机密容器跨平台可信热迁移技术

耿朝阳[1,2]，闵振南[1,2]，王文浩[1,2]

[1]中国科学院信息工程研究所 网络空间安全防御重点实验室 北京中国 100093

[2]中国科学院大学网络空间安全学院 北京中国 100093

摘要 机密容器通过采用 AMD SEV 等可信执行环境技术进行硬件隔离和加密，能够在操作系统或运行平台不可信的情况下保护容器中的数据，提高程序的安全性。在当前产业环境下，机密容器作为新兴的安全技术有着广阔的应用前景。随着产业的升级发展，会出现多个机密容器共享同一机密虚拟机的情况。当机密虚拟机的性能无法满足多个机密容器的需求时，就需要对一个或多个机密容器进行热迁移。然而，当前传统的容器热迁移技术不支持机密容器，因为机密容器内存状态是加密的，导致现有迁移技术无法直接应用。尽管 AMD SEV 提供了对机密虚拟机整体迁移的支持，但在一个机密虚拟机中包含多个机密容器的场景下，仍难以满足机密容器故障恢复、负载均衡等实际应用需求。为解决这一问题，论文结合机密容器技术的发展现状，并借鉴普通容器和虚拟机的热迁移方案，首次提出了在多机密容器运行环境下的一种机密容器跨平台可信热迁移技术的基础架构，以及在使用设备驱动处理请求过程中用于识别单一容器的新技术方案。在此基础上，论文给出了一种面向该架构的热迁移方法。通过检索机密虚拟机中的 Virtio 结构，在运行多个机密容器的机密虚拟机中获取单一机密容器虚拟设备状态的详细信息，在源平台保存该机密容器的虚拟设备状态，并在迁移的目标平台进行恢复。论文基于 AMD SEV-SNP 平台对提出的机密容器热迁移方案进行了原型实现，并集成至 CRIU 框架中。论文对提出的热迁移方案进行了评估，实验结果表明，本方案能够在可忽略的停机时间下完成对机密容器的可信热迁移。

关键词 机密容器 热迁移 可信执行环境

中图法分类号 TP309.2

A Trusted Cross-Platform Live Migration Technology for Confidential Containers

Geng Zhaoyang[1,2], Min Zhennan[1,2], Wang Wenhao[1,2]

[1]Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China

Abstract Confidential containers, which utilize hardware isolation and encryption through trusted execution environment technologies such as AMD SEV, can safeguard the data within the container when the operating system or runtime platform is untrustworthy. This enhances the overall security of the program. In the contemporary industrial landscape, confidential containers are gaining traction as emerging security technologies due to their wide-ranging application potential. As the industry undergoes advancements and evolution, there will likely be multiple confidential containers sharing a single confidential virtual machine. If the performance of this shared virtual machine falls short for multiple confidential containers, it becomes imperative to execute live migration on one or more of these containers. Traditional container live migration techniques, however, do not accommodate confidential containers because the memory state of these containers is encrypted. This makes existing migration methodologies unsuitable for direct application. While AMD SEV offers support for the comprehensive migration of confidential virtual machines, addressing the specific requirements of confidential container failure recovery, load balancing, and other applications remains challenging, especially when multiple confidential containers are housed within a single confidential virtual machine. To address this gap, this paper integrates the current state of confidential container technology and draws inspiration from the live migration strategies of ordinary containers and virtual machines. It introduces, for the first time, an infrastructure for cross-platform trusted live migration technology tailored for the cases when multiple confidential containers run within the same container runtime environment and proposes a novel technical approach for identifying individual container identifiers during the process of utilizing device drivers to handle requests. Building on this foundation, the paper presents a live migration methodology that aligns with this architecture. This paper presents a prototype of a confidential container live migration scheme, implemented on the AMD SEV-SNP platform and integrated into the CRIU framework. The scheme involves retrieving

Virtio structures from confidential virtual machines running multiple confidential containers to obtain detailed information about the states of these virtual devices. This information is then saved on the source platform and restored on the target platform where the confidential containers are migrated. The evaluation of the proposed live migration scheme demonstrates its effectiveness, with experimental results indicating that it can successfully complete trusted live migration of confidential containers with negligible downtime.
Key words confidential container, live migration, trusted execution environment

1 引言

尽管越来越多的产品使用云计算技术，但大多数厂商对云产品仍然缺乏信任，特别是一些金融和医疗类企业。这些企业的数据大多具有较高的保密性，因此很难将其放到不受信任的云环境中。可信计算基（TCB）指的是在计算机体系结构中一系列对计算机安全至关重要的软硬件的集合。机密计算的目的是将云服务提供者置于 TCB 之外，使数据敏感的企业能够将其设施放入云厂商提供的云服务环境中。方法是创建一个基于硬件的可信执行环境（TEE）[1-7]，主要功能是通过加密的方式将用户提供的机密数据与云服务厂商提供的云环境隔离开来。特别是，AMD EPYC CPU 能够在 TEE 内运行虚拟机（VM）[6]，并保护 VM 的完整性和机密性，防御不受信任的云服务厂商[8]。

Intel TDX[9]和 AMD SEV[10-12]是两个保护虚拟机安全性的可信执行环境。Intel TDX 利用 TDX 模块管理机密虚拟机，虚拟机监管程序（hypervisor）可以使用 TDX 定义的接口为其上运行的虚拟机提供各种资源。此外，AMD SEV 使用另一套硬件机制来保护机密虚拟机的安全[13-21]。例如，AMD SEV 依赖于平台安全处理器（PSP）来启用远程认证并防止客户机页被恶意修改。

机密容器（Confidential Container）是一种安全的容器技术，专注于保护应用程序和数据的隐私和安全。与普通容器直接运行于主机环境下不同，机密容器运行于一个单独的微型机密虚拟机中，拥有独立的操作系统内核，以及虚拟化层的安全隔离，例如 Kata Container[22]。机密容器可以在容器化的环境中运行应用程序和服务，同时利用加密技术保护容器中的数据和通信，以提高应用程序和数据的安全性。通常，机密容器采用硬件隔离和加密技术来实现安全保护，如使用 Intel SGX[23]等可信执行环境技术。与传统容器技术相比，机密容器提供了更安全和隐私保护的容器化解决方案。机密容器可在不信任的云环境或公共网络中运行，同时确保数据隐私和安全性。

在当前产业环境中，机密容器作为一项新兴的安全技术正受到关注和应用。随着产业的持续升级，可能出现多个机密容器共享同一台机密虚拟机的情况。然而，随着机密容器数量的增加，机密虚拟机的性能可能难以满足所有机密容器的需求。为了解决这一挑战，必须实施机密容器的热迁移。这意味着在机密虚拟机运行的情况下，将一个或多个机密容器无缝地迁移到另一台机密虚拟机中。这种操作旨在确保机密容器在需要时能够获得足够的计算资源，同时保持整个系统的运行连续性。

机密容器的热迁移涉及对容器状态、网络连接和存储的有效管理，要求精确控制容器的运行状态，以确保迁移过程的平稳进行。同时，对于机密容器的性能和安全性，在迁移过程中也需要考虑和保障。总体而言，机密容器的热迁移是当前产业环境中关键的安全技术操作，为确保系统高效运行和安全性提供了必要的手段。

然而，目前的容器热迁移技术例如 CRIU 仅支持对 Docker 这类直接运行于主机环境下的普通容器。机密容器运行于机密虚拟机内，并且普遍采用 Virtio 半虚拟化实现设备模拟。因此面向机密容器的热迁移方案除了需要迁移内存状态外，还需要保存和恢复使用 Virtio 标准的虚拟设备的状态。

由于机密容器的内存状态被加密且主机平台不可信，导致现有热迁移工具无法直接应用。另外一方面，尽管 AMD SEV 提供了对机密虚拟机整体迁

移的支持，但在一个机密虚拟机中包含多个机密容器的情况下，由于机密容器的 Virtio 设备状态由机密虚拟机统一管理，现有的迁移工具尚不支持识别 Virtio 设备状态的来源，因此无法对某个或某些机密容器进行单独热迁移。

因此，论文首次提出了一种面向多机密容器运行环境下的机密容器跨平台可信热迁移技术。为保证迁移过程中认证的安全性，以及与迁移方案与容器本身实现无关，论文提出在机密虚拟机的内部建立迁移代理和迁移模块的方法，建立安全传输通道，确保迁移过程中状态信息不被篡改解决了机密容器热迁移过程中内存状态迁移的安全性。另外，在机密容器虚拟设备状态保存与恢复方面，论文选择在现有机密虚拟机使用的 Virtio 结构中增加识别机密容器的标识，通过检索机密虚拟机中的 Virtio 结构，识别与特定容器关联的虚拟设备请求的状态，在源平台保存机密容器的 Virtio 设备状态，并在迁移的目标平台进行恢复。

论文提出的方案经过评估证实，在实际多机密容器运行时可以完成对单个容器及其 Virtio 设备状态进行可信热迁移，评估结果表明论文所提出的迁移方案与现有的迁移工具性能开销差距在 5%以内，并且在大规模数据库读写的迁移过程中的停机时间小于 1%，满足实际机密容器热迁移的需求，具有实际意义。

论文的主要贡献包括以下几个方面：

（1）针对基于 Virtio 设备虚拟化的机密容器，提出了一种增加标识以识别机密容器的迁移方案。通过该方案，首次实现了在多个机密容器环境下的单个机密容器热迁移方案，并成功将该系统集成到现有热迁移工具中。

（2）针对机密容器热迁移，提出了增加迁移代理和增加迁移模块的方式，以实现对机密容器更加高效的热迁移方案。

（3）以 AMD SEV 为平台对方案进行了实现，并设计了多组实验，重点针对 Virtio 设备性能和热迁移方案进行验证。实验结果表明，论文提出的机密容器热迁移方案具有较低的停机时间和良好的性能。

论文的整体结构框架分为五个部分：第二章介绍了现有的机密容器技术、普通容器热迁移方案以及相关背景，指出了现有方案的优势和不足。第三章提出了面向机密容器热迁移的系统框架，详细阐述了机密容器可信热迁移的具体流程。第四章对论文提出的热迁移方案进行了评估。最后，第五章总结全文。

2 背景

2.1 AMD SEV

AMD 安全加密虚拟化（SEV）是一种由 AMD 服务器级 EPYC 处理器架构支持的可信执行环境（TEE），其目标是为云客户提供机密虚拟机服务。在 SEV 的威胁模型中，其他虚拟机和云主机本身都被视为不可信，因为攻击者可以在 hypervisor 级别执行任意代码，甚至对机器（例如 DRAM 芯片[6]）进行物理访问。为了实现这一目标，SEV 引入了由 AMD 安全处理器（AMD-SP）和 AES 内存加密引擎组成的专用安全子系统，以保护正在使用中的数据。

启用 SEV 后，通过硬件内存加密提供加密隔离，以保护虚拟机的机密性。VM 的内存页始终以加密形式存储，而 VM 加密密钥由 AMD-SP 提供和保护。SEV 采用 128 位 AES 加密和 XOR 加密 XOR（XEX）加密模式，该模式结合了物理地址特定的调整，使得相同的明文在每个存储位置生成不同的密文[24]。

SEV-ES（Encrypted State）于 2017 年首次引入[25]。SEV-ES 增加了对 CPU 寄存器的额外保护。在 SEV-ES 之前，在虚拟机到虚拟机管理程序（VMEXIT）的世界切换期间，CPU 寄存器未加密地存储在虚拟机控制块（VMCB）中。在 SEV-ES 中，硬件自动加密指定虚拟机保存区域（VMSA）中的寄存器，并提供额外的完整性保护。此外，引入了一个客户机主机通信协议，用于需要向管理程序公开寄存器的指令（例如 CPUID、RDMSR 等）。客户机 VM 内的 VMM 通信处理程序（#VC 处理程序）通过在硬件的帮助下拦截这些指令，将必要的寄存器值传递到一个名为 Guest Host Communication Block（GHCB）的共享区域，通过 VMGEXIT 指令触发特殊的 VMEXIT，然后从 GHCB 读取生成的寄存器值。

SEV-SNP 是一种于 2020 年发布的安全技术[21]，

旨在为加密虚拟机提供额外的保护，并实现与 hypervisor 更强隔离。使用 SEV-SNP 需要同时启用 SEV 和 SEV-ES 功能。SEV-SNP 的主要功能是保护 VM 内存的完整性，以防止 hypervisor 攻击，这些攻击通常涉及到客户机数据损坏、别名、重放等各种攻击方式。为了实现这一点，SEV-SNP 引入了一种新的系统数据结构，称为反向映射表（RMP），用于对内存访问执行额外的安全检查。该结构维护主机物理地址到客户机物理地址的第二次转换，并跟踪内存页的所有权，从而防止管理程序修改或重新映射客户 VM 的私有内存[15]。除了内存保护外，SEV-SNP 还具有多个安全功能，例如新的虚拟机特权级别（VMPL）架构、中断注入限制和侧信道保护。这些功能旨在实现额外的使用模式和增强的安全保护。

2.2 机密容器

容器化技术已经成为云原生应用程序开发的核心。然而，这些技术也带来了一些数据隐私和安全性的挑战。传统容器在共享操作系统内核的基础上提供隔离，容器之间可以共享一些资源和权限。尽管容器提供了适度的隔离，但在某些情况下，攻击者可能通过容器之间的漏洞或攻击来获取未经授权的访问，这对处理敏感数据的应用程序构成潜在风险。

特别是在多云或混合云环境中，数据在不同云提供商之间传输时可能会面临风险。数据在传输过程中可能会暴露给不信任的网络，从而可能导致数据泄露或中间人攻击。

为了应对这些挑战，机密容器将机密计算概念应用于容器化环境。机密容器提供了一种方式，使容器内的数据和应用程序在运行时保持加密，从而提高了容器化应用程序的安全性。机密容器使用可信执行环境技术实现硬件级别的隔离和加密，确保数据在容器内得到充分保护。

以 Kata Container[22]为例，它利用机密虚拟机作为每个容器或一组容器的隔离边界。Kata Container 不仅保持了容器的轻量和高速特性，同时还充分利用了机密虚拟机的安全优势。

近年来，许多硬件厂商推出了自己的可信执行环境技术解决方案，如 Intel SGX[23]、AMD SEV[21]等。这意味着机密容器技术可以在多种硬件平台上得到广泛构建和应用。

相较于传统容器，机密容器提供了更高级别的安全性、数据隐私和运行时代码完整性。它们特别适用于需要处理敏感数据的应用程序，如金融服务、医疗保健、电子商务和政府部门等。机密容器使得容器化应用程序能够在不信任的云环境或公共网络中运行，同时保护数据隐私和安全性。

机密容器的几个重要特点包括：

（1）安全性：基于硬件可信执行环境，确保容器中的数据在运行时得到加密和保护，防止云提供商和高权限第三方直接访问或篡改容器中的数据。

（2）易用性：用户应用程序无需修改即可迁移到机密容器环境中，简化了迁移过程。

（3）信任独立：解决了租户和云提供商之间的信任依赖问题，使租户的数据对于云提供商不再是透明的。

（4）可自证性：用户可以通过远程证明等手段验证当前使用的容器环境的真实可信性，增加了安全性的可验证性。

机密容器技术代表了数据隐私和安全性领域的一个重要发展趋势，为敏感数据的处理提供了更高级别的保护。随着云计算的普及，机密容器技术将继续发展，为各行各业和组织提供更强大的数据隐私和安全性保障。

2.3 机密容器迁移及挑战

虚拟机管理程序等虚拟化技术在共享硬件资源上实现了高效的多租户，然而，全虚拟机的开销导致了通过容器实现操作系统级虚拟化的兴起。容器隔离进程并管理计算资源，同时避免了管理程序和虚拟机的开销[26]。

但是，从技术上来看容器和虚拟机是不同的，许多云平台虚拟机管理技术都要重新为容器设计。在这些技术中，一个关键的挑战是如何在不中断的情况下在主机之间迁移正在运行的容器，这也被称为容器热迁移。容器热迁移提供了动态重新定位容器的功能，同时维护有状态应用程序的可用性，这有利于负载平衡、硬件维护和故障恢复等重要用例[27]。尽管有这些好处，但热迁移涉及容器状态保存和无缝恢复方面的复杂技术挑战。

目前，工业界和学术界针对容器热迁移已提出了不同的方法。热迁移机制大概可分为 Pre-Copy 和 Post-Copy。当容器在源主机上运行时，Pre-Copy 方法迭代地将源端容器执行状态复制到目标主机，直到容器状态改变较少（例如最后一次迭代产生的脏数据较少），然后复制剩余状态数据，最后在目标主机上恢复容器执行。Pre-Copy 代表方案有 CloudHopper[28]。而 Post-Copy 则是将支持容器运行的最小状态子集传输到目标主机，然后在目标主机上运行容器，根据容器运行过程中产生的错误从源主机复制数据到目标主机。Post-Copy 的代表方案有 Voyager[29]。

目前 CRIU（Checkpoint/Restore In Userspace）[30]工具是大多数容器热迁移方案的基础，该工具在用户态实现了对进程的保存和恢复。目前，CRIU 已被整合到 Docker[31]，OpenVZ[32]等项目，提供容器热迁移能力。

但机密容器的热迁移与普通容器的迁移有所不同，其中涉及到机密容器普遍使用的 Virtio 设备的状态，包括网络连接和存储的有效管理，需要精确控制容器的运行状态，以确保迁移的平稳进行。同时，对于机密虚拟机的性能和安全性，也需要在迁移过程中加以考虑和保障。

现有的热迁移工具例如 CRIU 等仅支持对 Docker 这类直接运行于主机环境下的容器。机密容器运行于机密虚拟机内，并且普遍采用 Virtio 半虚拟化实现设备模拟。无法满足机密容器热迁移的实际应用需求。这是因为机密容器的内存状态经过加密，且主机平台不可信，导致现有热迁移工具无法直接应用。另一方面，尽管 AMD SEV 提供了对机密虚拟机整体迁移的支持，但在一个机密虚拟机中包含多个机密容器的情况下，由于机密容器的 Virtio 设备状态由机密虚拟机统一管理，现有的迁移工具尚不支持识别 Virtio 设备状态的来源，因此无法对某个或某些机密容器进行单独的热迁移。

3 系统架构设计

3.1 系统整体架构

3.1.1 机密容器基本架构

在虚拟化环境中，Docker 容器直接运行于主机环境下，共享操作系统内核。而机密容器不同，机密容器例如 Kata Container 在机密虚拟机内运行，并利用机密虚拟机作为每个容器或一组容器的隔离边界，提供了更高的安全性。

Virtio 标准是一种用于虚拟化环境中设备虚拟化的通用标准接口，旨在提高虚拟机与宿主机之间的通信效率和性能。该标准通过定义设备驱动程序和通信协议，为虚拟机与宿主机之间提供了一种高效、低成本的通信和设备访问方式。Virtio 标准在云计算、虚拟化服务器、虚拟桌面基础设施（VDI）、容器化技术等领域得到广泛应用。云服务提供商通常采用 Virtio 标准来优化虚拟机性能，提高资源利用率，并降低运行成本。Virtio 标准目前已支持内存、块设备、网络设备等。

机密容器通常使用 QEMU（Quick Emulator）提供的 Virtio 设备接口来发起具体的请求。QEMU 在启动机密容器时会开启 Virtio 选项为机密容器提供 Virtio 设备支持，具体表现为客户机端的前端设备驱动，存在于客户机内核内部，负责处理机密容器发起的请求。同时，在宿主机的 QEMU 进程中，Virtio 标准呈现为后端 Virtio 设备。该设备与客户机共享一块内存区域，用于存储处理具体 Virtio 请求的数据结构，即 Virtio Ring。这个后端设备由 QEMU 模拟，在获取具体请求后与宿主机中的真实设备进行交互。

在这个结构下，机密容器发起请求的基本传输处理流程包含多个步骤，如图 1 所示：机密容器通过③ 向前端虚拟设备驱动发起特定格式的请求，其中包含由具体请求组成的队列、包含设备信息的上下文信息等；④ 前端驱动在获得请求后进行解析、封装、提取，并按照 Virtio 标准将请求置入共享内存 Virtio Ring 中，再通过⑤ notify、kick 通知后端设备进行处理。QEMU 模拟的后端设备接收到具体请求后，由⑥ 宿主机中真实的请求处理程序进行处理。处理完成后，宿主机中的驱动通过一系列操作将返回的结果写回 Virtio Ring 中。随后，③ 和④ 过程中的客户机的驱动会从共享内存中获取返回结果并传递给具体容器。

然而，在多个机密容器共享一个 QEMU 模拟的 Virtio 设备的情况下，当对其中一个容器进行迁移时，

面临的挑战是：由于向 Virtio Ring 中写入的请求由客户机内核统一管理，机密容器通过写入寄存器中的地址来确定虚拟设备的请求和结果，现有容器迁移工具例如 CRIU 的迁移方案通常是整体迁移内存中的数据，对整个 Virtio Ring 的结构进行迁移，因此在目标平台中无法区分 Virtio Ring 中的请求来源。这导致迁移至目标平台后无法恢复容器的 Virtio 设备状态。

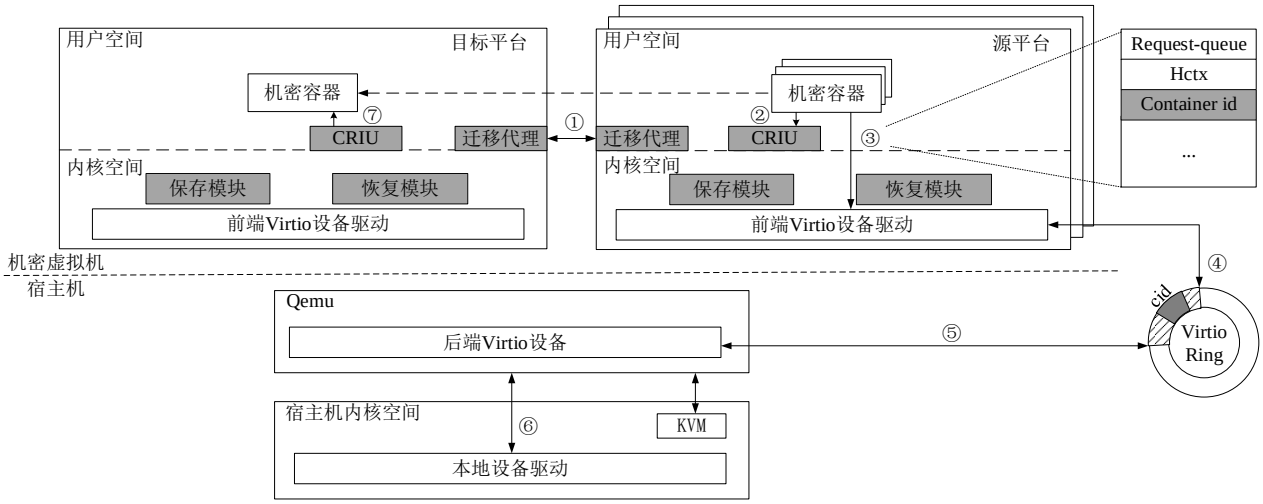

图 1 迁移方案整体架构

Figure1 Architecture of migration technology

3.1.2 迁移方案概述

在迁移过程中，针对多个机密容器在虚拟机平台上运行的情况，论文将机密容器所在的原始环境视为源平台，而待迁移至的对应目标机密虚拟机平台被定义为目标平台。

论文基于 CRIU 技术提出了一种在多机密容器使用 Virtio 设备的情况下进行迁移的方案，系统的架构设计如图 1 所示。

其中主要包括：

（1）在两个迁移平台中添加了迁移代理①，迁移代理基于 CRIU 热迁移工具，修改 cr_dump_tasks 和 cr_restore_tasks 模块，并将其与机密虚拟机远程认证结合，通过远程认证，建立基于共享密钥的安全传输通道。

（2）在迁移代理完成源平台和目标平台的密钥协商并建立安全传输通道后，在源平台通过② CRIU 收集被迁移机密容器的上下文信息，将其存储为镜像文件，通过迁移代理建立的安全传输通道在⑦ 目标平台进行恢复，完成机密容器内存状态的迁移。

（3）论文提出了增加用于识别具体请求来源的标识结构，以实现机密容器热迁移过程中更加细粒度的迁移。在机密容器发起请求时，将一个用于唯一识别该机密容器的标识 Cid（Container id）放入请求格式中。通过修改内核中传输请求的处理函数，例如 virtqueue_add_sgs，对该标识进行检查，并在③和④中额外申请 Virtio Ring 中的空间用于存放该标识，此额外申请的空间需要通过修改 Virtio 的基本数据结构 Available Ring，Used Ring，Descriptor Table 来实现。这个标识所占的空间开销与机密容器发出的请求个数以及 QEMU 初始化 VM 时设置的 Virtio Ring 大小有关。

（4）论文在机密虚拟机的内核中增加了两个用于迁移的内核模块，即保存模块和恢复模块。保存模块通过 CRIU 获取机密容器标识，从而在内核中检索 Virtio 数据结构中对应容器的请求信息，并将其保存为镜像。在恢复模块中，通过镜像中的虚拟设备状态信息调整 Virtio Ring 中的指针，将迁移的状态信息写入目标平台的 Virtio Ring 中。

此方案需要对 Virtio 标准进行调整以允许在 Virtio Ring 中引入标识。在迁移过程中，为每个机密容器维护一个唯一的标识符，确保请求和返回的

结果正确返回给发送请求的特定容器。这样的改进对于确保机密容器的安全性和正确性至关重要。

迁移方案的整体流程如图 2 所示，首先在源平台中迁移组件通过 CRIU 获取机密容器的 Cid，并通过迁移模块检测是否使用虚拟设备，之后便启动源平台状态保存进程，通过复制内存页和 Virtio Ring 中机密容器的相关信息，并将信息保存为镜像，通过安全传输通道发送至目标平台。在目标平台中根据镜像文件中的内容来恢复机密容器。

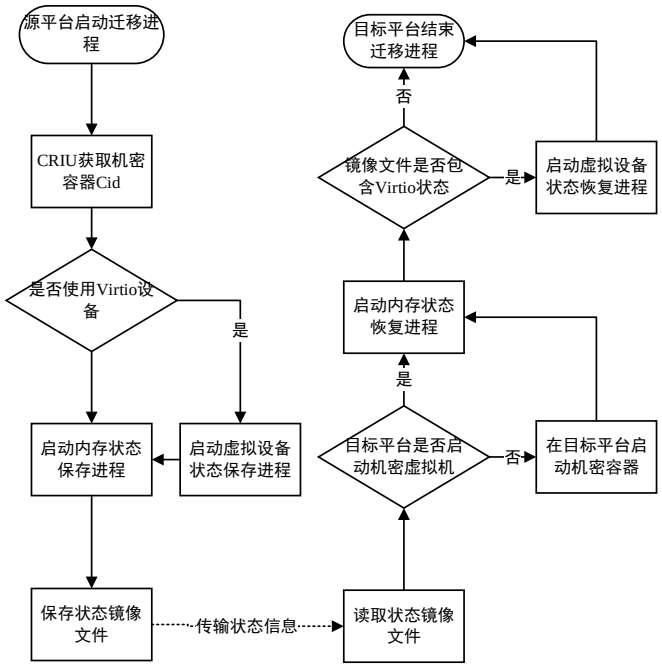

图 2 迁移方案整体流程

Figure2 Process of migration technology

3.1.3 迁移方案安全性分析

机密容器和机密虚拟机运行于云服务商所提供的云环境中，此环境下的服务提供商可以监视、控制环境中的行为，并且可以篡改、修改甚至提供恶意系统，并且可以篡改直接运行与云环境下的容器。论文提出的机密容器跨平台可信热迁移的工作中，用户对提供云环境的服务商不信任，认为除机密虚拟机外的其他软件和硬件环境都不可信，其中包括操作系统、hypervisor、内存和外围设备。对于机密容器而言，其旨在防止不受信任的容器工作负载或该容器工作负载的用户获得对主机基础设施的控制、获取信息或篡改主机基础设施。其依旧对主机平台上的设施不信任，包括宿主机操作系统、hypervisor 和外围设备等，但是机密容器信任客户机即机密虚拟机。因此论文所提方案中可信计算基只包含机密虚拟机和运行于机密虚拟机中的机密容器。

在实际迁移的过程中，迁移方案中状态保存过程的安全性由机密容器和机密虚拟机本身进行保障，而在状态信息进行传输的过程中，论文使用 SEV 进行远程认证并完成 DH 密钥协商，以此建立安全传输通道，其安全性由具体使用的安全传输协议进行保障。

3.2 虚拟设备状态迁移技术

论文提出在内核和 Virtio 标准中增加用于识别特定机密容器标识的虚拟设备状态迁移技术，以解决现有的迁移工具不支持识别 Virtio 设备状态的来源的问题。

Virtio Ring 是 Virtio 标准中的一个关键数据结构，用于实现虚拟机与宿主机之间的异步通信。它包括 Used Ring、Available Ring 和 Descriptor Table 三个部分。当请求到达时，Virtio 标准将请求分解后放入 Available Ring，并通知后端进行处理。请求结果则通过相同的数据结构放入 Used Ring，通知客户机已完成处理。

在虚拟环境中使用 Virtio 设备时，通常多个容器或进程共享一个或多个 Virtio 设备，提高资源利用效率。然而，这种共享方式在容器迁移时带来了挑战。现有解决方案通常采用整体迁移结构，增加了额外开销，无法实现对单个容器的细粒度迁移。

论文提出的机密容器跨平台可信热迁移技术针对这一问题，如图 3 所示通过在机密虚拟机中增加两个内核模块（保存模块和恢复模块）来更高效地保存和恢复机密容器 Virtio 设备的使用状态。这一技术可以提供更灵活的容器迁移方案，更好地满足细粒度迁移的需求。

算法 1：保存 Virtio 设备状态	
输入：机密容器标识 Cid 及内核结构体*g_vblk	
输出：Virtio Ring 结构中的请求 request	
1	blk_vring ← to_vring_virtqueue(g_vblk)
2	desc ← GetDesc(blk_vring)
3	avail ← GetAvail(blk_vring)
4	used ← GetUsed(blk_vring)
5	end_id ← Used Ring latest Id
6	j ← Available Ring latest Id previous Id
7	While(j % blk_vring->num ≠ avail->idx % blk_vring->num) do
8	If Cid = avail->ring[j % blk_vring->num].cid
9	While(True) do
10	Add desc.add to request
11	If desc.flags ≠ VRING_DESC_F_NEXT
12	Break

```
13        End While
14     End If
15          If avail->ring[j % blk_vring->num].id = end_id AND
avail->ring[(j - 1) % blk_vring->num].id ≠ end_id
16        Break
17     j ← j - 1
18  End While
19  Return request
```

保存模块如算法 1 所示，在迁移时通过 CRIU 获得用于识别机密容器的唯一标识 Cid 和全局符号表中的 *g_vlk*，模块首先根据该标识和内核结构体 *g_vblk* 确定 Used Ring 中最后处理的请求，其次如算法 1 的第 7 行所示对机密虚拟机的 Virtio Available Ring 进行逆序遍历。通过这种方式可以获取到最新添加入 Available Ring 当中，但是还没有添加到 Used Ring 中的 Descriptor Table 表项，这些表项表示客户机已经提交但还未被宿主机中 Qemu 处理完毕的请求，在对机密容器进行热迁移时需对这些请求进行保存和恢复。

算法 2：恢复 Virtio 设备状态

输入：机密容器标识 Cid 及内核结构体*g_vblk

```
1   vq ← to_vring_virtqueue(g_vblk)
2   For request in migration file do
3      rq ← kmalloc request buffer
4      rq ← request
5      sgs ← sg_init_table()
6      sgs ← sg_set_buf(rq)
7      adjust VRing's head
8      For n in rq's length do
9         addr ← vring_map_one_sg(vq, sgs[n], flag)
10        vring_add_desc(vq, addr, sgs[n])
11     Modify the Available Ring, Used Ring and Descriptor Table
```

恢复模块如算法 2 所示，在恢复时通过 CRIU 获得机密容器 Cid 和对应的状态信息，模块首先调整对应设备 Virtio Ring 的头部指针，其次根据机密虚拟机的 Virtio Available Ring 中的 *Idx* 查找到 Available Ring 中下一个可以写入的地址，并将迁移的状态信息构造为内核 Virtio 设备请求信息，通过系统调用，如算法第 6 行的 sg_set_buf 和算法第 10 行的 vring_add_desc 进行更新。同时同步 Used Ring 的状态和 Descriptor Table 的表项，确保具体请求存入 Request buffer 中。

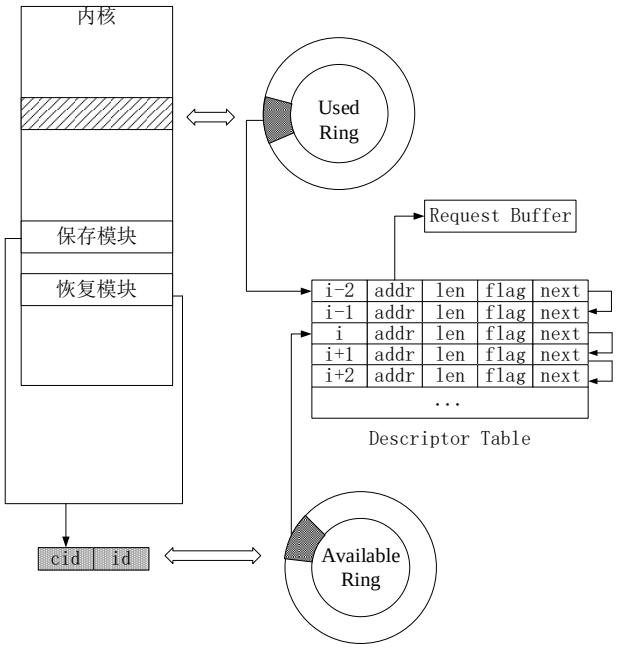

图 3 迁移系统模块

Figure3 Migration System Module

3.3 内存状态迁移技术

论文基于 CRIU 提出了一种机密容器内存状态迁移技术方案。

在原有的 CRIU 迁移方案中，CRIU 通过 ptrace 机制向待迁移的程序进程注入代码，实现对待迁移进程上下文信息的收集。主要步骤包括：在保存阶段，CRIU 依赖/proc 文件系统，通过/proc/pid/fd 和/proc/pid/fdinfo 收集文件描述信息，通过/proc/pid/maps 和/proc/pid/map_files 收集内存信息，并将这些信息以镜像的形式保存。在恢复阶段，CRIU 读取保存的镜像信息，通过一次或多次调用 fork()函数来重新创建进程，并根据镜像信息完成对 namespace 的准备、重新映射内存等操作。

论文提出的机密容器可信热迁移方案继承了 CRIU 对内存状态的迁移方案，并在此基础通过增加迁移代理的方式完成了对机密容器内存状态的热迁移。

在迁移开始时迁移代理首先进行源平台与目标平台间的密钥协商，完成源平台与目标平台的远程认证，并通过协商密钥建立安全传输通道。

论文所使用的 AMD SEV 可信执行环境提供了机密虚拟机远程认证的功能，远程认证允许源平台

与目标平台之间建立信任，并提供密码学证明，证明所迁移的机密容器是真实且未经篡改的。因此源平台保存的机密容器内存状态信息通过双方协商的会话密钥进行加密传输至目标平台，目标平台解密后，按照 CRIU 的内存状态恢复方案重新申请、映射机密容器的内存状态信息，完成对机密容器内存状态的恢复。

论文提出的机密容器内存状态迁移技术解决了机密容器迁移过程中源平台与目标平台之间由于机密虚拟机所使用密钥不同而导致的内存状态无法直接迁移的问题。

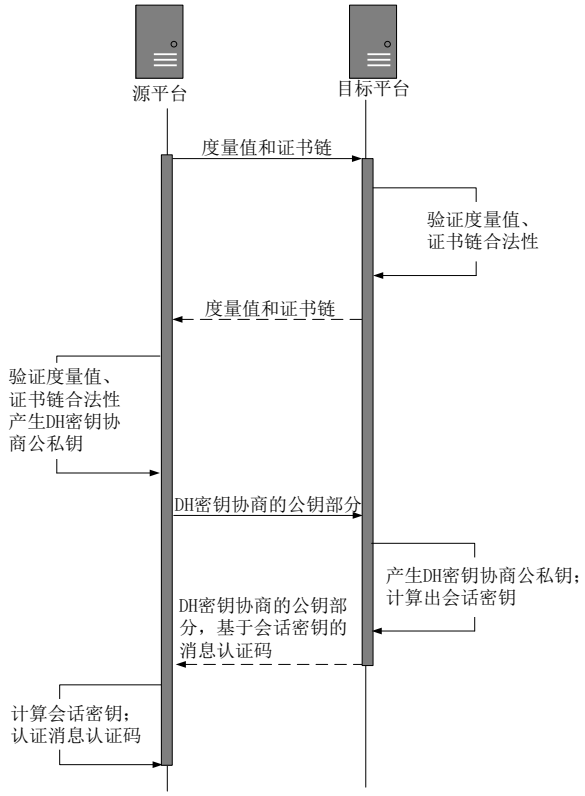

图 4 迁移平台密钥协商过程

Figure4 Migration platform key negotiation process

3.4 迁移流程

3.4.1 迁移平台密钥协商流程

论文提出的跨平台机密容器可信热迁移技术首先需要通过迁移代理在源平台和目标平台间进行密钥协商。在迁移平台密钥协商前，需要在两个平台中确保机密虚拟机启动，若源平台或目标平台尚未启动机密虚拟机，需要先启动机密虚拟机再进行机密容器的迁移流程。

在 AMD SEV 中，安全处理器（Security Processor, SP）是一个关键组件，负责管理 SEV 技术中的加密和安全功能。SP 的主要职责包括：

（1）密钥管理和加密引擎： SP 负责生成和管理虚拟机的加密密钥，并执行虚拟机内存的加密和解密操作。它包含加密引擎，用于加密虚拟机的内存页，确保内存中的数据对于不被信任的实体是不可见的。

（2）启动时的认证： SP 在虚拟机启动时执行远程认证过程，确保虚拟机的启动是可信的，防止虚拟机被恶意篡改。SP 通过与平台上的信任链进行通信，验证虚拟机的身份和完整性。

源平台和目标平台远程认证的主要过程如图 4 所示：

（1）虚拟机度量和证书链生成： 两个平台的机密虚拟机在启动时进行度量，生成自己的度量值，并生成相应的证书链。证书链包含机密虚拟机的身份信息、度量值和数字签名，用于后续的认证和密钥协商过程。

（2）远程认证请求： 源平台通过迁移代理发起远程认证请求，向目标平台发送自己的度量值和证书链。

（3）证书链验证和度量值比对： 目标平台迁移代理首先验证源平台的证书链的合法性和完整性。然后，目标平台迁移代理检查源平台的度量值是否与之前记录的度量值匹配，以确保源平台的完整性和可信度。

（4）密钥协商： 如果证书链验证和度量值比对都通过，则目标平台迁移代理向源平台发送自己的度量值和证书链。

（5）源平台验证目标平台的证书链的合法性和完整性： 源平台迁移代理检查目标平台的度量值是否与之前记录的度量值匹配，以确保目标平台的完整性和可信度。若验证通过，则发送自己的 DH 会话密钥协商的公钥部分。

（6）密钥解密和确认： 源平台接收到目标平台发送的加密的会话密钥公钥，使用自己的私钥得到会话密钥。源平台发送自己的 DH 会话密钥协商的公钥部分，并使用会话密钥加密一条确认消息，并将其发送至目标平台。

（7）密钥确认和安全通信： 目标平台迁移代

理接收到源平台发送的确认消息后，使用自己的私钥得到会话密钥，使用会话密钥进行解密，并验证确认消息的完整性和真实性。一旦目标平台确认会话密钥的有效性，源平台和目标平台可以使用该会话密钥进行安全的通信。

3.4.2 源平台状态保存流程

论文提出的机密容器跨平台可信热迁移技术中源平台状态保存主要包括内存状态保存和虚拟设备状态保存两个方面。

内存状态保存流程如图 5 所示，在执行该方案过程中，首先对两个机密虚拟机进行远程认证，并在认证成功后建立安全传输通道，这是确保通信安全性和可信度的关键环节。

在源平台和目标平台之间的远程认证过程中，使用了 AMD SP（Security Processor）进行认证，通过基于公钥的密钥交换协议创建安全通道。这个安全通道在迁移过程中确保后续数据传输的隐私保护和完整性。

论文基于 CRIU 技术，通过 /proc/$pid/smaps 解析 VMAs 中的映射信息；从/proc/$pid/map_files 中读取映射文件；从/proc/$pid/fd 读取文件描述符；利用 ptrace 接口并解析/proc/$pid/stat 文件来保存任务的核心参数等方式收集被迁移机密容器的内存状态信息，并根据功能进行分类，将其存储为不同的镜像文件。

为确保在迁移机密容器状态信息的过程中的安全性，通过远程认证建立的安全传输通道，将之前生成的镜像文件传输至目标平台。

虚拟设备状态保存流程如图 6 所示，首先，源平台使用 CRIU 启动迁移进程，利用 CRIU 向内核中的保存模块传递机密容器的标识 Cid。保存模块通过算法 1 使用机密容器的标识 Cid 读取虚拟设备执行过程中 Virtio Ring 内的信息，从而获取机密容器对 Virtio 设备的请求和返回状态，并将其保存为镜像文件。通过迁移代理建立安全传输通道，将保存有虚拟设备状态和内存状态的镜像文件发送至目标平台。

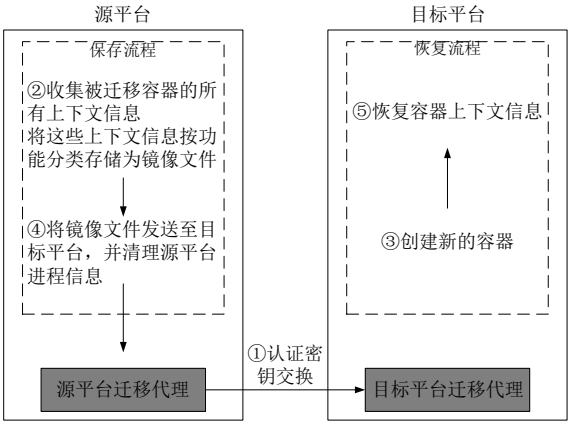

图 5 内存状态迁移流程

Figure5 Memory state migration process

3.4.3 目标平台状态恢复流程

目标平台状态恢复主要包括内存状态恢复和虚拟设备状态恢复两个方面。

内存状态恢复流程如图 5 所示，在目标平台接收到镜像文件后，系统将使用 CRIU 技术解析这些文件，恢复出备份前的状态，使得被迁移的机密容器能够从备份前的状态继续执行。

虚拟设备状态恢复如图 6 所示，在目标平台中，利用 CRIU 将源平台的镜像文件传递至内核中的恢复模块。恢复模块通过构造 Virtio 设备的请求信息，从而将镜像文件中保存的状态信息依据机密容器的标识 Cid 插入目标平台的 Virtio Ring 中，进而在目标平台中恢复虚拟设备的使用状态。迁移完成后，通过 Virtio Ring 中的 Cid 识别具体机密容器的请求，为其分配所需的 Virtio 设备。

在所有迁移工作完成后通过 CRIU 对源平台的机密容器进行销毁。

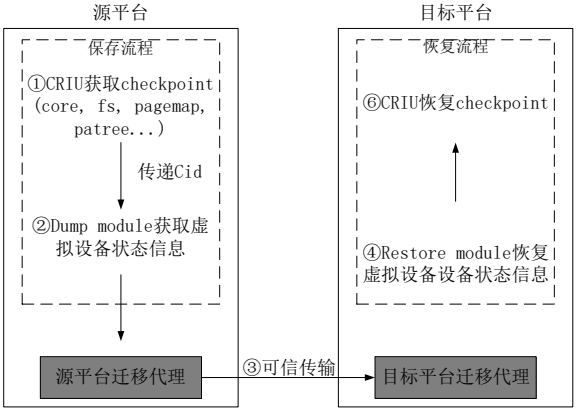

图 6 虚拟设备状态迁移流程

Figure6 Virtio device state migration process

4 评估

4.1 设置

在实际应用中，MySQL 数据库作为最受欢迎的数据库之一，需要在不同的服务节点进行迁移。因此，在论文的评估中使用了一台具备 AMD SEV 可信执行环境的服务器作为迁移的物理机，来模拟实际环境下的物理机器。该机器配置有 AMD EPYC 7543 32-Core CPU、64GB 内存，操作系统为 Ubuntu 22.04，内核版本为 Linux 6.5.0。论文使用了两台机密虚拟机作为迁移的源平台和目标平台，每台机器都使用了与宿主机相同的 4 核 CPU 和 4GB 内存。两台机器的操作系统为 Ubuntu 22.04，并使用基于 Linux 5.19 版本的内核作为迁移系统。两台机密虚拟机使用 Virtio-blk 虚拟设备作为具体迁移过程中的实验目标，使用 Virtio-blk 构建两台迁移机器的根文件系统，每个 Virtio-blk 的大小为 20GB。

由于目前还没有针对 Virtio 虚拟化设备的标准基准测试，论文模拟了相同数据量的 MySQL 进行读写，并将缓冲区中的数据同步到硬盘的过程，捕获了迁移所需的信息。对于整体迁移方案的性能测试，论文使用 Sysbench 来改变负载的大小，对基本的文件读写和模拟硬件性能进行了测试。在论文的评估测试实验中，机密容器标识在系统中传输和 Qemu 对应数据结构的改变带来了一部分的性能开销，但对整体性能而言并不明显，可以忽略。

4.2 迁移方案性能

迁移方案在正常运行时需要评估的性能指标包括 Virtio-blk 进行读写的吞吐量以及相应的延迟。论文使用了 Sysbench 作为评估测试的工具，对使用了迁移标识的 Virtio-blk 设备进行基本的访问性能测试和读写过程中的延迟测试。Virtio-blk 设备在通过 Qemu 模拟的机密虚拟机中挂载。论文测试了需要读写的文件大小从 32MB 到 1024MB 的区间，评估内容为读写过程中的吞吐量。在每种配置中进行 5 次测试，并计算其平均值。如图 7 所示，随着数据量的增加，吞吐量的开销也增加，当文件大小为 1GB 时，吞吐量的开销趋于稳定。迁移方案的延迟会随着数据量的增大而增加并与现实正常系统的开销趋于一致。

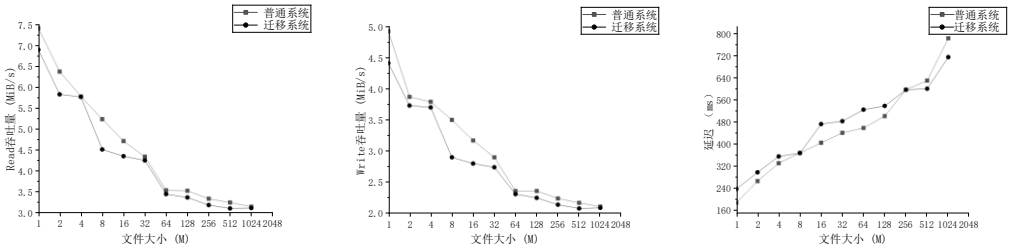

图 7 迁移方案性能

Figure7 Performance of migration technology

4.3 停机时间

通过模拟数据库将读写内容同步到硬盘的过程，论文测量了在同步过程中进行迁移所需的停机时间。停机时间被划分为保存时间和恢复时间。论文将迁移方案中的保存和恢复功能集成到通用的迁移工具 CRIU 中，测试方式是在模拟数据库在正常系统下读写所需时间的 1/2 时进行迁移。如表 1 所示，论文计算了迁移所需的停机时间占总读写时间的比重。由于进行机密容器保存的过程在迁移方案中只需要使用保存模块遍历固定数量的 Virtio Ring，因此所需时间在 125ms 左右。而恢复模块则需要在 Virtio Ring 中寻找暂未被使用的区域进行插入，因此当所需插入的请求数超过 Virtio Ring 的固定大小时，需要较多时间。如表 1 中迁移的停机时间占程序运行总时间的比重可知，随着数据量的增加，数据库读写所需的时间远超迁移方案停机时间。因此，在面向云计算等吞吐量较大的服务迁移过程中，论文提出的机密容器跨平台可信热迁移方案表现出优越的性能。

表 1 停机时间

Table1 Downtime

模拟数据库读写大小（table * table size)	迁移方案时间（ms）		百分比
	保存时间	恢复时间	
5*10	93	45	3.28%
10*10	110	88	3.09%
5*100	95	73	5.07%
10*100	99	80	3.17%
5*1000	125	82	3.55%
10*1000	103	84	2.13%
5*10000	104	68	1.30%
10*10000	135	136	1.24%
5*100000	100	134	0.32%
10*100000	138	136	0.14%

迁移方案基于 CRIU，弥补了在面向多机密容器环境下细粒度迁移方面的不足。如图 8 所示，将迁移方案集成到 CRIU 时会存在一定的延迟。在面对数据量较小的迁移时，CRIU 与迁移方案所需的时间开销相差不大。然而，迁移方案的延迟在大数据量与现实正常系统的开销和较于小数据量误差更大，当 MySQL 读写数据量增长到大于 1GB 时，二者的开销差距会增大。尽管如此，根据评估结果显示，此开销在总执行时间中的比重可以忽略不计。因此，在合理的范围内，仍然能够满足热迁移的需求。

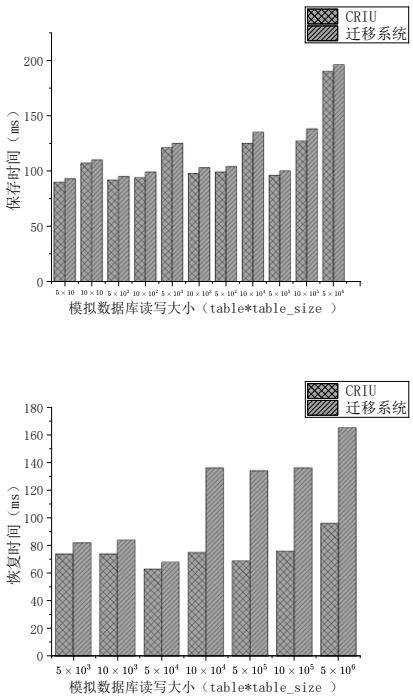

图 8 迁移方案与 CRIU 的比较

Figure8 Comparison between migration technology and CRIU

5 总结

论文首次提出了一种机密容器跨平台热迁移方案，并详细描述了该方案的实现方法，包括机密虚拟机中设备驱动的标识和传输流程。论文的研究内容包括机密容器进程内存状态的保存与恢复，以及机密容器虚拟设备状态的保存与恢复。解决了目前现有容器迁移不支持机密容器的问题，并在面向多容器共用 Virtio 设备的情况下完成了更加细粒度的热迁移。

论文首次提出了一种机密容器的跨平台迁移方法。首先，在源平台和目标平台之间设立迁移代理，通过远程认证在源平台和目标平台之间建立安全通道。其次由迁移代理完成容器状态的迁移，容器状态包括内存状态和虚拟设备状态，分别将源平台的进程状态和虚拟设备状态保存为镜像文件，并在目标平台进行恢复。在虚拟设备状态的保存与恢复过程当中，通过对虚拟设备增加容器 Cid 作为标识符，完成了对特定机密容器的虚拟设备状态的识别。

实验评估证实，论文提出的机密容器跨平台可信热迁移方案符合大规模机密容器环境中对热迁移的需求，即在迁移的停机时间上可以达到可以忽略的程度。虽然在恢复时间上与现有的普通容器热迁移方案有着一定的差距，但是面对工业界所使用的大型容器服务而言，仍在可接受范围。综上所述，论文提出机密容器可信热迁移方案在实际多机密容器运行时可以完成对单个容器及其 Virtio 请求进行细粒度迁移，具有实际意义。

参考文献

[1] ARM L. Arm security technology-building a secure system using trustzone technology[R]. *PRD-GENC-C. ARM Ltd.* Apr.(cit. on p.), 2009.

[2] Bahmani R, Brasser F, Dessouky G, et al. CURE: A Security Architecture with CUstomizable and Resilient Enclaves[C] *USENIX Security Symposium.* 2021: 1073-1090.

[3] Brasser F, Gens D, Jauernig P, et al. SANCTUARY: ARMing TrustZone with User-space Enclaves[C] *NDSS.* 2019.

[4] Costan V, Lebedev I A, Devadas S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation[C] *USENIX Security Symposium.*

2016: 857-874.

[5] Intel I. Software guard extensions programming reference, revision 2[J]. 2014.

[6] Kaplan D, Powell J, Woller T. AMD memory encryption[J]. White paper, 2016.

[7] Lee D, Kohlbrenner D, Shinde S, et al. Keystone: An open framework for architecting trusted execution environments[C] *Proceedings of the Fifteenth European Conference on Computer Systems.* 2020: 1-16.

[8] Ge X, Kuo H C, Cui W. Hecate: Lifting and Shifting On-Premises Workloads to an Untrusted Cloud[C] *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security.* 2022: 1231-1242.

[9] Sardar M U, Musaev S, Fetzer C. Demystifying attestation in intel trust domain extensions via formal verification[J]. *IEEE access, 2021*, 9: 83067-83079.

[10] Devices A M. AMD64 architecture programmer's manual volume 2: System programming[J]. 2006, 2006.

[11] Porter N, Golanand G, Lugani S. Introducing Google Cloud Confidential Computing with Confidential VMs[J]. https://cloud. google. com/blog/products/identitysecurity/introducing-google-cloud-confidential-computing-with-confidentialvms/(visited on Dec. 25, 2020)(cit. on p. 42), 2020.

[12] Russinovich M. Azure and AMD announce landmark in confidential computing evolution[J]. *Retrieved September*, 2021, 13: 2021.

[13] Buhren R, Gueron S, Nordholz J, et al. Fault attacks on encrypted general purpose compute platforms[C]*Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 2017: 197-204.

[14] Du Z H, Ying Z, Ma Z, et al. Secure encrypted virtualization is unsecure[EB/OL]. ArXiv preprint ArXiv:1712.05090, 2017.

[15] Wilke L, Wichelmann J, Morbitzer M, et al. Sevurity: No security without integrity: Breaking integrity-free memory encryption with minimal assumptions[C]*2020 IEEE Symposium on Security and Privacy (SP). IEEE,* 2020: 1483-1496.

[16] Hetzelt F, Buhren R. Security analysis of encrypted virtual machines[J]. *ACM SIGPLAN Notices,* 2017, 52(7): 129-142.

[17] Morbitzer M, Huber M, Horsch J. Extracting secrets from encrypted virtual machines[C] *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy.* 2019: 221-230.

[18] Morbitzer M, Huber M, Horsch J, et al. Severed: Subverting amd's virtual machine encryption[C] *Proceedings of the 11th European Workshop on Systems Security.* 2018: 1-6.

[19] Li M, Zhang Y. Exploiting unprotected i/o operations in amd's secure encrypted virtualization[C] *Proceedings of the 2019 USENIX Security Symposium*. 2019.

[20] Li M, Zhang Y, Lin Z. Crossline: Breaking" security-by-crash" based memory isolation in amd sev[C] *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.* 2021: 2937-2950.

[21] Sev-Snp AMD. Strengthening VM isolation with integrity protection and more[J]. White Paper, January, 2020: 8.

[22] Kata Containers The speed of containers, the security of VMs. https://katacontainers.io/collateral/kata-containers-1pager.pdf

[23] Arnautov S, Trach B, Gregor F, et al. {SCONE}: Secure linux containers with intel {SGX}[C] *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016: 689-703.

[24] Li M, Wilke L, Wichelmann J, et al. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP[C] *2022 IEEE Symposium on Security and Privacy (SP).* IEEE, 2022: 337-351.

[25] Kaplan D. Protecting VM register state with SEV-ES[J]. White paper, 2017.

[26] Merkel D. Docker: lightweight linux containers for consistent development and deployment[J]. *Linux j*, 2014, 239(2): 2.

[27] Bhardwaj A, Krishna C R. Virtualization in cloud computing: Moving from hypervisor to containerization—a survey[J]. *Arabian Journal for Science and Engineering,* 2021, 46(9): 8585-8601.

[28] Jain S. Cloud Hopper: A Unified Cloud Solution to Manage Heterogeneous Clouds[D]. , 2016.

[29] Nadgowda S, Suneja S, Bila N, et al. Voyager: Complete container state migration[C] *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE*, 2017: 2137-2142.

[30] Pickartz S, Eiling N, Lankes S, et al. Migrating LinuX containers using CRIU[C] *High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P^ 3MA, VHPC, WOPSSS, Frankfurt, Germany*, June 19–23, 2016, Revised Selected Papers 31. SpRinger International Publishing, 2016: 674-684.

[31] Use containers to Build, Share and Run your applications. https://.docker.com/resources/what-container/

[32] OpenVZ: a container-based virtualization for Linux. http://wiki.openvz.org/Main_Page