS and GS segment registers, and set them to the same values, based on the current TCS and its enclave's SECS. Last, both instructions disable the same subset of the logical processor's debugging features.

ERESUME 与 EENTER（5.4.1）共享以下行为。 两条指令都在当前 SSA 中写入 U RSP，U RBP 和 AEP 字段。两条指令都遵循相同的过程来备份 XCR0 和 FS 和 GS 段寄存器，并根据当前的 TCS 及其安全区的 SECS 将它们设置为相同的值。最后，两条指令都禁用了逻辑处理器调试功能的相同子集。

An interesting edge case that ERESUME handles correctly is that it sets XCR0 to the XFRM enclave attribute before performing an XRSTOR. It follows that ERESUME fails if the requested feature bitmap (RFBM) in the SSA is not a subset of XFRM. This matters because, while an AEX will always use the XFRM value as the RFBM, enclave code executing on another thread is free to modify the SSA contents before ERESUME is called.

ERESUME 正确处理的一个有趣的边缘情况是它在执行 XRSTOR 之前将 XCR0 设置为 XFRM 飞地属性。如果 SSA 中请求的特征位图（RFBM）不是 XFRM 的子集，则 ERESUME 失败。这很重要，因为虽然 AEX 总是使用 XFRM 值作为 RFBM，但在另一个线程上执行的飞地代码可以在调用 ERESUME 之前自由修改 SSA 内容。

The correct sequencing of actions in the ERESUME implementation prevents a malicious application from using an enclave to modify registers associated with extended architectural features that are not declared in XFRM. This would break the system software's ability to provide thread-level execution context isolation.

ERESUME 实现中的操作的正确排序可防止恶意应用程序使用飞地修改与未在 XFRM 中声明的扩展体系结构功能相关联的寄存器。这将破坏系统软件提供线程级执行上下文隔离的能力。

5.5 EPC Page Eviction

Modern OS kernels take advantage of address translation (x 2.5) to implement page swapping, also referred to as paging (x 2.5). In a nutshell, paging allows the OS kernel to over-commit the computer's DRAM by evicting rarely used memory pages to a slower storage medium called the disk.

Paging is a key contributor to utilizing a computer's resources effectively. For example, a desktop system hose user runs multiple programs concurrently can evict memory pages allocated to inactive applications without a significant degradation in user experience.

Unfortunately, the OS cannot be allowed to evict an enclave's EPC pages via the same methods that are used to implement page swapping for DRAM memory outside the PRM range. In the SGX threat model, enclaves do not trust the system software, so the SGX design offers an EPC page eviction method that can defend against a malicious OS that attempts any of the

active address translation attacks described in x 3.7.

The price of the security afforded by SGX is that an S kernel that supports evicting EPC pages must use modified page swapping implementation that interacts with the SGX mechanisms. Enclave authors can mostly ignore EPC evictions, similarly to how today's application developers can ignore the OS kernel's paging implementation.

As illustrated in Figure 69, SGX supports evicting EPC pages to DRAM pages outside the PRM range. The system software is expected to use its existing page swapping implementation to evict the contents of these pages out of DRAM and onto a disk.

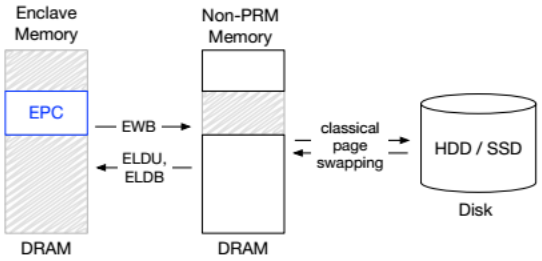


Figure 69: SGX offers a method for the OS to evict EPC pages into non-PRM DRAM. The OS can then use its standard paging feature to evict the pages out of DRAM.

SGX's eviction feature revolves around the EWB instruction, described in detail in x 5.5.4. Essentially, EWB evicts an EPC page into a DRAM page outside the EPC and marks the EPC page as available, by zeroing the VALID field in the page's EPCM entry. The SGX design relies on symmetric key cryptograpy 3.1.1 to guarantee the confidentiality and integrity of the evicted EPC pages, and on nonces (x 3.1.4) to guarantee the freshness of the pages brought back into the EPC. These nonces are stored in Version Arrays (VAs), covered in x 5.5.2, which are EPC pages dedicated to nonce storage.

Before an EPC page is evicted and freed up for use by other enclaves, the SGX implementation must ensure that no TLB has address translations associated with the evicted page, in order to avoid the TLB-based address translation attack described in x 3.7.4.

As explained in x 5.1.1, SGX leaves the system software in charge of managing the EPC. It naturally follows that the SGX instructions described in this section, which are used to implement EPC paging, are only available to system software, which runs at ring 0 x 2.3.

In today's software stacks (x 2.3), only the OS kernel implements page swapping in order to support the over-committing of DRAM. The hypervisor is only used to partition the computer's physical resources between operating systems. Therefore, this section is written with the expectation that the OS kernel will also take on the responsibility of EPC page swapping. For simplicity, we often use the term "OS kernel" instead of "system software". The reader should be aware that the SGX design does not preclude a system where the hypervisor implements its own EPC page swapping. Therefore, "OS kernel" should really be read as "the system software that performs EPC paging".

5.5.1 Page Eviction and the TLBs

One of the least promoted accomplishments of SGX is that it does not add any security checks to the memory execution units (x 2.9.4, x 2.10). Instead, SGX's access control checks occur after an address translation (x 2.5) is performed, right before the translation result is

written into the TLBs (x 2.11.5). This aspect is generally downplayed throughout the SDM, but it becomes visible when explaining SGX's EPC page eviction mechanism.

A full discussion of SGX's memory access protections checks merits its own section, and is deferred to x 6.2. The EPC page eviction mechanisms can be explained using only two requirements from SGX's security model. First, when a logical processor exits an enclave, either via EEXIT (x 5.4.2) or via an AEX (x 5.4.3), its TLBs are flushed. Second, when an EPC page is deallocated from an enclave, all logical processors executing that enclave's code must be directed to exit the enclave. This is sufficient to guarantee the removal of any TLB entry targeting the deallocated EPC.

System software can cause a logical processor to exit an enclave by sending it an Inter-Processor Interrupt (IPI, x 2.12), which will trigger an AEX when received. Essentially, this is a very coarse-grained TLB shootdown.

SGX does not trust system software. Therefore, before marking an EPC page's EPCM entry as free, the SGX implementation must ensure that the OS kernel has flushed all the TLBs that might contain translations for the page. Furthermore, performing IPIs and TLB flushes for each page eviction would add a significant overhead to a paging implementation, so the SGX design allows a batch of pages to be evicted using a single IPI / TLB flush sequence.

The TLB flush verification logic relies on a 1-bit EPCM entry field called BLOCKED. As shown in Figure 70, the VALID and BLOCKED fields yield three possible EPC page states. A page is free when both bits are zero, in use when VALID is one and BLOCKED is zero, and blocked when both bits are one.

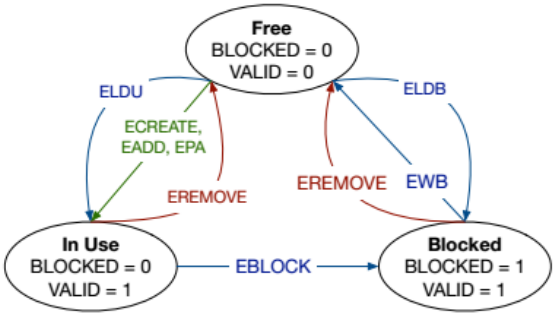


Figure 70: The VALID and BLOCKED bits in an EPC page's EPCM entry can be in one of three states. EADD and its siblings allocate new EPC pages. EREMOVE permanently deallocates an EPC page. EBLOCK blocks an EPC page so it can be evicted using EWB. ELDB and ELDU load an evicted page back into the EPC.

Blocked pages are not considered accessible to enclaves. If an address translation results in a blocked EPC page, the SGX implementation causes the translation to result in a Page Fault (#PF, x 2.8.2). This guarantees that once a page is blocked, the CPU will not create any new TLB entries pointing to it.

Furthermore, every SGX instruction makes sure that the EPC pages on which it operates are not blocked. For example, EENTER ensures that the TCS it is given is not blocked, that its enclave's SECS is not blocked, and that every page in the current SSA is not blocked.

In order to evict a batch of EPC pages, the OS kernel must first issue EBLOCK instructions targeting them. The OS is also expected to remove the EPC page's mapping from page tables, but is not trusted to do so.

After all the desired pages have been blocked, the OS kernel must execute an ETRACK instruction, which directs the SGX implementation to keep track of which logical processors have had their TLBs flushed. ETRACK requires the virtual address of an enclave's SECS (x 5.1.3). If the OS wishes to evict a batch of EPC pages belonging to multiple enclaves, it must issue an ETRACK for each enclave.

Following the ETRACK instructions, the OS kernel must induce enclave exits on all the logical processors that are executing code inside the enclaves that have been ETRACKed. The SGX design expects that the OS will use IPIs to cause AEXs in the logical processors whose TLBs must be flushed.

The EPC page eviction process is completed when the OS executes an EWB instruction for each EPC page to be evicted. This instruction, which will be fully described in x 5.5.4, writes an encrypted version of the EPC page to be evicted into DRAM, and then frees the page by clearing the VALID and BLOCKED bits in its EPCM entry. Before carrying out its tasks, EWB ensures that the EPC page that it targets has been blocked, and checks the state set up by ETRACK to make sure that all the relevant TLBs have been flushed.

An evicted page can be loaded back into the EPC via the ELDU and ELDB instructions. Both instructions start up with a free EPC page and a DRAM page that has the evicted contents of an EPC page, decrypt the DRAM page's contents into the EPC page, and restore the corresponding EPCM entry. The only difference between ELDU and ELDB is that the latter sets the BLOCKED bit in the page's EPCM entry, whereas the former leaves it cleared.

ELDU and ELDB resemble ECREATE and EADD, in the sense that they populate a free EPC page. Since the page that they operate on was free, the SGX security model predicates that no TLB entries can possibly target it. Therefore, these instructions do not require a mechanism similar to EBLOCK or ETRACK.

5.5.2 The Version Array (VA)

When EWB evicts the contents of an EPC, it creates an 8-byte nonce (x 3.1.4) that Intel's documentation calls a page version. SGX's freshness guarantees are built on the assumption that nonces are stored securely, so EWB stores the nonce that it creates inside a Version Array (VA).

Version Arrays are EPC pages that are dedicated to storing nonces generated by EWB. Each VA is divided into slots, and each slot is exactly large enough to store one nonce. Given that the size of an EPC page is 4KB, and each nonce occupies 8 bytes, it follows that each VA has 512 slots.

VA pages are allocated using the EPA instruction, which takes in the virtual address of a free EPC page, and turns it into a Version Array with empty slots. VA pages are identified by the PT VA type in their EPCM entries. Like SECS pages, VA pages have the ENCLAVEADDRESS fields in their EPCM entries set to zero, and cannot be accessed directly by any software, including enclaves.

Unlike the other page types discussed so far, VA pages are not associated with any enclave. This means they can be deallocated via EREMOVE without any restriction. However, freeing up a VA page whose slots are in use effectively discards the nonces in those slots, which results in losing the ability to load the corresponding evicted pages back into the EPC. Therefore, it is unlikely that a correct OS implementation will ever call EREMOVE on a VA with non-free

slots.

According to the pseudo-code for EPA and EWB in the SDM, SGX uses the zero value to represent the free slots in a VA, implying that all the generated nonces have to be non-zero. This also means that EPA initializes a VA simply by zeroing the underlying EPC page. However, since software cannot access a VA's contents, neither the use of a special value, nor the value itself is architectural.

5.5.3 Enclave IDs

The EWB and ELDU / ELDB instructions use an enclave ID (EID) to identify the enclave that owns an evicted page. The EID has the same purpose as the ENCLAVESECS (x 5.1.2) field in an EPCM entry, which is also used to identify the enclave that owns an EPC page. This section explains the need for having two values represent the same concept by comparing the two values and their uses.

The SDM states that ENCLAVESECS field in an EPCM entry is used to identify the SECS of the enclave owning the associated EPC page, but stops short of describing its format. In theory, the ENCLAVESECS field can change its representation between SGX implementations since SGX instructions never expose its value to software.

However, we will later argue that the most plausible representation of the ENCLAVESECS field is the physical address of the enclave's SECS. Therefore, the ENCLAVESECS value associated with a given enclave will change if the enclave's SECS is evicted from the EPC and loaded back at a different location. It follows that the ENCLAVESECS value is only suitable for identifying an enclave while its SECS remains in the EPC.

According to the SDM, the EID field is a 64-bit field stored in an enclave's SECS. ECREATE's pseudocode in the SDM reveals that an enclave's ID is generated when the SECS is allocated, by atomically incrementing a global counter. Assuming that the counter does not roll over, this process guarantees that every enclave created during a power cycle has a unique EID.

Although the SDM does not specifically guarantee this, the EID field in an enclave's SECS does not appear to be modified by any instruction. This makes the EID's value suitable for identifying an enclave throughout its lifetime, even across evictions of its SECS page from the EPC.

5.5.4 Evicting an EPC Page

The system software evicts an EPC page using the EWB instruction, which produces all the data needed to restore the evicted page at a later time via the ELDU instruction, as shown in Figure 71.

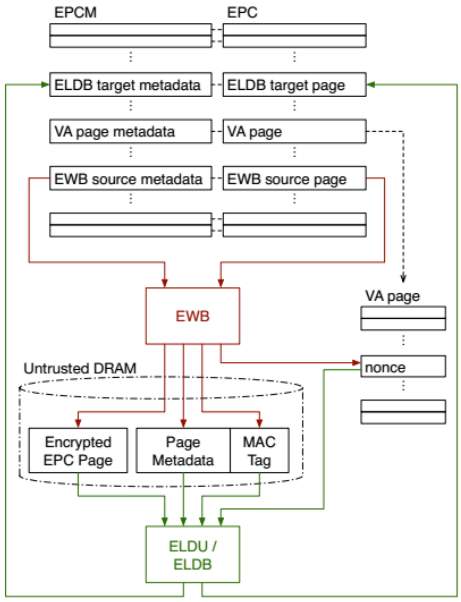


Figure 71*:* The EWB instruction outputs the encrypted contents of the evicted EPC page, a subset of the fields in the page's EPCM entry, a MAC tag, and a nonce. All this information is used by the ELDB or ELDU instruction to load the evicted page back into the EPC, with confidentiality, integrity and freshness guarantees.

EWB's output consists of an encrypted version of the evicted EPC page's contents, a subset of the fields in the EPCM entry corresponding to the page, the nonce discussed in x 5.5.2, and a message authentication code (MAC, x 3.1.3) tag. With the exception of the nonce, EWB writes its output in DRAM outside the PRM area, so the system software can choose to further evict it to disk.

The EPC page contents is encrypted, to protect the confidentiality of the enclave's data while the page is stored in the untrusted DRAM outside the PRM range. Without the use of encryption, the system software could learn the contents of an EPC page by evicting it from the EPC.

The page metadata is stored in a Page Information (PAGEINFO) structure, illustrated in Figure 72. This structure is similar to the PAGEINFO structure described in x 5.3.2 and depicted in Figure 64, except that the SECINFO field has been replaced by a PCMD field, which contains the virtual address of a Page Crypto Metadata (PCMD) structure.

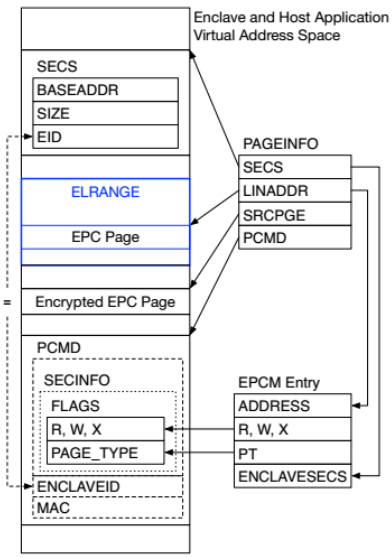


Figure 72: The PAGEINFO structure used by the EWB and ELDU / ELDB instructions

The LINADDR field in the PAGEINFO structure is used to store the ADDRESS field in the EPCM entry, which indicates the virtual address intended for accessing the page. The PCMD structure embeds the Security Information (SECINFO) described in x 5.3.2, which is used to store the page type (PT) and the access permission flags (R, W, X) in the EPCM entry. The PCMD structure also stores the enclave's ID (EID, x 5.5.3). These fields are later used by ELDU or ELDB to populate the EPCM entry for the EPC page that is reloaded.

The metadata described above is stored unencrypted, so the OS has the option of using the information inside as-is for its own bookkeeping. This has no negative impact on security, because the metadata is not confidential. In fact, with the exception of the enclave ID, all the metadata fields are specified by the system software when ECREATE is called. The enclave ID is only useful for identifying the enclave that the EPC page belongs to, and the system software already has this information as well.

Asides from the metadata described above, the PCMD structure also stores the MAC tag generated by EWB. The MAC tag covers the authenticity of the EPC page contents, the metadata, and the nonce. The MAC tag is checked by ELDU and ELDB, which will only load an evicted page back into the EPC if the MAC verification confirms the authenticity of the page data, metadata, and nonce. This security check protects against the page swapping attacks described in x 3.7.3.

Similarly to EREMOVE, EWB will only evict the EPC page holding an enclave's SECS if there is no other EPCM entry whose ENCLAVESECS field references the SECS. At the same time, as an optimization, the SGX implementation does not perform ETRACK-related checks when evicting a SECS. This is safe because a SECS is only evicted if the EPC has no pages belonging to the SECS' enclave, which implies that there isn't any TCS belonging to the enclave in the EPC, so no processor can be executing enclave code.

The pages holding Version Arrays can be evicted, just like any other EPC page. VA pages are never accessible by software, so they can't have any TLB entries pointing to them. Therefore, EWB evicts VA pages without performing any ETRACK-related checks. The ability to evict VA pages has profound implications that will be discussed in x 5.5.6.

EWB's data flow, shown in detail in Figure 73, has an aspect that can be confusing to OS developers. The instruction reads the virtual address of the EPC page to be evicted from a register (RBX) and writes it to the LINADDR field of the PAGEINFO structure that it is provided. The separate input (RBX) could have been removed by providing the EPC page's address in the LINADDR field.

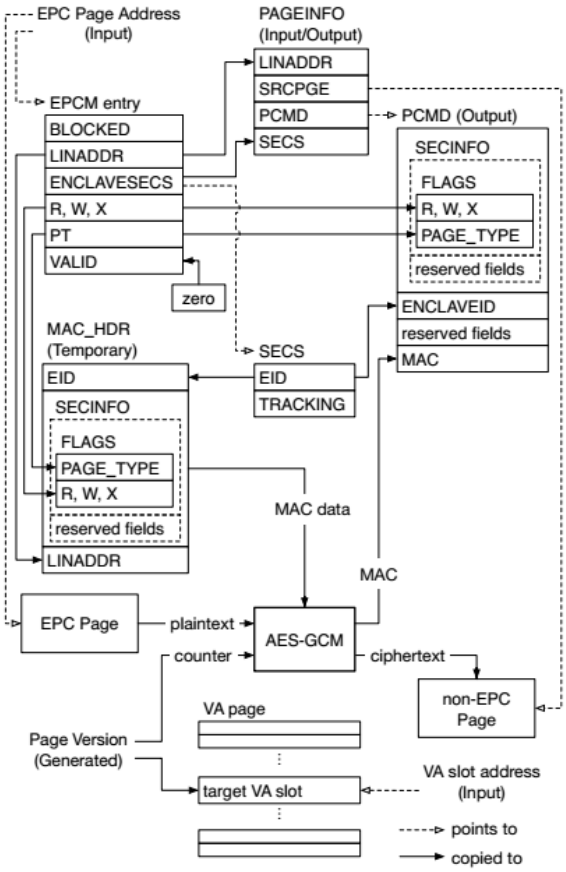


Figure 73: The data flow of the EWB instruction that evicts an EPC page. The page's content is encrypted in a non-EPC RAM page. A nonce is created and saved in an empty slot inside a VA page. The page's EPCM metadata and a MAC are saved in a separate area in non-EPC memory.

5.5.5 Loading an Evicted Page Back into EPC

After an EPC page belonging to an enclave is evicted, any attempt to access the page from enclave code will result in a Page Fault (#PF, x 2.8.2). The #PF will cause the logical processor to exit enclave mode via AEX (x 5.4.3), and then invoke the OS kernel's page fault handler.

Page faults receive special handling from the AEX process. While leaving the enclave, the AEX logic specifically checks if the hardware exception that triggered the AEX was #PF. If that is the case, the AEX implementation clears the least significant 12 bits of the CR2 register, which stores the virtual address whose translation caused a page fault.

In general, the OS kernel's page handler needs to be able to extract the virtual page number (VPN, x 2.5.1) from CR2, so that it knows which memory page needs to be loaded back into DRAM. The OS kernel may also be able to use the 12 least significant address bits, which are not part of the VPN, to better predict the application software's memory access patterns.

However, unlike the
bits that make up the VPN, the bottom 12 bits are not absolutely necessary for the fault handler to carry out its job. Therefore, SGX's AEX implementation clears these 12 bits, in order to limit the amount of information that is learned by the page fault handler.

When the OS page fault handler examines the address in the CR2 register and determines that the faulting address is inside the EPC, it is generally expected to use the ELDU or ELDB instruction to load the evicted page back into the EPC. If the outputs of EWB have been evicted from DRAM to a slower storage medium, the OS kernel will have to read the outputs back into DRAM before invoking ELDU / ELDB.

ELDU and ELDB verify the MAC tag produced by EWB, described in x 5.5.4. This prevents the OS kernel from performing the page swapping-based active address translation attack described in x 3.7.3.

5.5.6 Eviction Trees

The SGX design allows VA pages to be evicted from the EPC, just like enclave pages. When a VA page is evicted from EPC, all the nonces stored by the VA slots become inaccessible to the processor. Therefore, the evicted pages associated with these nonces cannot be restored by ELDB until the OS loads the VA page back into the EPC.

In other words, an evicted page depends on the VA page storing its nonce, and cannot be loaded back into the EPC until the VA page is reloaded as well. The dependency graph created by this relationship is a forest of eviction trees. An eviction tree, shown in Figure 74, has enclave EPC pages as leaves, and VA pages as inner nodes. A page's parent is the VA page that holds its nonce. Since EWB always outputs a nonce in a VA page, the root node of each eviction tree is always a VA page in the EPC.

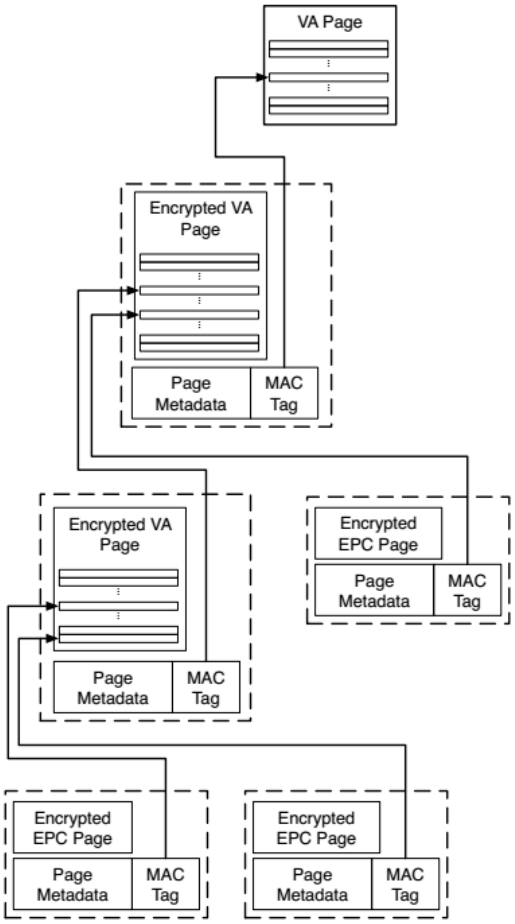


Figure 74: A version tree formed by evicted VA pages and enclave EPC pages. The enclave pages are leaves, and the VA pages are inner nodes. The OS controls the tree's shape, which impacts the performance of evictions, but not their correctness.

A straightforward inductive argument shows that when an OS wishes to load an evicted enclave page back into the EPC, it needs to load all the VA pages on the path from the eviction tree's root to the leaf corresponding to the enclave page. Therefore, the number of page loads required to satisfy a page fault inside the EPC depends on the shape of the eviction tree that contains the page. The SGX design leaves the OS in complete control of the shape of the eviction trees. This has no negative impact on security, as the tree shape only impacts the performance of the eviction scheme, and not its correctness.

5.6 SGX Enclave Measurement

SGX implements a software attestation scheme that follows the general principles outlined in *x* 3.3. For the purposes of this section, the most relevant principle is that a remote party authenticates an enclave based on its measurement, which is intended to identify the software that is executing inside the enclave. The remote party compares the enclave measurement reported by the trusted hardware with an expected measurement, and only proceeds if the two values match.

SGX 实施了一个遵循 3.3 节中概述的一般原则的软件认证方案。出于本节的目的，最相关的原则是远程方根据其 measurement 对安全区进行认证，该 measurement 旨在识

别在安全区内执行的软件。远程方将可信硬件报告的 measurement 与预期 measurement 值进行比较，并且仅在两个值匹配时才进行。

5.3 explains that an SGX enclave is built using the ECREATE (5.3.1), EADD (5.3.2) and EEXTEND instructions. After the enclave is initialized via EINIT (5.3.3), the instructions mentioned above cannot be used anymore. As the SGX measurement scheme follows the principles outlined in 3.3.2, the measurement of an SGX enclave is obtained by computing a secure hash (3.1.3) over the inputs to the ECREATE, EADD and EEXTEND instructions used to create the enclave and load the initial code and data into its memory. EINIT finalizes the hash that represents the enclave's measurement.

5.3 节解释了 SGX 飞地是使用 ECREATE（5.3.1），EADD（5.3.2）和 EEXTEND 指令构建的。通过 EINIT（ 5.3.3 节）初始化飞地后，上述指令不再使用。由于 SGX measurement 方案遵循 3.3.2 节中概述的原则，因此通过计算用于创建飞地的 ECREATE，EADD 和 EEXTEND 指令的输入的安全散列（ 3.1.3 节）来获得 SGX 飞地的 measurement 值。并将初始代码和数据加载到其内存中。EINIT 完成代表飞地 measurement 的哈希。

Along with the enclave's contents, the enclave author is expected to specify the sequence of instructions that should be used in order to create an enclave whose measurement will match the expected value used by the remote party in the software attestation process. The .so and .dll dynamically loaded library file formats, which are SGX's intended enclave delivery methods, already include informal specifications for loading algorithms. We expect the informal loading specifications to serve as the starting points for specifications that prescribe the exact sequences of SGX instructions that should be used to create enclaves from .so and .dll files.

除了飞地的内容外，enclave author 还应指定应该使用的指令序列，以便创建一个飞地，其 measurement 值将与软件认证过程中远程方使用的预期值相匹配。.so 和.dll 动态加载的库文件格式是 SGX 预期的安全区 delivery methods，已经包含了加载算法的 informal specifications。我们希望非正式加载规范可以作为规范的起点，这些规范规定了应该用于从.so 和.dll 文件创建安全区的 SGX 指令的确切顺序。

As argued in 3.3.2, an enclave's measurement is computed using a secure hashing algorithm, so the system software can only build an enclave that matches an expected measurement by following the exact sequence of instructions specified by the enclave's author.

如 3.3.2 节中所述，使用安全哈希算法计算飞地的 measurement 值，因此系统软件只能通过遵循 enclave author 指定的指令的确切顺序来构建与预期 measurement 匹配的飞地。

The SGX design uses the 256-bit SHA-2 [21] secure hash function to compute its measurements. SHA-2 is a block hash function (3.1.3) that operates on 64-byte blocks, uses a 32-byte internal state, and produces a 32- byte output. Each enclave's measurement is stored in the MRENCLAVE field of the enclave's SECS. The 32- byte field stores the internal state and final output of the 256-bit SHA-2 secure hash function.

SGX 设计使用 256 位 SHA-2 [21]安全散列函数来计算其 measurement 值。SHA-2 是一个块散列函数（3.1.3 节），它在 64 字节块上运行，使用 32 字节内部状态，并产生 32 字节输出。每个飞地的 measurement 值都存储在飞地 SECS 的 MRENCLAVE 字段中。32 字节字段存储 256 位 SHA-2 安全散列函数的内部状态和最终输出。

5.6.1 Measuring ECREATE

The ECREATE instruction, overviewed in 5.3.1, first initializes the MRENCLAVE field

in the newly created SECS using the 256-bit SHA-2 initialization algorithm, and then extends the hash with the 64-byte block depicted in Table 16.

在 5.3.1 节中概述的 ECREATE 指令首先使用 256 位 SHA-2 初始化算法初始化新创建的 SECS 中的 MRENCLAVE 字段，然后使用表 16 中所示的 64 字节块扩展散列。

Offset	Size	Description
0	8	"ECREATE\0"
8	8	SECS.SSAFRAMESIZE (§ 5.2.5)
16	8	SECS.SIZE (§ 5.2.1)
32	8	32 zero (0) bytes

Table 16: 64-byte block extended into MRENCLAVE by ECREATE

The enclave's measurement does not include the BASEADDR field. The omission is intentional, as it allows the system software to load an enclave at any virtual address inside a host process that satisfies the ELRANGE restrictions (5.2.1), without changing the enclave's measurement. This feature can be combined with a compiler that generates position-independent enclave code to obtain relocatable enclaves.

飞地的 measurement 不包括 BASEADDR 字段。省略是有意的，因为它允许系统软件在满足 ELRANGE 限制（5.2.1 节）的主机进程内的任何虚拟地址加载一个飞地，而不改变飞地的 measurement。此功能可以与生成位置无关的 enclave 代码的编译器结合使用，以获得可重定位的 enclaves。

The enclave's measurement includes the SSAFRAMESIZE field, which guarantees that the SSAs (5.2.5) created by AEX and used by EENTER (5.4.1) and ERESUME (5.4.4) have the size that is expected by the enclave's author. Leaving this field out of an enclave's measurement would allow a malicious enclave loader to attempt to attack the enclave's security checks by specifying a bigger SSAFRAMESIZE than the enclave's author intended, which could cause the SSA contents written by an AEX to overwrite the enclave's code or data.

飞地的测量包括 SSAFRAMESIZE 字段，该字段保证由 AEX 创建并由 EENTER（5.4.1 节）和 ERESUME（5.4.4 节）使用的 SSA（5.2.5 节）具有 enclave's author 所期望的 size。将该字段从飞地的 measurement 中移除将允许恶意飞地 loader 通过指定比 enclave's author 想要的更大的 SSAFRAMESIZE 来尝试攻击飞地的安全检查(security checks)，这可能导致 AEX 写入的 SSA 内容覆盖飞地的代码或数据。

5.6.2 Measuring Enclave Attributes

The enclave's measurement does not include the enclave attributes (5.2.2), which are specified in the ATTRIBUTES field in the SECS. Instead, it is included directly in the information that is covered by the attestation signature, which will be discussed in 5.8.1.

飞地的 measurement 不包括飞地属性（5.2.2 节），这些属性在 SECS 的 ATTRIBUTES 字段中指定。相反，它直接包含在证明签名所涵盖的信息中，将在 5.8.1 节中讨论。

The SGX software attestation definitely needs to cover the enclave attributes. For example, if XFRM (5.2.2, 5.2.5) would not be covered, a malicious enclave loader could attempt to subvert an enclave's security checks by setting XFRM to a value that enables architectural extensions that change the semantics of instructions used by the enclave, but still produces an XSAVE output that fits in SSAFRAMESIZE.

SGX 软件认证肯定需要 cover 飞地属性。例如，如果不覆盖 XFRM（5.2.2 节,5.2.5 节），则恶意安全区加载程序可以尝试通过将 XFRM 设置为启用体系结构扩展的值来破

坏飞地的安全检查，该体系结构扩展更改了所使用的指令的语义。飞地，但仍然产生适合 SSAFRAMESIZE 的 XSAVE 输出。

The special treatment applied to the ATTRIBUTES SECS field seems questionable from a security standpoint, as it adds extra complexity to the software attestation verifier, which translates into more opportunities for exploitable bugs. This decision also adds complexity to the SGX software attestation design, which is described in 5.8.

从安全角度来看，应用于 ATTRIBUTES SECS 领域的特殊处理似乎是有问题的，因为它增加了软件认证验证程序的额外复杂性，从而转化为可利用漏洞的更多机会。此决定还增加了 SGX 软件认证设计的复杂性，该设计在 5.8 中描述。

The most likely reason why the SGX design decided to go this route, despite the concerns described above, is the wish to be able to use a single measurement to represent an enclave that can take advantage of some architectural extensions, but can also perform its task without them.

尽管存在上述问题，SGX 设计决定走这条路线的最可能原因是希望能够使用单一 e measurement 来表示飞地，其既可以利用某些架构扩展，但也可以在没有它们的情况下执行任务。

Consider, for example, an enclave that performs image processing using a library such as OpenCV, which has routines optimized for SSE and AVX, but also includes generic fallbacks for processors that do not have these features. The enclave's author will likely wish to allow an enclave loader to set bits 1 (SSE) and 2 (AVX) to either true or false. If ATTRIBUTES (and, by extension, XFRM) was a part of the enclave's measurement, the enclave author would have to specify that the enclave has 4 valid measurements. In general, allowing n architectural extensions to be used independently will result in 2^n valid measurements.

例如，考虑使用诸如 OpenCV 之类的库执行图像处理的飞地，其具有针对 SSE 和 AVX 优化的例程，但还包括用于没有这些特征的处理器的通用 fallbacks。Enclave's author 可能希望允许 enclave loader 将位 1（SSE）和 2（AVX）设置为真或假。如果 ATTRIBUTES（以及扩展，XFRM）是飞地 measurement 的一部分，enclave's author 必须指定飞地有 4 个有效测量。通常，允许独立使用 n 个架构扩展将导致2^n个有效测量。

5.6.3 Measuring EADD

The **EADD** instruction, described in 5.3.2, extends the SHA-2 hash in MRENCLAVE with the 64-byte block shown in Table 17.

5.3.2 节中描述的 EADD 指令使用表 17 中所示的 64 字节块扩展了 MRENCLAVE 中的 SHA-2 哈希。

Offset	Size	Description
0	8	"EADD\0\0\0\0"
8	8	ENCLAVEOFFSET
16	48	SECINFO (first 48 bytes)

Table 17: 64-byte block extended into MRENCLAVE by EADD. The ENCLAVEOFFSET is computed by subtracting the BASEADDR in the enclave's SECS from the LINADDR field in the PAGEINFO structure.

表 17：由 EADD 扩展到 MRENCLAVE 的 64 字节块。ENCLAVEOFFSET 是通过从 PAGEINFO 结构中的 LINADDR 字段中扣除（subtract）安全区 SECS 中的 BASEADDR 来计算的。

The address included in the measurement is the address where the EADDed page is expected to be mapped in the enclave's virtual address space. This ensures that the system software sets up the enclave's virtual memory layout according to the enclave author's specifications. If a malicious enclave loader attempts to set up the enclave's layout incorrectly, perhaps in order to mount an active address translation attack (3.7.2), the loaded enclave's measurement will differ from the measurement expected by the enclave's author.

Measurement 中包含的地址是 EADD 后的页面应该映射到飞地虚拟地址空间中映射的地址。这可确保系统软件根据 enclave's author 的规范设置飞地虚拟 memory layout。如果恶意 enclave loader 试图错误地设置飞地 layout，可能是为了安装有效的地址转换攻击（3.7.2 节），加载的飞地 measurement 将与 enclave's author 所期望的 measurement 不同。

The virtual address of the newly created page is measured relatively to the start of the enclave's ELRANGE. In other words, the value included in the measurement is LINADDR - BASEADDR. This makes the enclave's measurement invariant to BASEADDR changes, which is desirable for relocatable enclaves. Measuring the relative addresses still preserves all the information about the memory layout inside ELRANGE, and therefore has no negative security impact.

新创建的页面的虚拟地址 measured relatively to 飞地的起始 ELRANGE。换句话说，measurement 中包含的值是 LINADDR - BASEADDR。这使得飞地的 measurement 对 BASEADDR 变化不变，这对于可重新定位的飞地来说是理想的。测量相对地址仍然保留了 ELRANGE 内部 memory layout 的所有信息，因此没有负面的安全影响。

EADD also measures the first 48 bytes of the SECINFO structure (5.3.2) provided to EADD, which contain the page type (PT) and access permissions (R, W, X) field values used to initialize the page's EPCM entry. By the same argument as above, including these values in the measurement guarantees that the memory layout built by the system software loading the enclave matches the specifications of the enclave author.

EADD 还 measure 提供给 EADD 的 SECINFO 结构的前 48 个字节（5.3.2 节），其中包含用于初始化页面的 EPCM 条目的页面类型（PT）和访问权限（R，W，X）字段值。通过与上述相同的论证，在 measurement 中包括这些值保证了由加载飞地的系统软件构建的 memory layout 与 enclave author 的规范相匹配。

The EPCM field values mentioned above take up less than one byte in the SECINFO structure, and the rest of the bytes are reserved and expected to be initialized to zero. This leaves plenty of expansion room for future SGX features.

上面提到的 EPCM 字段值在 SECINFO 结构中占用少于一个字节，其余字节被保留并且预期初始化为零。这为未来的 SGX 功能留下了充足的扩展空间。

The most notable omission from Table 17 is the data used to initialize the newly created EPC page. Therefore, the measurement data contributed by EADD guarantees that the enclave's memory layout will have pages allocated with prescribed access permissions at the desired virtual addresses. However, the measurements don't cover the code or data loaded in these pages.

表 17 中最值得注意的遗漏是用于初始化新创建的 EPC 页面的数据。因此，EADD 提供的 measurement 数据保证了飞地的 memory layout 将在所需的虚拟地址处分配具有规定访问权限的页面。但是，measurement 不包括这些页面中加载的代码或数据。

For example, EADD's measurement data guarantees that an enclave's memory layout

consists of three executable pages followed by five writable data pages, but it does not guarantee that any of the code pages contains the code supplied by the enclave's author.

例如，EADD 的 measurement 数据保证了飞地的 memory layout 由三个可执行页面和五个可写数据页面组成，但它不保证任何代码页面都包含 enclave's author 提供的代码。

5.6.4 Measuring EEXTEND

The EEXTEND instruction exists solely for the reason of measuring data loaded inside the enclave's EPC pages. The instruction reads in a virtual address, and extends the enclave's measurement hash with the five 64-byte blocks in Table 18, which effectively guarantee the contents of a 256-byte chunk of data in the enclave's memory.

EEXTEND 指令仅用于 measure 飞地 EPC 页面内加载的数据。该指令读入一个虚拟地址，并使用表 18 中的五个 64 字节块扩展了飞地的 measurement 哈希，这有效地保证了飞地内存中 256 字节数据块的内容。

Offset	Size	Description
0	8	"EEXTEND\0"
8	8	ENCLAVEOFFSET
16	48	48 zero (0) bytes
64	64	bytes 0 - 64 in the chunk
128	64	bytes 64 - 128 in the chunk
192	64	bytes 128 - 192 in the chunk
256	64	bytes 192 - 256 in the chunk

Table 18: 64-byte blocks extended into MRENCLAVE by EEXTEND. The ENCLAVEOFFSET is computed by subtracting the BASEADDR in the enclave's SECS from the LINADDR field in the PAGEINFO structure.

表 18：EEXTEND 扩展到 MRENCLAVE 的 64 字节块。 ENCLAVEOFFSET 是通过从 PAGEINFO 结构中的 LINADDR 字段中扣除安全区 SECS 中的 BASEADDR 来计算的。

Before examining the details of EEXTEND, we note that SGX's security guarantees only hold when the contents of the enclave's key pages is measured. For example, EENTER (5.4.1) is only guaranteed to perform controlled jumps inside an enclave's code if the contents of all the Thread Control Structure (TCS, 5.2.4) pages are measured. Otherwise, a malicious enclave loader can change the OENTRY field (5.2.4, 5.4.1) in a TCS while building the enclave, and then a malicious OS can use the TCS to perform an arbitrary jump inside enclave code. By the same argument, all the enclave's code should be measured by EEXTEND. Any code fragment that is not measured can be replaced by a malicious enclave loader.

在检查 EEXTEND 的详细信息之前，我们注意到 SGX 的安全保证仅在 measure 飞地的关键页面内容时保留。例如，如果 measure 所有线程控制结构（TCS，5.2.4 节）页面的内容，则 EENTER（5.4.1 节）保证在飞地代码内执行受控跳转。否则，恶意安全区加载程序可以在构建安全区时更改 TCS 中的 OENTRY 字段（ 5.2.4，5.4.1 节），然后恶意操作系统可以使用 TCS 在安全区代码内执行任意跳转。根据同一论点，所有飞地的代码都应该由 EEXTEND 来衡量。任何未 measured 代码片段都可以被恶意安全区加载器替换。

Given these pitfalls, it is surprising that the SGX design opted to decouple the virtual address space layout measurements done by EADD from the memory content measurements done by EEXTEND.

鉴于这些缺陷，SGX 设计选择将 EADD 完成的虚拟地址空间布局测量与 EEXTEND 完成的存储器内容测量分离。

At a first pass, it appears that the decoupling only has one benefit, which is the ability to load un-measured user input into an enclave while it is being built. However, this benefit only translates into a small performance improvement, because enclaves can alternatively be designed to copy the user input from untrusted DRAM after being initialized. At the same time, the decoupling opens up the possibility of relying on an enclave that provides no meaningful security guarantees, due to not measuring all the important data via EEXTEND calls.

似乎解耦（decoupling）只有一个好处，即能够在构建时将未 measured 用户输入加载到飞地。然而，这种好处仅转化为小的性能改进，因为可以选择将飞地设计为在初始化之后从不受信任的 DRAM 复制用户输入。与此同时，由于没有通过 EEXTEND 调用 measure 所有重要数据，因此解耦开启了依赖于飞地的可能性，该飞地没有提供有意义的安全保障。

However, the real reason behind the EADD / EEXTEND separation is hinted at by the EINIT pseudo-code in the SDM, which states that the instruction opens an interrupt (x 2.12) window while it performs a computationally intensive RSA signature check. If an interrupt occurs during the check, EINIT fails with an error code, and the interrupt is serviced. This very unusual approach for a processor instruction suggests that the SGX implementation was constrained in respect to how much latency its instructions were allowed to add to the interrupt handling process.

然而，EADD / EEXTEND 分离背后的真正原因是由 SDM 中的 EINIT 伪代码暗示，该代码指出该指令在执行计算密集型 RSA 签名检查时打开中断（2.12 节）窗口。 如果在检查期间发生中断，EINIT 将失败并显示错误代码，并且中断服务。处理器指令的这种非常不寻常的方法表明 SGX 实现在其指令被允许添加到中断处理过程的延迟时间方面受到限制。

In light of the concerns above, it is reasonable to conclude that EEXTEND was introduced because measuring an entire page using 256-bit SHA-2 is quite time consuming, and doing it in EADD would have caused the instruction to exceed SGX's latency budget. The need to hit a certain latency goal is a reasonable explanation for the seemingly arbitrary 256-byte chunk size.

鉴于上述问题，可以合理地得出 EEXTEND 的结论是因为使用 256 位 SHA-2 测量整个页面非常耗时，而在 EADD 中进行操作会导致指令超出 SGX 的延迟预算。达到某个延迟目标的需要是对看似任意的 256 字节块大小的合理解释。

The EADD / EEXTEND separation will not cause security issues if enclaves are authored using the same tools that build today's dynamically loaded modules, which appears to be the workflow targeted by the SGX design. In this workflow, the tools that build enclaves can easily identify the enclave data that needs to be measured.

如果使用构建当今动态加载模块的相同工具创建安全区，则 EADD / EEXTEND 分离不会导致安全问题，这似乎是 SGX 设计所针对的工作流程。在此工作流程中，构建飞地的工具可以轻松识别需要 measured 的飞地数据。

It is correct and meaningful, from a security perspective, to have the message blocks

provided by EEXTEND to the hash function include the address of the 256-byte chunk, in addition to the contents of the data. If the address were not included, a malicious enclave loader could mount the memory mapping attack described in 3.7.2 and illustrated in Figure 54.

从安全角度来看，除了数据内容之外，将 EEXTEND 提供给哈希函数的消息块包括 256 字节块的地址是正确和有意义的。如果未包含地址，则恶意安全区加载程序可能会 load 3.7.2 节中描述的内存映射攻击，如图 54 所示。

More specifically, the malicious loader would EADD the errorOut page contents at the virtual address intended for disclose, EADD the disclose page contents at the virtual address intended for errorOut, and then EEXTEND the pages in the wrong order. If EEXTEND would not include the address of the data chunk that is measured, the steps above would yield the same measurement as the correctly constructed enclave.

更具体地说，恶意加载器会将 EADD 的 errorOut 页面内容放在虚拟地址上，用于在虚拟地址上 disclose, EADD the disclose 页面内容，用于 errorOut，然后以错误的顺序 EEXTEND 页面。如果 EEXTEND 不包括 measured 数据块的地址，则上述步骤将产生与正确构造的飞地相同的 measurement。

The last aspect of EEXTEND worth analyzing is its support for relocating enclaves. Similarly to EADD, the virtual address measured by EEXTEND is relative to the enclave's BASEADDR. Furthermore, the only SGX structure whose content is expected to be measured by EEXTEND is the TCS. The SGX design has carefully used relative addresses for all the TCS fields that represent enclave addresses, which are OENTRY, OFSBASGX and OGSBASGX.

值得分析的 EEXTEND 的最后一个方面是它对重新安置飞地的支持。与 EADD 类似，EEXTEND measured 虚拟地址 relative to 飞地的 BASEADDR。此外，唯一的由 EEXTEND 测量的 SGX 结构是 TCS。SGX 设计仔细使用了代表安全区地址的所有 TCS 字段的相对地址，即 OENTRY，OFSBASGX 和 OGSBASGX。

5.6.5 Measuring EINIT

The EINIT instruction (5.3.3) concludes the enclave building process. After EINIT is successfully invoked on an enclave, the enclave's contents are "sealed", meaning that the system software cannot use the EADD instruction to load code and data into the enclave, and cannot use the EEXTEND instruction to update the enclave's measurement.

EINIT 指令（5.3.3 节）总结了 enclave building 过程。在飞地成功调用 EINIT 后，飞地的内容被"密封"，这意味着系统软件不能使用 EADD 指令将代码和数据加载到飞地，并且不能使用 EEXTEND 指令更新飞地的 measurement。

EINIT uses the SHA-2 finalization algorithm (3.1.3) on the MRENCLAVE field of the enclave's SECS. After EINIT, the field no longer stores the intermediate state of the SHA-2 algorithm, and instead stores the final output of the secure hash function. This value remains constant after EINIT completes, and is included in the attestation signature produced by the SGX software attestation process.

EINIT 在飞地 SECS 的 MRENCLAVE 字段上使用 SHA-2 finalization 算法（3.1.3 节）。在 EINIT 之后，该字段不再存储 SHA-2 算法的中间状态，而是存储安全哈希函数的最终输出。在 EINIT 完成后，此值保持不变，并包含在 SGX 软件认证过程生成的认证签名中。

5.7 SGX Enclave Versioning Support

The software attestation model (3.3) introduced by the Trusted Platform Module (4.4) relies on a measurement (5.6), which is essentially a content hash, to identify the software inside a container. The downside of using content hashes for identity is that there is no relation between the identities of containers that hold different versions of the same software.

可信平台模块（4.4）引入的软件证明模型（3.3）依赖于 measurement（5.6），其基本上是内容哈希，以识别容器内的软件。使用内容哈希来识别身份的缺点是，容纳不同版本的同一软件的容器的身份之间没有关系。

In practice, it is highly desirable for systems based on secure containers to handle software updates without having access to the remote party in the initial software attestation process. This entails having the ability to migrate secrets between the container that has the old version of the software and the container that has the updated version. This requirement translates into a need for a separate identity system that can recognize the relationship between two versions of the same software. SGX supports the migration of secrets between enclaves that represent different versions of the same software, as shown in Figure 75.

实际上，非常希望基于安全容器的系统能够在初始软件认证过程中无需中访问远程方处理软件更新。这需要能够在具有旧版本软件的容器和具有更新版本的容器之间迁移秘密。该要求转化为需要单独的身份系统，该系统可以识别同一软件的两个版本之间的关系。SGX 支持在代表不同版本的同一软件的飞地之间迁移秘密，如图 75 所示。

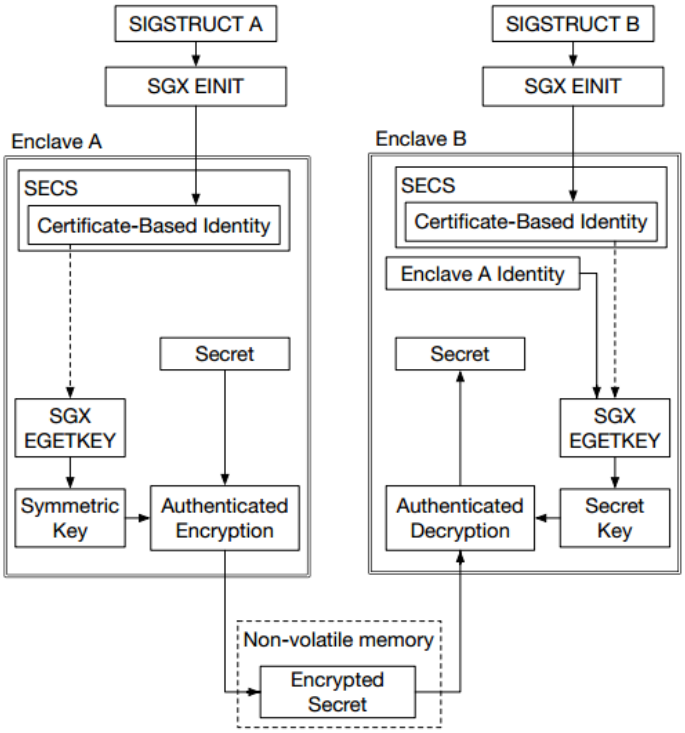


Figure 75: SGX has a certificate-based enclave identity scheme, which can be used to migrate secrets between enclaves that contain different versions of the same software module. Here, enclave A's secrets are migrated to enclave B.

图 75：SGX 具有基于证书的飞地身份方案，可用于在包含相同软件模块的不同版本的飞地之间迁移机密。在这里，飞地 A 的秘密被迁移到飞地 B.

The secret migration feature relies on a one-level certificate hierarchy (3.2.1), where each enclave author is a Certificate Authority, and each enclave receives a certificate from its author. These certificates must be formatted as Signature Structures (SIGSTRUCT), which are described

in 5.7.1. The information in these certificates is the basis for an enclave identity scheme, presented in 5.7.2, which can recognize the relationship between different versions of the same software.

秘密迁移功能依赖于一级证书层次结构（3.2.1 节），其中每个安全区作者是证书颁发机构，每个安全区都从其作者接收证书。这些证书必须格式化为签名结构（SIGSTRUCT），如 5.7.1 节中所述。 这些证书中的信息是 5.7.2 中提出的飞地识别方案的基础，该方案可以识别同一软件的不同版本之间的关系。

The EINIT instruction (5.3.3) examines the target enclave's certificate and uses the information in it to populate the SECS (5.1.3) fields that describe the enclave's certificate-based identity. This process is summarized in 5.7.4.

EINIT 指令（5.3.3 节）检查目标飞地的证书，并使用其中的信息填充描述飞地基于证书的身份的 SECS（5.1.3 节）字段。该过程在 5.7.4 节总结。

Last, the actual secret migration process is based on the key derivation service implemented by the EGETKEY instruction, which is described in 5.7.5. The sending enclave uses the EGETKEY instruction to obtain a symmetric key (3.1.1) based on its identity, encrypts its secrets with the key, and hands off the encrypted secrets to the untrusted system software. The receiving enclave passes the sending enclave's identity to EGETKEY, obtains the same symmetric key as above, and uses the key to decrypt the secrets received from system software.

最后，实际的秘密迁移过程基于 EGETKEY 指令实现的密钥派生服务，如 5.7.5 所述。发送飞地使用 EGETKEY 指令根据其标识获得对称密钥（3.1.1 节），用密钥加密其密钥，并将加密的密钥交给不受信任的系统软件。接收飞地将发送区域的标识传递给 EGETKEY，获得与上述相同的对称密钥，并使用密钥解密从系统软件接收的秘密。

The symmetric key obtained from EGETKEY can be used in conjunction with cryptographic primitives that protect the confidentiality (3.1.2) and integrity (3.1.3) of an enclave's secrets while they are migrated to another enclave by the untrusted system software. However, symmetric keys alone cannot be used to provide freshness guarantees (3.1), so secret migration is subject to replay attacks. This is acceptable when the secrets being migrated are immutable, such as when the secrets are encryption keys obtained via software attestation.

从 EGETKEY 获得的对称密钥可以与加密原语结合使用，加密原语保护飞地秘密的机密性（3.1.2 节）和完整性（3.1.3 节），同时由不受信任的系统软件迁移到另一个飞地。但是，单独的对称密钥不能用于提供新鲜度（freshness）保证（3.1），因此秘密迁移会受到重放攻击。当被迁移的秘密是不可变的时，例如当秘密是通过软件认证获得的加密密钥时，这是可接受的

5.7.1 Enclave Certificates

The SGX design requires each enclave to have a certificate issued by its author. This requirement is enforced by EINIT (5.3.3), which refuses to operate on enclaves without valid certificates.

SGX 设计要求每个 enclave 都有其 author 颁发的证书。此要求由 EINIT（5.3.3）强制执行，EINIT 拒绝在无有效证书的飞地上运行。

The SGX implementation consumes certificates formatted as Signature Structures (SIGSTRUCT), which are intended to be generated by an enclave building toolchain, as shown in Figure 76.

SGX 实现使用格式化为签名结构（SIGSTRUCT）的证书，这些证书旨在由飞地 building toolchain 生成，如图 76 所示。

A SIGSTRUCT certificate consists of metadata fields, the most interesting of which are presented in Table 19, and an RSA signature that guarantees the authenticity of the metadata, formatted as shown in Table 20. The semantics of the fields will be revealed in the following sections.

SIGSTRUCT 证书由元数据字段组成，其中最有趣的是表 19，以及保证元数据真实性的 RSA 签名，格式如表 20 所示。字段的语义将在以下部分中显示。

Field	Bytes	Description
ENCLAVEHASH	32	Must equal the enclave's measurement (§ 5.6).
ISVPRODID	32	Differentiates modules signed by the same private key.
ISVSVN	32	Differentiates versions of the same module.
VENDOR	4	Differentiates Intel enclaves.
ATTRIBUTES	16	Constrains the enclave's attributes.
ATTRIBUTEMASK	16	Constrains the enclave's attributes.

Table 19: A subset of the metadata fields in a SIGSTRUCT enclave certificate

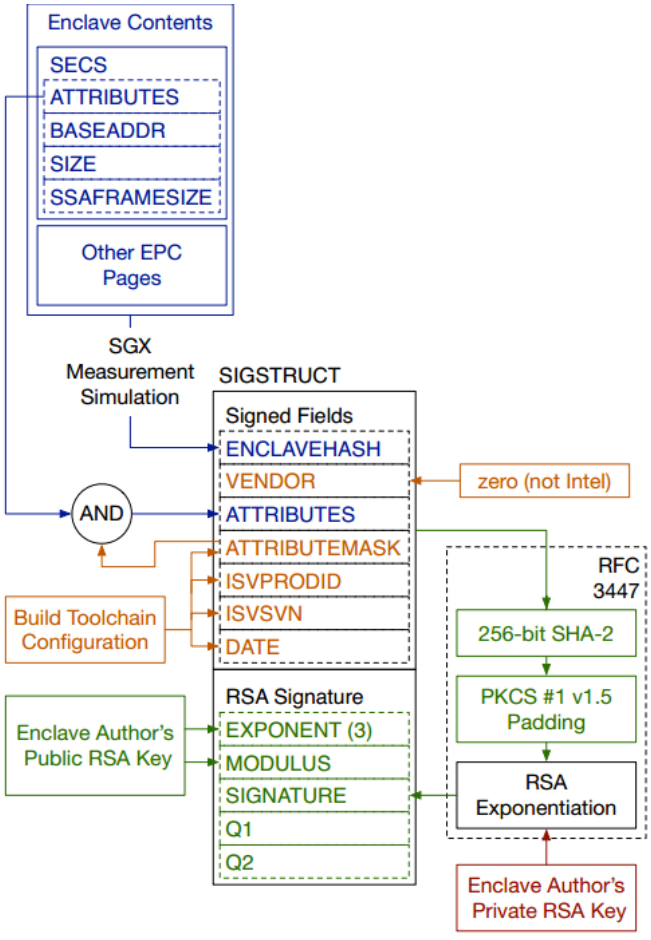


Figure 76: An enclave's Signature Structure (SIGSTRUCT) is intended to be generated by an

enclave building toolchain that has access to the enclave author's private RSA key.
图 76：安全区的签名结构（SIGSTRUCT）旨在由飞地 building toolchain 生成，该 building toolchain 可以访问 enclave author 的 RSA 私钥。

The enclave certificates must be signed by RSA signatures (3.1.3) that follow the method described in RFC 3447 [111], using 256-bit SHA-2 [21] as the hash function that reduces the input size, and the padding method described in PKCS #1 v1.5 [112], which is illustrated in Figure 45.

安全区证书必须由 RSA 签名（3.1.3 节）签署，签名遵循 RFC 3447 [111]中描述的方法，使用 256 位 SHA-2 [21]哈希函数减小输入大小，PKCS＃1 v1.5 [112]填充方法如图 45 所示。

The SGX implementation only supports 3072-bit RSA keys whose public exponent is 3. The key size is likely chosen to meet FIPS' recommendation [20], which makes SGX eligible for use in U.S. government applications. The public exponent 3 affords a simplified signature verification algorithm, which is discussed in 6.5. The simplified algorithm also requires the fields Q1 and Q2 in the RSA signature, which are also described in 6.5.

SGX 实现仅支持 3072 位 RSA 密钥，其公共指数为 3。密钥大小可能选择满足 FIPS 提议[20]，这使得 SGX 有资格在美国政府应用程序中使用。公共指数 3 提供简化的签名验证算法，其在 6.5 节中讨论。简化算法还需要 RSA 签名中的字段 Q1 和 Q2，这也在 6.5 节中描述。

Field	Bytes	Description
MODULUS	384	RSA key modulus
EXPONENT	4	RSA key public exponent
SIGNATURE	384	RSA signature (See § 6.5)
Q1	384	Simplifies RSA signature verification. (See § 6.5)
Q2	384	Simplifies RSA signature verification. (See § 6.5)

Table 20: The format of the RSA signature used in a SIGSTRUCT enclave certificate

5.7.2 Certificate-Based Enclave Identity

An enclave's identity is determined by three fields in its certificate (5.7.1): the modulus of the RSA key used to sign the certificate (MODULUS), the enclave's product ID (ISVPRODID) and the security version number (ISVSVN).

飞地的 identity 由其证书中的三个字段（5.7.1 节）确定：用于签署 identity 的 RSA 密钥的模数（MODULUS），飞地的产品 ID（ISVPRODID）和安全版本号（ISVSVN）。

The public RSA key used to issue a certificate identifies the enclave's author. All RSA keys used to issue enclave certificates must have the public exponent set to 3, so they are only differentiated by their moduli. SGX does not use the entire modulus of a key, but rather a 256-bit SHA-2 hash of the modulus. This is called a signer measurement (MRSIGNER), to parallel the name of enclave measurement (MRENCLAVE) for the SHA-2 hash that identifies an enclave's contents.

用于颁发证书的 RSA 公钥标识了 enclave author。用于发布飞地证书的所有 RSA 密钥必须将公共指数设置为 3，因此它们仅通过其模数进行区分。SGX 不使用密钥的整个模数，而是使用模数的 256 位 SHA-2 哈希。这称为 signer measurement（MRSIGNER），对比于标识飞地内容的 SHA-2 哈希的 enclave measurement（MRENCLAVE）。

The SGX implementation relies on a hard-coded MRSIGNER value to recognize certificates issued by Intel. Enclaves that have an Intel-issued certificate can receive additional privileges, which are discussed in 5.8.

SGX 实施依赖于硬编码的 MRSIGNER 值来识别英特尔颁发的证书。具有 Intel 颁发的证书的飞地可以获得其他权限，这些权限在 5.8 节中讨论。

An enclave author can use the same RSA key to issue certificates for enclaves that represent different software modules. Each module is identified by a unique Product ID (ISVPRODID) value. Conversely, all the enclaves whose certificates have the same ISVPRODID and are issued by the same RSA key (and therefore have the same MRENCLAVE) are assumed to represent different versions of the same software module. Enclaves whose certificates are signed by different keys are always assumed to contain different software modules.

Enclave author 可以使用相同的 RSA 密钥为代表不同软件 module 的飞地颁发证书。每个模块由唯一的产品 ID（ISVPRODID）值标识。相反，假定其证书具有相同 ISVPRODID 并由相同 RSA 密钥（因此具有相同 MRENCLAVE）发布的所有飞地代表同一软件 module 的不同版本。证书由不同密钥签名的飞地包含不同的软件 module。

Enclaves that represent different versions of a module can have different security version numbers (SVN). The SGX design disallows the migration of secrets from an enclave with a higher SVN to an enclave with a lower SVN. This restriction is intended to assist with the distribution of security patches, as follows.

代表 module 的不同版本的飞地可以具有不同的安全版本号（SVN）。 SGX 设计不允许将具有较高 SVN 的飞地的秘密迁移到具有较低 SVN 的飞地。此限制旨在帮助分发安全补丁，如下所示。

If a security vulnerability is discovered in an enclave, the author can release a fixed version with a higher SVN. As users upgrade, SGX will facilitate the migration of secrets from the vulnerable version of the enclave to the fixed version. Once a user’s secrets have migrated, the SVN restrictions in SGX will deflect any attack based on building the vulnerable enclave version and using it to read the migrated secrets.

如果在安全区中发现安全漏洞，author 可以发布具有更高 SVN 的固定版本。随着用户升级，SGX 将促进秘密从易受攻击版本的飞地迁移到固定版本。一旦用户的秘密迁移，SGX 中的 SVN 限制将根据构建易受攻击的飞地版本并使用它来读取迁移的秘密来转移任何攻击。

Software upgrades that add functionality should not be accompanied by an SVN increase, as SGX allows secrets to be migrated freely between enclaves with matching SVN values. As explained above, a software module’s SVN should only be incremented when a security vulnerability is found. SIGSTRUCT only allocates 2 bytes to the ISVSVN field, which translates to 65,536 possible SVN values. This space can be exhausted if a large team (incorrectly) sets up a continuous build system to allocate a new SVN for every software build that it produces, and each code change triggers a build.

添加功能的软件升级不应伴随 SVN 增加，因为 SGX 允许在具有匹配 SVN 值的飞地之间自由迁移秘密。如上所述，只有在发现安全漏洞时才应增加软件模块的 SVN。SIGSTRUCT 仅为 ISVSVN 字段分配 2 个字节，转换为 65,536 个可能的 SVN 值。如果大型团队（错误地）设置连续构建系统为其生成的每个软件构建分配新 SVN，并且每个代码更改触发构建，则此空间可能会耗尽。

5.7.3 CPU Security Version Numbers

The SGX implementation itself has a security version number (CPUSVN), which is used in the key derivation process implemented [138] by EGETKEY, in addition to the enclave's identity information. CPUSVN is a 128-bit value that, according to the SDM, reflects the processor's microcode update version.

SGX 实现本身具有安全版本号（CPUSVN），除了安全区的 identity 信息之外，它还用于 EGETKEY 实现的密钥推导过程[138]。根据 SDM，CPUSVN 是 128 位值，反映了处理器的微代码更新版本。

The SDM does not describe the structure of CPUSVN, but it states that comparing CPUSVN values using integer comparison is not meaningful, and that only some CPUSVN values are valid. Furthermore, CPUSVNs admit an ordering relationship that has the same semantics as the ordering relationship between enclave SVNs. Specifically, an SGX implementation will consider all SGX implementations with lower SVNs to be compromised due to security vulnerabilities, and will not trust them.

SDM 没有描述 CPUSVN 的结构，但是它声明使用整数比较来比较 CPUSVN 值没有意义，并且只有一些 CPUSVN 值是有效的。此外，CPUSVN 允许一种排序关系，其具有与飞地 SVN 之间的排序关系相同的语义。具体而言，SGX 实施将考虑低 SVN 的 SGX 实施由于安全漏洞而受到损害，并且不会信任它们。

An SGX patent [138] discloses that CPUSVN is a concatenation of small integers representing the SVNs of the various components that make up SGX's implementation. This structure is consistent with all the statements made in the SDM.

SGX 专利[138]公开了 CPUSVN 是小整数的串联，表示构成 SGX 实现的各种组件的 SVN。此结构与 SDM 中的所有语句一致。

5.7.4 Establishing an Enclave's Identity

When the EINIT (5.3.3) instruction prepares an enclave for code execution, it also sets the SECS (5.1.3) fields that make up the enclave's certificate-based identity, as shown in Figure 77.

当 EINIT（5.3.3）指令为代码执行准备一个飞地时，它还设置构成飞地基于证书的标识的 SECS（5.1.3）字段，如图 77 所示。

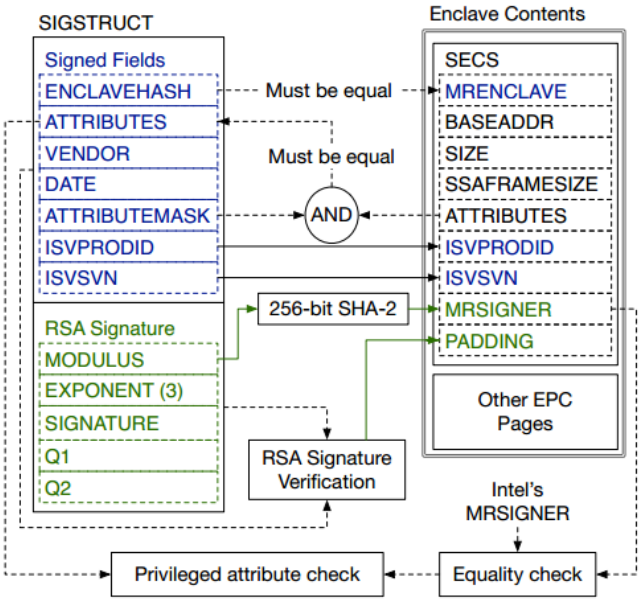


Figure 77: EINIT verifies the RSA signature in the enclave's certificate. If the certificate is valid, the information in it is used to populate the SECS fields that make up the enclave's certificate-based identity.

图 77：EINIT 验证飞地证书中的 RSA 签名。如果证书有效，则其中的信息用于填充构成飞地基于证书的标识的 SECS 字段。

EINIT requires the virtual address of the SIGSTRUCT certificate issued to the enclave, and uses the information in the certificate to initialize the certificate-based identity information in the enclave's SECS. Before using the information in the certificate, EINIT first verifies its RSA signature. The SIGSTRUCT fields Q1 and Q2, along with the RSA exponent 3, facilitate a simplified verification algorithm, which is discussed in 6.5.

EINIT 需要发送到安全区的 SIGSTRUCT 证书的虚拟地址，并使用证书中的信息初始化安全区 SECS 中基于证书的身份信息。在使用证书中的信息之前，EINIT 首先验证其 RSA 签名。SIGSTRUCT 字段 Q1 和 Q2 以及 RSA 指数 3 有助于简化验证算法，这将在 6.5 中讨论。

If the SIGSTRUCT certificate is found to be properly signed, EINIT follows the steps discussed in the following few paragraphs to ensure that the certificate was issued to the enclave that is being initialized. Once the checks have completed, EINIT computes MRSIGNER, the 256-bit SHA-2 hash of the MODULUS field in the SIGSTRUCT, and writes it into the enclave's SECS. EINIT also copies the ISVPRODID and ISVSVN fields from SIGSTRUCT into the enclave's SECS. As explained in x 5.7.2, these fields make up the enclave's certificate-based identity.

如果发现 SIGSTRUCT 证书已正确签名，EINIT 将遵循以下几段中讨论的步骤，以确保证书已颁发给正在初始化的飞地。一旦检查完成，EINIT 计算 MRSIGNER，即 SIGSTRUCT 中 MODULUS 字段的 256 位 SHA-2 哈希值，并将其写入飞地的 SECS。EINIT 还将 SIGSTRUCT 的 ISVPRODID 和 ISVSVN 字段复制到了飞地的 SECS 中。如 5.7.2 中所述，这些字段构成了飞地的 certificate-based identity.。

After verifying the RSA signature in SIGSTRUCT, EINIT copies the signature's padding into the PADDING field in the enclave's SECS. The PKCS #1 v1.5 padding scheme, outlined in Figure 45, does not involve randomness, so PADDING should have the same value for all

enclaves.

在验证 SIGSTRUCT 中 RSA 签名后，EINIT 将签名的填充复制到飞地 SECS 中的 PADDING 字段中。图 45 中概述的 PKCS＃1 v1.5 填充方案不涉及随机性，因此 PADDING 对于所有飞地应具有相同的值。

EINIT performs a few checks to make sure that the enclave undergoing initialization was indeed authorized by the provided SIGSTRUCT certificate. The most obvious check involves making sure that the MRENCLAVE value in SIGSTRUCT equals the enclave's measurement, which is stored in the MRENCLAVE field in the enclave's SECS.

EINIT 执行一些检查以确保正在进行初始化的飞地确实由提供的 SIGSTRUCT 证书授权。最明显的检查包括确保 SIGSTRUCT 中的 MRENCLAVE 值等于飞地的测量值，该测量值存储在飞地 SECS 的 MRENCLAVE 字段中。

However, MRENCLAVE does not cover the enclave's attributes, which are stored in the ATTRIBUTES field of the SECS. As discussed in 5.6.2, omitting ATTRIBUTES from MRENCLAVE facilitates writing enclaves that have optimized implementations that can use architectural extensions when present, and also have fallback implementations that work on CPUs without the extensions. Such enclaves can execute correctly when built with a variety of values in the XFRM (5.2.2, 5.2.5) attribute. At the same time, allowing system software to use arbitrary values in the ATTRIBUTES field would compromise SGX's security guarantees.

但是，MRENCLAVE 不包括飞地的属性，这些属性存储在 SECS 的属性字段中。正如 5.6.2 中所讨论的，从 MRENCLAVE 中省略 ATTRIBUTES 有助于编写具有优化实现的飞地，这些实现可以使用体系结构扩展，并且还具有在没有扩展的情况下在 CPU 上工作的回退（fallback）实现。当在 XFRM（5.2.2 节,5.2.5 节）属性中使用各种值构建时，此类飞地可以正确执行。同时，允许系统软件在 ATTRIBUTES 字段中使用任意值会损害 SGX 的安全保障。

When an enclave uses software attestation (3.3) to gain access to secrets, the ATTRIBUTES value used to build it is included in the SGX attestation signature (x 5.8). This gives the remote party in the attestation process the opportunity to reject an enclave built with an undesirable ATTRIBUTES value. However, when secrets are obtained using the migration process facilitated by certificate-based identities, there is no remote party that can check the enclave's attributes.

当飞地使用软件认证（3.3 节）来获取对机密的访问权限时，用于构建它的 ATTRIBUTES 值包含在 SGX 证明签名（5.8 节）中。这使得证明过程中的远程方有机会拒绝用不期望的 ATTRIBUTES 值构建的安全区。但是，当使用基于证书的身份促进的迁移过程获得秘密时，没有远程方可以检查安全区的属性。

The SGX design solves this problem by having enclave authors convey the set of acceptable attribute values for an enclave in the ATTRIBUTES and ATTRIBUTEMASK fields of the SIGSTRUCT certificate issued for the enclave. EINIT will refuse to initialize an enclave using a SIGSTRUCT if the bitwise AND between the ATTRIBUTES field in the enclave's SECS and the ATTRIBUTESMASK field in the SIGSTRUCT does not equal the SIGSTRUCT's ATTRIBUTES field. This check prevents enclaves with undesirable attributes from obtaining and potentially leaking secrets using the migration process.

SGX 设计通过让 enclave 在为飞地发布的 SIGSTRUCT 证书 ATTRIBUTES 和 ATTRIBUTEMASK 字段中传送一组可接受的属性值来解决这个问题。如果飞地的 SECS 中的 ATTRIBUTES 字段与 SIGSTRUCT 中的 ATTRIBUTESMASK 字段之间的按位 AND

不等于 SIGSTRUCT 的 ATTRIBUTES 字段，则 EINIT 将拒绝使用 SIGSTRUCT 初始化飞地。此检查可防止具有不良属性的飞地使用迁移过程获取并可能泄漏机密。

Any enclave author can use SIGSTRUCT to request any of the bits in an enclave's ATTRIBUTES field to be zero. However, certain bits can only be set to one for enclaves that are signed by Intel. EINIT has a mask of restricted ATTRIBUTES bits, discussed in 5.8. The EINIT implementation contains a hard-coded MRSIGNER value that is used to identify Intel's privileged enclaves, and only allows privileged enclaves to be built with an ATTRIBUTES value that matches any of the bits in the restricted mask. This check is essential to the security of the SGX software attestation process, which is described in 5.8.

任何 enclave author 都可以使用 SIGSTRUCT 来请求飞地的 ATTRIBUTES 字段中的任何位为零。但是，对于由英特尔签名的安全区，某些位只能设置为 1。EINIT 具有受限 ATTRIBUTES 位的掩码，在 5.8 节中讨论。EINIT 实现包含一个硬编码的 MRSIGNER 值，用于标识 Intel 的特权飞地，并且只允许使用与受限掩码中的任何位匹配的 ATTRIBUTES 值构建特权飞地。 此检查对于 SGX 软件证明过程的安全性至关重要，该过程在 5.8 中进行了描述。

Last, EINIT also inspects the VENDOR field in SIGSTRUCT. The SDM description of the VENDOR field in the section dedicated to SIGSTRUCT suggests that the field is essentially used to distinguish between special enclaves signed by Intel, which use a VENDOR value of 0x8086, and everyone else's enclaves, which should use a VENDOR value of zero. However, the EINIT pseudocode seems to imply that the SGX implementation only checks that VENDOR is either zero or 0x8086.

最后，EINIT 还检查了 SIGSTRUCT 中的 VENDOR 字段。SDM 描述表明，专用于 SIGSTRUCT 的部分中的 VENDOR 字段，主要用于区分由英特尔签署的特殊飞地，其使用 VENDOR 值 0x8086，其他飞地使用 VENDOR 值为零。但是，EINIT 伪代码似乎暗示 SGX 实现仅检查 VENDOR 是零还是 0x8086。

5.7.5 Enclave Key Derivation

SGX's secret migration mechanism is based on the symmetric key derivation service that is offered to enclaves by the EGETKEY instruction, illustrated in Figure 78.

SGX 的秘密迁移机制基于 EGETKEY 指令提供给飞地的对称密钥推导服务，如图 78 所示。

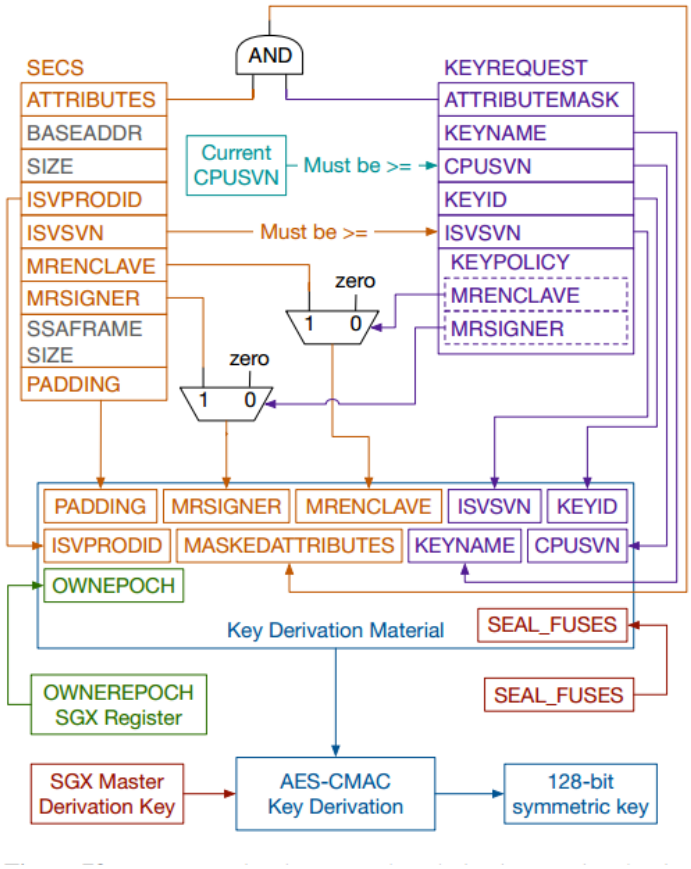


Figure 78: EGETKEY implements a key derivation service that is primarily used by SGX's secret migration feature. The key derivation material is drawn from the SECS of the calling enclave, the information in a Key Request structure, and secure storage inside the CPU's hardware.

图 78：EGETKEY 实现了主要由 SGX 的秘密迁移功能使用的密钥派生服务。密钥派生材料来自调用区域的 SECS，密钥请求结构中的信息以及 CPU 硬件内的安全存储。

The keys produced by EGETKEY are derived based on the identity information in the current enclave's SECS and on two secrets stored in secure hardware inside the SGX-enabled processor. One of the secrets is the input to a largely undocumented series of transformations that yields the symmetric key for the cryptographic primitive underlying the key derivation process. The other secret, referred to as the CR SEAL FUSES in the SDM, is one of the pieces of information used in the key derivation material.

EGETKEY 生成的密钥是根据当前飞地 SECS 中的 identity 信息以及存储在启用 SGX 的处理器内的安全硬件中的两个秘密得出的。其中一个秘密是对大量未记录的一系列转换的输入，这些转换产生了密钥派生过程背后的加密原语的对称密钥。另一个秘密，称为 SDM 中的 CR SEAL FUSES，是密钥派生材料中使用的信息之一。

The SDM does not specify the key derivation algorithm, but the SGX patents [110, 138] disclose that the keys are derived using the method described in FIPS SP 800-108 [34] using AES-CMAC [46] as a Pseudo-Random Function (PRF). The same patents state that the secrets used for key derivation are stored in the CPU's e-fuses, which is confirmed by the ISCA 2015 SGX tutorial [103].

SDM 没有指定密钥推导算法，但 SGX 专利[110,138]公开了密钥是使用 FIPS SP 800-108 [34]中描述的方法使用 AES-CMAC [46]作为伪随机数函数（PRF）得出的。相同的

专利声明用于密钥推导的秘密存储在 CPU 的 e-fuses，这由 ISCA 2015 SGX 教程[103]确认。

This additional information implies that all EGETKEY invocations that use the same key derivation material will result in the same key, even across CPU power cycles. Furthermore, it is impossible for an adversary to obtain the key produced from a specific key derivation material without access to the secret stored in the CPU's e-fuses. SGX's key hierarchy is further described in 5.8.2.

此附加信息意味着使用相同密钥派生材料的所有 EGETKEY 调用将产生相同的密钥，即使在 CPU power cycles 内也是如此。此外，敌手无法访问存储在 CPU 的 e-fuses 的秘密，不可能获得从特定密钥导出材料产生的密钥。 SGX 的密钥层次结构在 5.8.2 中进一步描述。

The following paragraphs discuss the pieces of data used in the key derivation material, which are selected by the Key Request (KEYREQUEST) structure shown in in Table 21,

以下段落讨论密钥派生材料中使用的数据，这些数据由表 21 中所示的密钥请求（KEYREQUEST）结构选择，

Field	Bytes	Description
KEYNAME	2	The desired key type; secret migration uses Seal keys
KEYPOLICY	2	The identity information (MRENCLAVE and/or MRSIGNER)
ISVSVN	2	The enclave SVN used in derivation
CPUSVN	16	SGX implementation SVN used in derivation
ATTRIBUTEMASK	16	Selects enclave attributes
KEYID	32	Random bytes

Table 21: A subset of the fields in the KEYREQUEST structure

The KEYNAME field in KEYREQUEST always participates in the key generation material. It indicates the type of the key to be generated. While the SGX design defines a few key types, the secret migration feature always uses Seal keys. The other key types are used by the SGX software attestation process, which will be outlined in 5.8.

KEYREQUEST 中的 KEYNAME 字段始终参与密钥生成材料。它指示要生成的密钥的类型。虽然 SGX 设计定义了几种密钥类型，但秘密迁移功能始终使用 Seal keys。其他密钥类型由将在 5.8 节中概述的 SGX 软件认证过程使用。

The KEYPOLICY field in KEYREQUEST has two flags that indicate if the MRENCLAVE and MRSIGNER fields in the enclave's SECS will be used for key derivation. Although the fields admits 4 values, only two seem to make sense, as argued below.

KEYREQUEST 中的 KEYPOLICY 字段有两个标志，指示安全区 SECS 中的 MRENCLAVE 和 MRSIGNER 字段是否将用于密钥派生。虽然这些字段允许 4 个值，但只有两个似乎有意义，如下所述。

Setting the MRENCLAVE flag in KEYPOLICY ties the derived key to the current enclave's measurement, which reflects its contents. No other enclave will be able to obtain the same key. This is useful when the derived key is used to encrypt enclave secrets so they can be stored by system software in non-volatile memory, and thus survive power cycles.

在 KEYPOLICY 中设置 MRENCLAVE 标志会将派生密钥与反映当前飞地内容的

measurement 值相关联。没有其他飞地能够获得相同的密钥。当派生密钥用于加密飞地 secrets 时，这很有用，因此它们可以由系统软件存储在非易失性存储器中，从而在 power cycles 中存活。

If the MRSIGNER flag in KEYPOLICY is set, the derived key is tied to the public RSA key that issued the enclave's certificate. Therefore, other enclaves issued by the same author may be able to obtain the same key, subject to the restrictions below. This is the only KEYPOLICY value that allows for secret migration.

如果设置了 KEYPOLICY 中的 MRSIGNER 标志，则派生密钥与发出飞地证书的 RSA 公钥相关联。因此，同一 author 发行的其他飞地可能能够获得相同的密钥，但受以下限制。这是允许秘密迁移的唯一 KEYPOLICY 值。

It makes little sense to have no flag set in KEYPOLICY. In this case, the derived key has no useful security property, as it can be obtained by other enclaves that are completely unrelated to the enclave invoking EGETKEY. Conversely, setting both flags is redundant, as setting MRENCLAVE alone will cause the derived key to be tied to the current enclave, which is the strictest possible policy.

在 KEYPOLICY 中没有设置标志是没有意义的。在这种情况下，派生密钥没有有用的安全属性，因为它可以通过与调用 EGETKEY 的飞地完全无关的其他飞地获得。相反，设置两个标志是多余的，因为单独设置 MRENCLAVE 将导致派生密钥绑定到当前的飞地，这是最严格的可能策略。

The KEYREQUEST structure specifies the enclave SVN (ISVSVN, 5.7.2) and SGX implementation SVN (CPUSVN, 5.7.3) that will be used in the key derivation process. However, EGETKEY will reject the derivation request and produce an error code if the desired enclave SVN is greater than the current enclave's SVN, or if the desired SGX implementation's SVN is greater than the current implementation's SVN.

KEYREQUEST 结构指定将在密钥派生过程中使用的飞地 SVN（ISVSVN，5.7.2 节）和 SGX 实现 SVN（CPUSVN，5.7.3 节）。但是，如果所需的飞地 SVN 大于当前飞地的 SVN，或者所需的 SGX 实现的 SVN 大于当前实现的 SVN，则 EGETKEY 将拒绝派生请求并产生错误代码。

The SVN restrictions prevent the migration of secrets from enclaves with higher SVNs to enclaves with lower SVNs, or from SGX implementations with higher SVNs to implementations with lower SVNs. 5.7.2 argues that the SVN restrictions can reduce the impact of security vulnerabilities in enclaves and in SGX's implementation.

SVN 限制防止秘密从具有较高 SVN 的安全区迁移到具有较低 SVN 的安全区，或者从具有较高 SVN 的 SGX 实施移植到具有较低 SVN 的实施。5.7.2 节认为 SVN 限制可以减少安全漏洞对飞地和新加坡交易所实施的影响。

EGETKEY always uses the ISVPRODID value from the current enclave's SECS for key derivation. It follows that secrets can never flow between enclaves whose SIGSTRUCT certificates assign them different Product IDs.

EGETKEY 始终使用当前飞地 SECS 的 ISVPRODID 值进行密钥派生。因此，秘密永远不会在其 SIGSTRUCT 证书为其分配不同产品 ID 的飞地之间流动。

Similarly, the key derivation material always includes the value of a 128-bit Owner Epoch (OWNEREPOCH) SGX configuration register. This register is intended to be set by the computer's firmware to a secret generated once and stored in non-volatile memory. Before the computer changes ownership, the old owner can clear the OWNEREPOCH from non-volatile

memory, making it impossible for the new owner to decrypt any enclave secrets that may be left on the computer.

类似地，密钥派生材料总是包括 128 位 Owner Epoch（OWNEREPOCH）SGX 配置寄存器的值。该寄存器旨在由计算机的固件设置为一次生成的秘密并存储在非易失性存储器中。在计算机更改所有权之前，旧所有者可以从非易失性存储器中清除 OWNEREPOCH，从而使新所有者无法解密可能留在计算机上的任何飞地秘密。

Due to the cryptographic properties of the key derivation process, outside observers cannot correlate keys derived using different OWNEREPOCH values. This makes it impossible for software developers to use the EGETKEY-derived keys described in this section to track a processor as it changes owners.

由于密钥派生过程的加密属性，外部观察者无法关联使用不同 OWNEREPOCH 值导出的密钥。这使得软件开发人员无法使用本节中描述的 EGETKEY 派生密钥来跟踪处理器，因为它更改了所有者。

The EGETKEY derivation material also includes a 256-bit value supplied by the enclave, in the KEYID field. This makes it possible for an enclave to generate a collection of keys from EGETKEY, instead of a single key. The SDM states that KEYID should be populated with a random number, and is intended to help prevent key wear-out.

EGETKEY 派生材料还包括由飞地在 KEYID 字段中提供的 256 位值。这使得飞地可以从 EGETKEY 生成密钥集合，而不是单个密钥。SDM 声明 KEYID 应填充随机数，旨在帮助防止密钥 wear-out。

Last, the key derivation material includes the bitwise AND of the ATTRIBUTES (5.2.2) field in the enclave's SECS and the ATTRIBUTESMASK field in the KEYREQUEST structure. The mask has the effect of removing some of the ATTRIBUTES bits from the key derivation material, making it possible to migrate secrets between enclaves with different attributes. 5.6.2 and 5.7.4 explain the need for this feature, as well as its security implications.

最后，密钥派生材料包括安全区 SECS 中 ATTRIBUTES（5.2.2）字段的 bitwise AND 和 KEYREQUEST 结构中的 ATTRIBUTESMASK 字段。掩码具有从密钥派生材料中移除一些 ATTRIBUTES 位的效果，使得可以在具有不同属性的飞地之间迁移秘密。5.6.2 节和 5.7.4 节解释了此功能的必要性及其安全隐患。

Before adding the masked attributes value to the key generation material, the EGETKEY implementation forces the mask bits corresponding to the INIT and DEBUG attributes (5.2.2) to be set. From a practical standpoint, this means that secrets will never be migrated between enclaves that support debugging and production enclaves.

在将掩码属性值添加到密钥生成材料之前，EGETKEY 实现强制设置与 INIT 和 DEBUG 属性（5.2.2）对应的掩码位。从实际角度来看，这意味着秘密永远不会在支持调试和生产飞地的飞地之间迁移。

Without this restriction, it would be unsafe for an enclave author to use the same RSA key to issue certificates to both debugging and production enclaves. Debugging enclaves receive no integrity guarantees from SGX, so it is possible for an attacker to modify the code inside a debugging enclave in a way that causes it to disclose any secrets that it has access to.

如果没有这个限制，enclave author 使用相同的 RSA 密钥向调试和 production 飞地发放证书将是不安全的。调试飞地不会从 SGX 接收完整性保证，因此攻击者可能会以某种方式修改调试飞地内的代码，从而泄露它可以访问的任何机密。

5.8 SGX Software Attestation

The software attestation scheme implemented by SGX follows the principles outlined in 3.3. An SGX-enabled processor computes a measurement of the code and data that is loaded in each enclave, which is similar to the measurement computed by the TPM (4.4). The software inside an enclave can start a process that results in an SGX attestation signature, which includes the enclave's measurement and an enclave message.

SGX 实施的软件认证计划遵循 3.3 节中概述的原则。SGX-enabled 的处理器计算 enclave 中加载的代码和数据的度量，这与 TPM（4.4）计算的 measurement 类似。安全区内的软件可以启动一个生成 SGX 认证签名的进程，其中包括 enclave 的 measurement 和 enclave 消息。

The cryptographic primitive used in SGX's attestation signature is too complex to be implemented in hardware, so the signing process is performed by a privileged Quoting Enclave, which is issued by Intel, and can access the SGX attestation key. This enclave is discussed in 5.8.2.

SGX 认证签名中使用的加密原语太复杂，无法在硬件中实现，因此签名过程由英特尔发布的可以访问 SGX 认证密钥的特权 Quoting Enclave 执行。Quoting Enclave 在 5.8.2 中讨论。

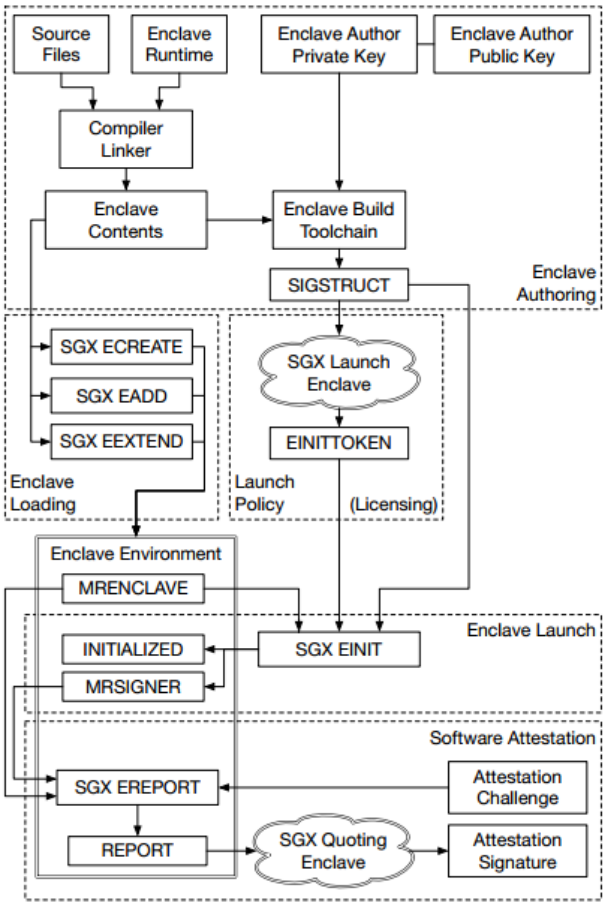


Figure 79: Setting up an SGX enclave and undergoing the software attestation process involves the SGX instructions EINIT and EREPORT, and two special enclaves authored by Intel, the SGX Launch Enclave and the SGX Quoting Enclave.

图 79：设置 SGX 飞地并进行软件认证过程涉及 SGX 指令 EINIT 和 EREPORT，以

及由 Intel 编写的两个特殊飞地，SGX Launch Enclave 和 SGX Quoting Enclave。

Pushing the signing functionality into the Quoting Enclave creates the need for a secure communication path between an enclave undergoing software attestation and the Quoting Enclave. The SGX design solves this problem with a local attestation mechanism that can be used by an enclave to prove its identity to any other enclave hosted by the same SGX-enabled CPU. This scheme, described in 5.8.1, is implemented by the EREPORT instruction.

将签名函数推入 Quoting Enclave，需要在进行软件认证的 enclave 和 Quoting Enclave 之间建立安全的通信路径。SGX 设计通过本地认证机制解决了这个问题，enclave 可以使用本地认证机制来向由同一个 SGX-enabled 的 CPU 控制的 enclave 证明其身份。该方案在 5.8.1 中描述，由 EREPORT 指令实现。

The SGX attestation key used by the Quoting Enclave does not exist at the time SGX-enabled processors leave the factory. The attestation key is provisioned later, using a process that involves a Provisioning Enclave issued by Intel, and two special EGETKEY (5.7.5) key types. The publicly available details of this process are summarized in 5.8.2.

在 SGX-enabled 的处理器出厂时，Quoting Enclave 使用的 SGX 认证密钥不存在。认证密钥随后使用涉及由英特尔发布的 Provisioning Enclave 的进程以及两个特殊的 EGETKEY（5.7.5）密钥类型提供。5.8.2 中总结了这一过程的公开可用细节。

The SGX Launch Enclave and EINITTOKEN structure will be discussed in 5.9. An enclave proves its identity to another target enclave via the EREPORT instruction shown in Figure 80. The SGX instruction produces an attestation Report (REPORT) that cryptographically binds a message supplied by the enclave with the enclave's measurement-based (5.6) and certificate-based (5.7.2) identities. The cryptographic binding is accomplished by a MAC tag (3.1.3) computed using a symmetric key that is only shared between the target enclave and the SGX implementation.

SGX 启动安全区和 EINITTOKEN 结构将在 5.9 中讨论。一个 enclave 通过图 80 所示的 EREPORT 指令向另一个目标 enclave 证明其身份。SGX 指令产生一个认证报告（REPORT），用于将 enclave 提供的消息与 enclave 的基于 measurement 的（5.6）和基于证书的（5.7.2）身份加密绑定。加密绑定是通过使用仅在目标 enclave 和 SGX 实现之间共享的对称密钥计算的 MAC 标签（3.1.3）来完成的。

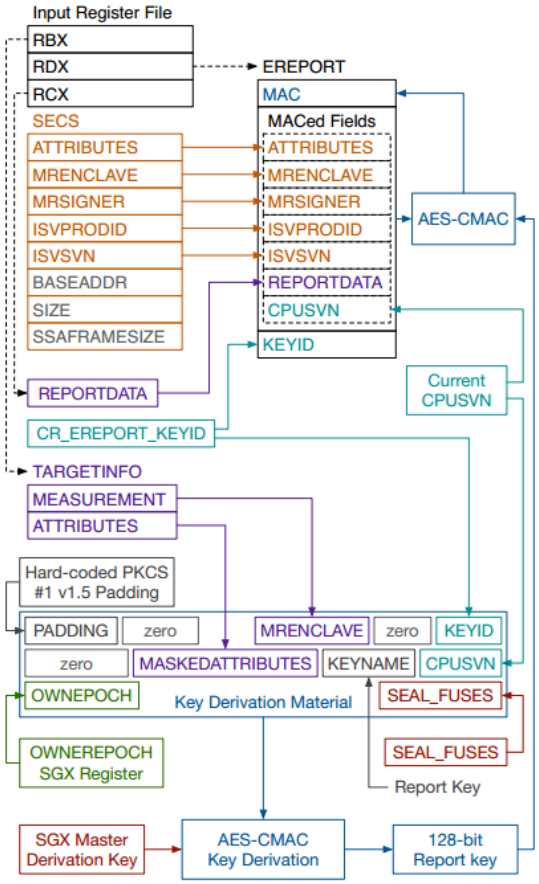


Figure 80: EREPORT data flow

The EREPORT instruction reads the current enclave's identity information from the enclave's SECS (5.1.3), and uses it to populate the REPORT structure. Specifically, EREPORT copies the SECS fields indicating the enclave's measurement (MRENCLAVE), certificate-based identity (MRSIGNER, ISVPRODID, ISVSVN), and attributes (ATTRIBUTES). The attestation report also includes the SVN of the SGX implementation (CPUSVN) and a 64-byte (512-bit) message supplied by the enclave.

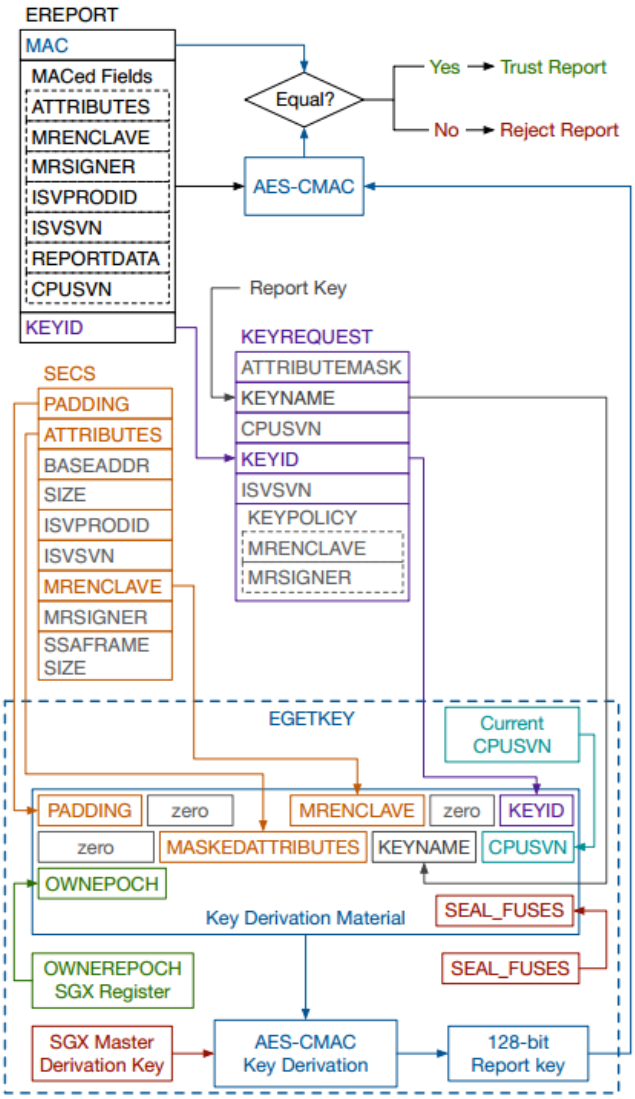
EREPORT 指令从 enclave 的 SECS（5.1.3）中读取当前 enclave 的身份信息，并使用它填充 REPORT 结构。具体而言，EREPORT 复制指示 enclave 的 measurement（MRENCLAVE），基于证书的身份（MRSIGNER，ISVPRODID，ISVSVN）和属性（ATTRIBUTES）的 SECS 字段。认证报告还包括 SGX 实施的 SVN（CPUSVN）和该 enclave 提供的 64 字节（512 位）消息。

The target enclave that receives the attestation report can convince itself of the report's authenticity as shown in Figure 81. The report's authenticity proof is its MAC tag. The key required to verify the MAC can only be obtained by the target enclave, by asking EGETKEY (5.7.5) to derive a Report key. The SDM states that the MAC tag is computed using a block cipher-based MAC (CMAC, [46]), but stops short of specifying the underlying cipher. One of the SGX papers [14] states that the CMAC is based on 128-bit AES.

接收证明报告的目标 enclave 可以说服自己报告的真实性，如图 81 所示。报告的真实性证明（authenticity proof）是其 MAC 标记。验证 MAC 所需的密钥只能通过目标 enclave 通过 EGETKEY（5.7.5）导出报告密钥获得。SDM 指出，MAC 标签是使用基于

块密码的 MAC（CMAC，[46]）计算出来的，但是没有指定底层密码。SGX 的一篇文章[14]指出 CMAC 基于 128 位 AES。

Figure 81*:* The authenticity of the REPORT structure created by EREPORT can and should be verified by the report's target enclave. The target's code uses EGETKEY to obtain the key used for the MAC tag embedded in the REPORT structure, and then verifies the tag.



The Report key returned by EGETKEY is derived from a secret embedded in the processor (5.7.5), and the key material includes the target enclave's measurement. The target enclave can be assured that the MAC tag in the report was produced by the SGX implementation, for the following reasons. The cryptographic properties of the underlying key derivation and MAC algorithms ensure that only the SGX implementation can produce the MAC tag, as it is the only entity that can access the processor's secret, and it would be impossible for an attacker to derive the Report key without knowing the processor's secret. The SGX design guarantees that the key produced by EGETKEY depends on the calling enclave's measurement, so only the target enclave can obtain the key used to produce the MAC tag in the report.

EGETKEY 返回的报告密钥来源于处理器中嵌入的秘密（5.7.5），密钥材料包括目标 enclave 的 measurement。由于以下原因，目标 enclave 可以确保报告中的 MAC 标签由 SGX 实施制作。底层的密钥派生和 MAC 算法的加密属性确保只有 SGX 实现才能生成

MAC 标记，因为它是唯一可以访问处理器秘密的实体，并且攻击者无法在不知道处理器的秘密的情况下派生 Report key。SGX 设计保证 EGETKEY 产生的密钥依赖于调用 enclave 的 measurement，因此只有目标 enclave 才能获得报告中用于生成 MAC 标签的密钥。

EREPORT uses the same key derivation process as EGETKEY does when invoked with KEYNAME set to the value associated with Report keys. For this reason, EREPORT requires the virtual address of a Report Target Info (TARGETINFO) structure that contains the measurement-based identity and attributes of the target enclave.

EREPORT 使用与 EGETKEY 在 KEYNAME 设置为与 Report keys 相关的值时调用的相同的密钥派生过程。因此，EREPORT 需要报告目标信息（TARGETINFO）结构的虚拟地址，TARGETINFO 结构包含目标安全区的 measurement-based 的身份和属性。

When deriving a Report key, EGETKEY behaves slightly differently than it does in the case of seal keys, as shown in Figure 81. The key generation material never includes the fields corresponding to the enclave's certificate-based identity (MRSIGNER, ISVPRODID, ISVSVN), and the KEYPOLICY field in the KEYREQUEST structure is ignored. It follows that the report can only be verified by the target enclave.

在导出报告密钥时，EGETKEY 的行为与密封密钥的行为稍有不同，如图 81 所示。密钥生成材料永远不会包含与 enclave 的基于证书的身份（MRSIGNER，ISVPRODID，ISVSVN）相对应的字段，并且 KEYREQUEST 结构中的 KEYPOLICY 字段将被忽略。因此，报告只能通过目标 enclave 进行核实。

Furthermore, the SGX implementation's SVN (CPUSVN) value used for key generation is determined by the current CPUSVN, instead of being read from the Key Request structure. Therefore, SGX implementation upgrades that increase the CPUSVN invalidate all outstanding reports. Given that CPUSVN increases are associated with security fixes, the argument in 5.7.2 suggests that this restriction may reduce the impact of vulnerabilities in the SGX implementation.

此外，用于密钥生成的 SGX 实现的 SVN（CPUSVN）值由当前 CPUSVN 确定，而不是从密钥请求结构中读取。因此，增加 CPUSVN 的 SGX 实施更新会使所有未完成的报告失效。鉴于 CPUSVN increases 与安全修复相关，5.7.2 中的论点表明，这种限制可能会降低 SGX 实施中漏洞的影响。

Last, EREPORT sets the KEYID field in the key generation material to the contents of an SGX configuration register (CR REPORT KEYID) that is initialized with a random value when SGX is initialized. The KEYID value is also saved in the attestation report, but it is not covered by the MAC tag.

最后，当 SGX 被初始化时，EREPORT 将密钥生成材料中的 KEYID 字段设置为 SGX 配置寄存器（CR REPORT KEYID）的内容，该寄存器使用随机值进行初始化。KEYID 值也保存在认证报告中，但不包含在 MAC 标签中。

5.8.2 Remote Attestation

The SDM paints a complete picture of the local attestation mechanism that was described in 5.8.1. The remote attestation process, which includes the Quoting Enclave and the underlying keys, is covered at a high level in an Intel publication [109]. This section's contents is based on the SDM, on one [14] of the SGX papers, and on the ISCA 2015 SGX tutorial [103].

SDM 描绘了 5.8.1 中描述的本地认证机制的完整图。远程认证过程包括 Quoting Enclave 和底层密钥。

SGX's software attestation scheme, which is illustrated in Figure 82, relies on a key generation facility and on a provisioning service, both operated by Intel.

如图 82 所示，SGX 的软件认证方案依赖于由英特尔运营的密钥生成设施以及 aprovisioning service。

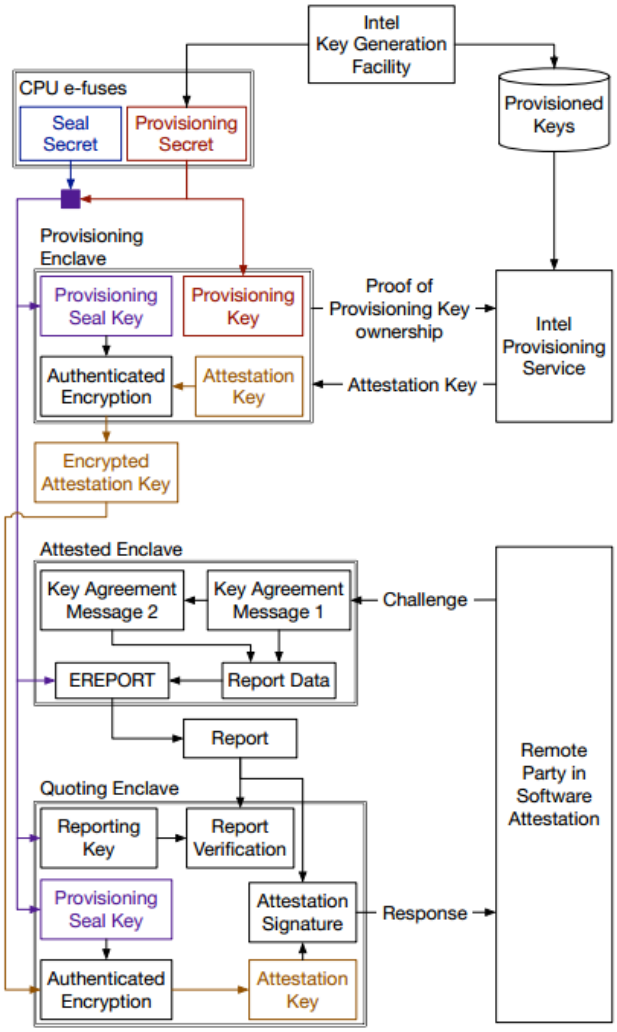


Figure 82: SGX's software attestation is based on two secrets stored in e-fuses inside the processor's die, and on a key received from Intel's provisioning service.

During the manufacturing process, an SGX-enabled processor communicates with Intel's key generation facility, and has two secrets burned into e-fuses, which are a one-time programmable storage medium that can be economically included on a high-performance chip's die. We shall refer to the secrets stored in e-fuses as the *Provisioning Secret* and the *Seal Secret*.

在制造过程中，SGX-enabled 的处理器与英特尔的密钥生成设备进行通信，并将两个 secrets 烧录到电子熔断器（e-fuses）中，这是一种可一次性编程的存储介质，可以经济地包含在高性能芯片的芯片中。我们将存储在 e-fuses 中的秘密称为 Provisioning Secret and the Seal Secret.

The Provisioning Secret is the main input to a largely undocumented process that outputs the SGX master derivation key used by EGETKEY, which was referenced in Figures 78, 79, 80, and 81.

Provisioning Secret 是一个很大程度上没有记录的进程的主要输入，它输出 EGETKEY 使用的 SGX 主衍生密钥，这在图 78,79,80 和 81 中被引用。

The Seal Secret is not exposed to software by any of the architectural mechanisms documented in the SDM. The secret is only accessed when it is included in the material used by the key derivation process implemented by EGETKEY (5.7.5). The pseudo-code in the SDM uses the CR SEAL FUSES register name to refer to the Seal Secret.

Seal Secret 不会通过 SDM 中记录的任何体系结构机制暴露给软件。只有当它被包含在由 EGETKEY（5.7.5）实现的密钥推导过程所使用的材料中时才可以访问该秘密。SDM 中的伪代码使用 CR SEAL FUSES 注册名称来指代 Seal Seal。

The names “Seal Secret” and “Provisioning Secret” deviate from Intel’s official documents, which confusingly use the “Seal Key” and “Provisioning Key” names to refer to both secrets stored in e-fuses and keys derived by EGETKEY.

“Seal Secret”和“Provisioning Secret”这两个名称背离了英特尔的官方文件，这些文件混淆地使用“Seal Key”和“Provisioning Key”名称来指代存储在 e-fuse 中的秘密和由 EGETKEY 派生的密钥。

The SDM briefly describes the keys produced by EGETKEY, but no official documentation explicitly describes the secrets in e-fuses. The description below is the only interpretation of all the public information sources that is consistent with all the SDM’s statements about key derivation.

SDM 简要描述了 EGETKEY 生成的密钥，但没有官方文档明确描述 e-fuses 的秘密。下面的描述是所有公共信息源的唯一解释，与所有 SDM 关于密钥推导的陈述一致。

The Provisioning Secret is generated at the key generation facility, where it is burned into the processor’s e-fuses and stored in the database used by Intel’s provisioning service. The Seal Secret is generated inside the processor chip, and therefore is not known to Intel. This approach has the benefit that an attacker who compromises Intel’s facilities cannot derive most keys produced by EGETKEY, even if the attacker also compromises a victim’s firmware and obtains the OWNEREPOCH (5.7.5) value. These keys include the Seal keys (5.7.5) and Report keys (5.8.1) introduced in previous sections.

Provisioning Secret 是在密钥生成工具中生成的，在该工具中它被烧入处理器的 e-fuses 并存储在 Intel 供应服务使用的数据库中。Seal Secret 是在处理器芯片内部生成的，因此英特尔并不知道。这种方法的好处在于，妥协于英特尔的设施的攻击者无法获得 EGETKEY 产生的大部分密钥，即使攻击者也 compromise 受害者的固件并获得 OWNEREPOCH（5.7.5）值。这些密钥包括前面部分介绍的 Seal keys（5.7.5）和 Report keys（5.8.1）。

The only documented exception to the reasoning above is the *Provisioning key*, which is effectively a shared secret between the SGX-enabled processor and Intel’s provisioning service. Intel has to be able to derive this key, so the derivation material does not include the Seal Secret or the OWNEREPOCH value, as shown in Figure 83.

唯一例外是 *Provisioning key*，它实际上是 SGX-enabled 的处理器和英特尔供应服务之间的共享密钥。英特尔必须能够推导出该密钥，因此派生材料不包括 Seal Secret 或 OWNEREPOCH 值，如图 83 所示。

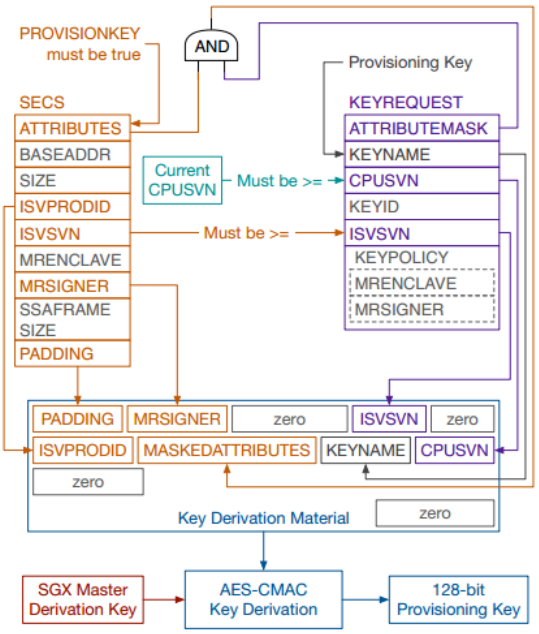


Figure 83: When EGETKEY is asked to derive a Provisioning key, it does not use the Seal Secret or OWNEREPOCH. The Provisioning key does, however, depend on MRSIGNER and on the SVN of the SGX implementation.

EGETKEY derives the Provisioning key using the current enclave's certificate-based identity (MRSIGNER, ISVPRODID, ISVSVN) and the SGX implementation's SVN (CPUSVN). This approach has a few desirable security properties. First, Intel's provisioning service can be assured that it is authenticating a Provisioning Enclave signed by Intel. Second, the provisioning service can use the CPUSVN value to reject SGX implementations with known security vulnerabilities. Third, this design admits multiple mutually distrusting provisioning services.

EGETKEY 使用当前安全区的基于证书的身份（MRSIGNER，ISVPRODID，ISVSVN）和 SGX 实施的 SVN（CPUSVN）派生 Provisioning key。这种方法有一些理想的安全属性。首先，英特尔的供应服务可以放心，它正在验证由英特尔签署的 Provisioning Enclave。其次，provisioning service 可以使用 CPUSVN 值来拒绝具有已知安全漏洞的 SGX 实现。第三，这种设计承认多种互不信任的供应服务。

EGETKEY only derives Provisioning keys for enclaves whose PROVISIONKEY attribute is set to true. 5.9.3 argues that this mechanism is sufficient to protect the computer owner from a malicious software provider that attempts to use Provisioning keys to track a CPU chip across OWNEREPOCH changes.

EGETKEY 仅为 PROVISIONKEY 属性设置为 true 的 enclave 派生 Provisioning keys。5.9.3 认为这种机制足以保护计算机所有者免受恶意软件提供者的影响，该恶意软件提供者试图通过 OWNEREPOCH 更改使用 Provisioning keys 来跟踪 CPU 芯片。

After the Provisioning Enclave obtains a Provisioning key, it uses the key to authenticate itself to Intel's provisioning service. Once the provisioning service is convinced that it is communicating to a trusted Provisioning enclave in the secure environment provided by a SGX-enabled processor, the service generates an *Attestation Key* and sends it to the Provisioning Enclave. The enclave then encrypts the Attestation Key using a *Provisioning Seal key*, and hands off the encrypted key to the system software for storage.

在 Provisioning Enclave 获得 Provisioning key 后，它将使用该密钥向 Intel 的 provisioning service 自我验证。一旦 provisioning service 确信它正在由 SGX-enabled 的处理器提供的安全环境中的可信 Provisioning enclave 进行通信，该服务会生成一个 *Attestation Key* 并将其发送到 Provisioning Enclave。然后，enclave 使用 *Provisioning Seal key* 对 Attestation Key 进行加密，然后将加密的密钥交给系统软件进行存储。

Provisioning Seal keys, are the last publicly documented type of special keys derived by EGETKEY, using the process illustrated in Figure 84. As their name suggests, Provisioning Seal keys are conceptually similar to the Seal Keys (5.7.5) used to migrate secrets between enclaves.

Provisioning Seal keys 是 EGETKEY 使用图 84 所示过程最后一次公开记录的特殊密钥类型。正如其名称所示，Provisioning Seal keys 在概念上与用于在 enclave 之间迁移秘密的 Seal Keys（5.7.5）类似。

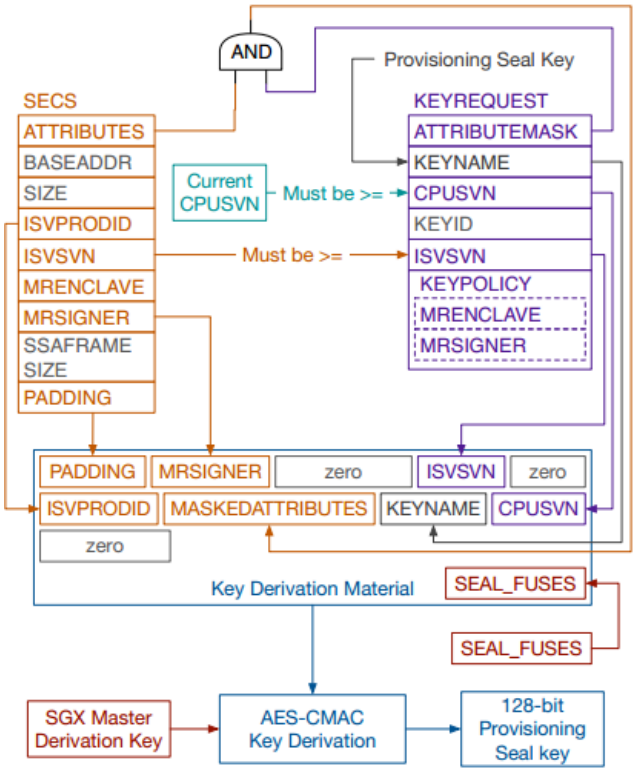


Figure 84: *The derivation material used to produce Provisioning Seal keys does not include the OWNEREPOCH value, so the keys survive computer ownership changes.*

The defining feature of Provisioning Seal keys is that they are not based on the OWNEREPOCH value, so they survive computer ownership changes. Since Provisioning Seal keys can be used to track a CPU chip, their use is gated on the PROVISIONKEY attribute, which has the same semantics as for Provisioning keys.

Provisioning Seal 密钥的定义特征是它们不基于 OWNEREPOCH 值，因此它们可以在计算机所有权更改后继续存在。由于 Provisioning Seal 密钥可用于跟踪 CPU 芯片，因此它们的使用将在 PROVISIONKEY 属性上进行选通，该属性与 Provisioning 键具有相同的语义。

Like Provisioning keys, Seal keys are based on the current enclave's certificate-based identity (MRSIGNER, ISVPROD, ISVSVN), so the Attestation Key encrypted by Intel's

Provisioning Enclave can only be decrypted by another enclave signed with the same Intel RSA key. However, unlike Provisioning keys, the Provisioning Seal keys are based on the Seal Secret in the processor's e-fuses, so they cannot be derived by Intel.

像 Provisioning keys 一样，Seal keys 基于当前 enclave 的基于证书的身份（MRSIGNER，ISVPROD，ISVSVN），因此英特尔 Provisioning Enclave 加密的 Attestation Key 只能用另一个使用相同 Intel RSA 密钥签名的 enclave 进行解密。 但是，与 Provisioning keys 不同，Provisioning Seal keys 基于处理器 e-fuses 中的密封秘密，因此它们不能由英特尔衍生。

When considered independently from the rest of the SGX design, Provisioning Seal keys have desirable security properties. The main benefit of these keys is that when a computer with an SGX-enabled processor exchanges owners, it does not need to undergo the provisioning process again, so Intel does not need to be aware of the ownership change. The confidentiality issue that stems from not using OWNEREPOCH was already introduced by Provisioning keys, and is mitigated using the access control scheme based on the PROVISIONKEY attribute that will be discussed in 5.9.3.

当从 SGX 设计的其余部分独立考虑时，Provisioning Seal keys 具有理想的安全属性。这些密钥的主要优点是，当具有 SGX-enabled 的处理器的计算机交换所有者时，不需要再次进行配置过程，因此英特尔不需要知道所有权更改。 源于不使用 OWNEREPOCH 的机密性问题已由 Provisioning keys 引入，并且使用基于将在 5.9.3 中讨论的 PROVISIONKEY 属性的访问控制方案进行缓解。

Similarly to the Seal key derivation process, both the Provisioning and Provisioning Seal keys depend on the bitwise AND of the ATTRIBUTES (5.2.2) field in the enclave's SECS and the ATTRIBUTESMASK field in the KEYREQUEST structure. While most attributes can be masked away, the DEBUG and INIT attributes are always used for key derivation.

与 Seal key 推导过程类似，Provisioning and Provisioning Seal keys 都依赖于 enclave 的 SECS 中的 ATTRIBUTES（5.2.2）字段与 KEYREQUEST 结构中的 ATTRIBUTESMASK 字段的按位与（AND）操作。 虽然大多数属性可以被屏蔽，但 DEBUG 和 INIT 属性总是用于密钥派生。

This dependency makes it safe for Intel to use its production RSA key to issue certificates for Provisioning or Quoting Enclaves with debugging features enabled. Without the forced dependency on the DEBUG attribute, using the production Intel signing key on a single debug Provisioning or Quoting Enclave could invalidate SGX's security guarantees on all the CPU chips whose attestation-related enclaves are signed by the same key. Concretely, if the issued SIGSTRUCT would be leaked, any attacker could build a debugging Provisioning or Quoting enclave, use the SGX debugging features to modify the code inside it, and extract the 128-bit Provisioning key used to authenticated the CPU to Intel's provisioning service.

这种依赖性使英特尔可以安全地使用其产生的 RSA 密钥为具有调试功能的 Provisioning 或 Quoted Enclaves 颁发证书。如果没有对 DEBUG 属性的强制依赖性，则在单个调试 Provisioning or Quoting Enclave 使用生产性英特尔签名密钥可能会使所有 CPU 证书上的 SGX 安全性保证失效，这些芯片的证明相关 enclaves 由相同密钥签名。具体而言，如果发布的 SIGSTRUCT 将被泄露，任何攻击者都可以构建调试 Provisioning 或 Quoting 安全区，使用 SGX 调试功能修改其中的代码，并提取将用于向 Intel 的 provisioning service 验证 CPU 的 128 位 Provisioning key。

After the provisioning steps above have been completed, the Quoting Enclave can be

invoked to perform SGX's software attestation. This enclave receives local attestation reports (5.8.1) and verifies them using the Report keys generated by EGETKEY. The Quoting Enclave then obtains the Provisioning Seal Key from EGETKEY and uses it to decrypt the Attestation Key, which is received from system software. Last, the enclave replaces the MAC in the local attestation report with an *Attestation Signature* produced with the Attestation Key.

在上述 provisioning 步骤完成后，可以调用 Quoting Enclave 以执行 SGX 的软件认证。该 enclave 接收本地认证报告（5.8.1）并使用 EGETKEY 生成的 Report keys 对其进行验证。Quoting Enclave 然后从 EGETKEY 获得 Provisioning Seal Key，并使用它来解密从系统软件接收到的 Attestation Key。最后，enclave 将本地认证报告中的 MAC 替换为使用 Attestation Key 生成的 *Attestation Signature*。

The SGX patents state that the name "Quoting Enclave" was chosen as a reference to the TPM (4.4)'s quoting feature, which is used to perform software attestation on a TPM-based system.

SGX 专利声称，"Quoting Enclave"的名称被选为 TPM（4.4）的 quoting feature 的参考，该功能用于在基于 TPM 的系统上执行软件认证。

The Attestation Key uses Intel's *Enhanced Privacy ID* (EPID) cryptosystem [26], which is a group signature scheme that is intended to preserve the anonymity of the signers. Intel's key provisioning service is the issuer in the EPID scheme, so it publishes the Group Public Key, while securely storing the Master Issuing Key. After a Provisioning Enclave authenticates itself to the provisioning service, it generates an EPID Member Private Key, which serves as the Attestation Key, and executes the EPID Join protocol to join the group. Later, the Quoting Enclave uses the EPID Member Private Key to produce Attestation Signatures.

Attestation Key 使用 Intel's *Enhanced Privacy ID*（EPID）密码系统[26]，这是一种旨在保护签名者匿名的群签名方案。英特尔的 key provisioning 服务是 EPID 方案中的发行方，因此它在发布 Group Public Key 的同时安全地存储 Master Issuing Key。在 Provisioning Enclave 向供应服务验证身份后，它会生成一个 EPID Member Private Key，该私钥用作证明密钥，并执行 EPID 加入协议以加入该 group。 之后，Quoting Enclave 使用 EPID Member Private Key 生成 Attestation Signatures。

The Provisioning Secret stored in the e-fuses of each SGX-enabled processor can be used by Intel to trace individual chips when a Provisioning Enclave authenticates itself to the provisioning service. However, if the EPID Join protocol is blinded, Intel's provisioning service cannot trace an Attestation Signature to a specific Attestation Key, so Intel cannot trace Attestation Signatures to individual chips.

当 Provisioning Enclave 向 provisioning service 认证自己时，英特尔可以使用存储在每个 SGX-enabled 的处理器的 e-fuses 中的 Provisioning Secret 来跟踪单个芯片。但是，如果 EPID 加入协议不起作用，则英特尔的 provisioning service 不能将 Attestation Signature 追踪到特定的 Attestation Key，因此英特尔不能将证明签名追踪到单个芯片。

Of course, the security properties of the description above hinge on the correctness of the proofs behind the EPID scheme. Analyzing the correctness of such cryptographic schemes is beyond the scope of this work, so we defer the analysis of EPID to the crypto research community.

当然，上述描述的安全属性取决于 EPID 方案背后证明的正确性。分析这些密码方案的正确性超出了本工作的范围，因此我们将 EPID 的分析推迟到密码研究社区。