2023

OpenAnolis
龙 蜥 社 区

云原生机密计算最佳实践白皮书

OpenAnolis
龙 蜥 社 区

目录 CONTENTS

01 OpenAnolis 龙蜥社区 01
OpenAnolis Community

02 机密计算简介与现状 04
Introduction And Status Of Confidential Computing

03 云原生机密计算SIG概述 07
Overview Of The Cloud Native Confidential Computing SIG

04 机密计算平台 11
Confidential Computing Platform

海光CSV：海光安全虚拟化技术 12

Intel SGX: Intel安全防护扩展 14

Intel TDX: Intel安全虚拟化技术 16

AMD SEV: AMD安全加密虚拟化技术 18

ARM CCA: Arm安全加密虚拟化技术 20

05 编程框架 22
Programming Framework

Intel SGX SDK/PSW/DCAP: Intel SGX 软件开发套件和平台软件服务 23

Intel Homomorphic Encryption: Intel 平台同态加密加速框架 29

- Intel_HE_Toolkit 开发指南 31

Apache Teaclave Java TEE SDK: 面向Java生态的机密计算编程框架 35
- Apache_Teaclave_Java_TEE_SDK 最佳实践 37

RATS-TLS: 跨机密计算平台的双向传输层安全协议 43

06 运行时底座 Runtime Foundation 46

海光CSV机密容器 47
- 基于runtime-attestation使用机密容器 47
- 基于pre-attestation使用机密容器 56
- 基于runtime-attestation使用签名容器 61

海光CSV机密虚拟机 68

Intel TDX机密容器 72

Intel vSGX：Intel SGX虚拟化 81
- Intel SGX虚拟机最佳实践 82

AMD SEV机密容器 91

AMD SEV机密虚拟机 102

Occlum: 基于Intel SGX的轻量级LibOS 107

Inclavare Containers: 面向机密计算场景的开源容器运行时技术栈 109

Enclave-CC: 进程级机密容器 111

07 解决方案 Solution 113

Intel Confidential Computing Zoo: Intel机密计算开源解决方案 114
- 部署TensorFlow Serving在线推理服务 115
- 部署TensorFlow横向联邦学习 121
- 部署隐私集合求交方案 126

PPML: 端到端隐私保护机器学习解决方案 133

OpenAnolis
龙 蜥 社 区

认识龙蜥

龙蜥社区（ OpenAnolis ）成立于 2020 年 9 月，由阿里云、ARM、统信软件、龙芯、飞腾、中科方德、Intel 等 24 家国内外头部企业共同成立龙蜥社区理事会，到目前有超过 300 家合作伙伴参与共建，是是国内领先的开源社区之一，具备较为领先的产业和技术影响力。目前，龙蜥操作系统下载量已超240万，整体装机量达300多万，100余款企业产品完成与龙蜥操作系统的适配。同时，统信软件、中科方德、中国移动云、麒麟软件、中标软件、凝思软件、浪潮信息、新支点、阿里云基于龙蜥开源操作系统推出各自商业版本及产品，在政务、金融、交通、通信等领域累计服务用户超过30万。

龙蜥开源影响力

龙蜥社区及龙蜥操作系统也获得了一定的行业认可，工信部电子标准院首批开源项目成熟度评估，成为唯一获得“卓越级”认证的开源项目、龙蜥社区荣登 2022 科创中国“开源创新榜”、荣获“中国开源云联盟年度优秀开源项目奖”、“OSCAR 开源尖峰案例奖”等 25 项行业奖项。

龙蜥项目运作模式

龙蜥社区已成立 50+ 个 SIG 工作组，围绕芯片、内核、编译器、安全、虚拟化及云原生等操作系统核心领域进行技术创新，已发布龙蜥 Anolis OS 7、Anolis OS 8.x 系列、Anolis OS 23 公测版、Lifsea OS 等多个社区版本，为应对即将停服的 CentOS，官网已上线「CentOS 停服专区」为用户提供迁移方案及长期稳定支持，致力于成为 CentOS 的更佳替代。

龙蜥运营管理

“为更好地运营和治理社区，龙蜥社区定期召开月度运营委员会会议、技术委员会会议，理事大会。

关于理事大会：龙蜥社区第二届理事大会圆满召开！理事换届选举、4 位特约顾问加入

关于运营委员会会议：龙蜥社区第15次运营委员会会议顺利召开

欢迎更多企业加入共建， 龙腾计划可参看：“龙腾计划”启动！邀请 500 家企业加入，与龙蜥社区一起拥抱无限生态。”

龙蜥开放的生态

为了鼓励合作伙伴在社区探索出更多的商业合作方式，真正牵引企业在龙蜥社区的合作落地，社区推出「龙腾计划」的升级版——「生态发展计划」，更聚焦在产品和商业合作本身。

详情可参看：*https://openanolis.cn/page/partner2*

关于龙蜥操作系统 （ Anolis OS ）

龙蜥操作系统（Anolis OS）搭载了 ANCK 版本的内核，性能和稳定性经过历年“双 11”历练，能为云上典型用户场景带来 40% 的综合性能提升，故障率降低 50%，兼容 CentOS 生态，提供平滑的 CentOS 迁移方案，并提供全栈国密能力。最新的长期支持版本 Anolis OS 8.6 已发布，更多龙蜥自研，支持 X86_64 、RISC-V、Arm64、LoongArch 架构，完善适配 Intel、飞腾、海光、兆芯、鲲鹏、龙芯等主流芯片。

下载体验链接： *https://openanolis.cn/download*

2021 年 12 月 31 日，龙蜥开源社区（OpenAnolis）上线「CentOS 停服专区」，为受 CentOS 停服影响的用户提供迁移方案及长期稳定支持。此次停服，龙蜥操作系统（Anolis OS）产品优势包括：打造系统化解决方案 AOMS、提供多款配套工具、承诺10年技术支持、兼容 CentOS 生态、具备差异化核心技术优势、历经丰富场景验证、沉淀用户迁移案例实践。

反馈与共创

OpenAnolis是一个开放包容的社区，因此我们也欢迎志同道合之士参与我们的文档修订。

对于文档中您认为不足之处，欢迎到我们的官方仓库 Whitebook ConfidentialComputing 新开issue，我们会第一时间进行响应。

另外，若您想更新文档，也同样欢迎在 Whitebook ConfidentialComputing 提PR

机密计算简介与现状

Introduction And Status Of Confidential Computing

机密计算简介与现状

数据安全与机密计算

数据在整个生命周期有三种状态：At-Rest（静态）、In-Transit（传输中）和 In-Use（使用中）。

• At-Rest 状态下，一般会把数据存放在硬盘、闪存或其他的存储设备中。保护 At-Rest 状态的数据有很多方法，比如对文件加密后再存放或者对存储设备加密。

• In-Transit 是指通过公网或私网把数据从一个地方传输到其他地方，用户可以在传输之前对文件加密或者采用安全的传输协议保证数据在传输中的安全，比如HTTPS、SSL、TLS、FTPS 等。

• In-Use 是指正在使用的数据。即便数据在传输过程中是被加密的，但只有把数据解密后才能进行计算和使用。也就意味着，如果数据在使用时没有被保护的话，仍然有数据泄露和被篡改的风险。

在这个世界上，我们不断地存储、使用和共享各种敏感数据：从信用卡数据到病历，从防火墙配置到地理位置数据。保护处于所有状态中的敏感数据比以往任何时候都更为重要。如今被广泛使用的加密技术可以用来提供数据机密性（防止未经授权的访问）和数据完整性（防止或检测未经授权的修改），但目前这些技术主要被用于保护传输中和静止状态的数据，目前对数据的第三个状态“使用中”提供安全防护的技术仍旧属于新的前沿领域。

机密计算指使用基于硬件的可信执行环境（Trusted Execution Environment，TEE）对使用中的数据提供保护。 通过使用机密计算，我们现在能够针对“使用中”的数据提供保护。

机密计算的核心功能有：

• 保护 In-Use 数据的机密性。未经授权的实体（主机上的应用程序、主机操作系统和Hypervisor、系统管理员或对硬件具有物理访问权限的任何其他人。）无法查看在TEE中使用的数据，内存中的数据是被加密的，即便被攻击者窃取到内存数据也不会泄露数据。

• 保护 In-Use 数据的完整性。防止未经授权的实体篡改正在处理中的数据，度量值保证了数据和代码的完整性，使用中有任何数据或代码的改动都会引起度量值的变化。

• 可证明性。通常 TEE 可以提供其起源和当前状态的证据或度量值，以便让另一方进行验证，并决定是否信任 TEE 中运行的代码。最重要的是，此类证据是由硬件签名，并且制造商能够提供证明，因此验证证据的一方就可以在一定程度上保证证据是可靠的，而不是由恶意软件或其他未经授权的实体生成的。

机密计算的现状与困境

业界内的诸多厂商就已经开始关注并投入到机密计算中。各大芯片厂家和云服务提供商（Cloud Service Provider，简称 CSP）都在机密计算领域投入研发资源，并组建了“机密计算联盟”。该联盟专门针对云服务及硬件生态，致力于保护计算时的数据安全。

目前机密计算正处于百花齐发和百家争鸣的阶段，市场和商业化潜力非常巨大。 但机密计算在云原生场景中还有一些不足：

1、用户心智不足。用户普遍对机密计算这项新技术的认知感不足，难以将其与自己的业务直接联系起来，导致需求不够旺盛。

2、技术门槛高。目前，相比传统开发方式，主流的机密计算技术的编程模型给人们对机密计算技术的印象是学习和使用门槛高，用户需要使用机密计算技术对业务进行改造，令很多开发者望而生畏。

3、应用场景缺乏普适性。目前，机密计算主要被应用于具有特定行业壁垒或行业特征的场景，如隐私计算和金融等。这些复杂场景让普通用户很难触达机密计算技术，也难以为普通用户打造典型应用场景。同时，不

同厂商的CPU TEE虽各自具有自身的特点，但都无法解决异构计算算力不足的问题，限制了机密计算的应用领域。

4、信任根和信任模型问题。在信创、数据安全和安全合规等政策性要求对CPU TEE的信任根存在自主可控的诉求；与此同时，虽然有部分用户愿意信任云厂商和第三方提供的解决方案，但多数用户对云厂商和第三方不完全信任，要求将机密计算技术方案从租户TCB中完全移除。

总之，目前已有的机密计算技术方案存在以上困境，不能够完全满足用户不同场景的安全需求。为了解决以上四个问题，云原生机密计算SIG应运而生，主要可概括为四点：

1、推广机密计算技术。

• 邀请参与方在龙蜥大讲堂介绍和推广机密计算技术与解决方案。

• 与芯片厂商合作，未来可以通过龙蜥实验室让外部用户体验机密计算技术，对机密计算有一个更深入化的了解。

2、提高机密计算技术的可用性。

• 支持多种机密计算硬件。

• 提供多种运行时底座和编程框架供用户选择。

3、提升机密计算技术的泛用性

• 为最有代表性的通用计算场景打造解决方案和案例（特性即产品）。

• 积极拥抱并参与到机密计算前沿技术领域的探索与实践，加速创新技术的落地。

4、澄清误会并增加用户信心

• 发布机密计算技术白皮书。

• 与社区和业界合作，未来提供结合了软件供应链安全的远程证明服务体系。

OpenAnolis
龙 蜥 社 区

云原生机密计算SIG概述

Overview Of The Cloud Native Confidential Computing SIG

云原生机密计算SIG概述

随着通信、网络和计算机技术的持续演进与广泛应用，数据资源的开放共享、交换流通成为推动“万物互联、智慧互通”的重要趋势。与此同时，近年来数据安全事件频发、数据安全威胁日趋严峻，数据的安全处理和流通受到了国内外监管部门的广泛重视。如何在保障安全的前提下最大程度发挥数据的价值，是当前面临的重要课题。

在日益严苛的隐私保护相关法律法规约束下，作为当前数据处理基础设施的云计算也正在经历一次重大的范式转换，即从默认以 CSP 为信任基础的计算范式走向信任链与 CSP 解耦的新范式。我们将此范式称为隐私保护云计算，而机密计算是实现隐私保护云计算的必由之路。

为拥抱隐私保护云计算新范式，促进隐私保护云计算生态发展，云原生机密计算SIG应运而生：

解决方案	Intel CCZoo (TensorFlow横向联邦学习、在线推理服务、隐私集合求交)
运行时	Gramine / 海光CSV机密容器 / 海光CSV机密虚拟机 / Inclavare Containers / KubeTEE Enclave Services / Occlum / SEV机密容器 / SEV机密虚拟机 / SGX虚拟化 / TDX机密容器 & 机密虚拟机
编程框架	Apache Teaclave Java TEE SDK / Intel HE Toolkit / Intel SGX & PSW & DCAP
OS适配	Anolis 8 + ANCK 5.10
硬件支持	AMD SEV(-ES)/ 海光 CSV 1+2 / Intel SGX 2.0 / Intel TDX 1.0

愿景

云原生机密计算SIG致力于通过开源社区合作共建的方式，为业界提供开源和标准化的机密计算技术以及安全架构，推动云原生场景下机密计算技术的发展。工作组将围绕下述核心项目构建云原生机密计算开源技术栈，降低机密计算的使用门槛，简化机密计算在云上的部署和应用步骤，拓展使用场景及方案。

云原生机密计算SIG的愿景是：

1）构建安全、易用的机密计算技术栈

2）适配各种常见机密计算硬件平台

3）打造典型机密计算产品和应用案例

项目介绍

海光 CSV 机密容器

CSV 是海光研发的安全虚拟化技术。CSV1 实现了虚拟机内存加密能力，CSV2 增加了虚拟机状态加密机制，CSV3 进一步提供了虚拟机内存隔离支持。CSV 机密容器能够为用户提供虚拟机内存加密和虚拟机状态加密能力，主机无法解密获取虚拟机的加密内存和加密状态信息。CSV 虚拟机使用隔离的 TLB、Cache 等硬件资源，支持安全启动、代码验证、远程认证等功能。

- 主页：*https://openanolis.cn/sig/coco/doc/533508829133259244*

Intel Confidential Computing Zoo

Intel 发起并开源了 Confidential Computing Zoo (CCZoo)，CCZoo 基于 Intel TEE（SGX,TDX）技术，提供了不同场景下各种典型端到端安全解决方案的参考案例，增加用户在机密计算方案实现上的开发体验，并引导用户结合参考案例快速设计自己特定的机密计算解决方案。 CCZoo 目前提供了基于 Libos + Intel TEE + OpenAnolis 容器的 E2E 安全解决方案参考案例，后续，CCZoo 计划基于 OpenAnolis ，提供更多的机密计算参考案例，为用户提供相应的容器镜像，实现敏捷部署。

- 主页：*https://cczoo.readthedocs.io*
- 代码库：*https://github.com/intel/confidential-computing-zoo*

Intel HE Toolkit

Intel HE Toolkit 旨在为社区和行业提供一个用于实验、开发和部署同态加密应用的平台。目前 Intel HE Toolkit 包括了主流的 Leveled HE 库，如 SEAL、Palisade和 HELib，基于使能了英特尔最新指令集加速的的 Intel HEXL 库，在英特尔至强处理器平台上为同态加密业务负载提供了卓越的性能体验。同时，Intel HE Toolkit即将集成半同态 Paillier 加速库 IPCL，为半同态加密应用提供加速支持。此外，Intel HE Toolkit 还提供了示例内核、示例程序和基准测试 HEBench。这些示例程序演示了利用主流的同态加密库构建各种同态加密应用保护用户隐私数据的能力。HEBench 则为各类第三方同态加密应用提供了公允的评价基准，促进了同态加密领域的研究与创新。

- 主页：*https://www.intel.com/content/www/us/en/developer/tools/homoorphic-encryption/*
- 代码库：Intel HE Toolkit: *https://github.com/intel/he-toolkit*

Intel HEXL: https:*//github.com/intel/hexl*

Intel Paillier Cryptosystem Library (IPCL): *https://github.com/intel/pailliercryptolib*

HE Bench: *https://github.com/hebench*

Intel SGX Platform Software and Datacenter Attestation Primitives

在龙蜥生态中为数据中心和云计算平台提供 Intel SGX 技术所需的平台软件服务，如远程证明等。

- RPM包：*https://download.01.org/intel-sgx/latest/linux-latest/distro/Anolis86/*
- 代码库：*https://github.com/intel/SGXDataCenterAttestationPrimitives*

Intel SGX SDK

在龙蜥生态中为开发者提供使用 Intel SGX 技术所需的软件开发套件，帮助开发者高效便捷地开发机密计算程序和解决方案。

- RPM包：*https://download.01.org/intel-sgx/latest/linux-latest/distro/Anolis86/*
- 代码库：*https://github.com/intel/linux-sgx*

Occlum

Occlum 是一个 TEE LibOS，是机密计算联盟（CCC, Confidential Computing Consortium）的官方开源项目。目前 Occlum 支持 Intel SGX 和 HyperEnclave 两种 TEE。Occlum 在 TEE 环境中提供了一个兼容 Linux 的运行环境，使得 Linux 下的应用可以不经修改就在 TEE 环境中运行。Occlum 在设计时将安全性作为最重要的设计指标，在提升用户开发效率的同时保证了应用的安全性。Occlum 极大地降低了程序员开发 TEE 安全应用的难度，提升了开发效率。

- 主页：*https://occlum.io/*
- 代码库：*https://github.com/occlum/occlum*

提供 TEE 有关的 Kubernetes 基础服务 (如集群规模的密钥分发和同步服务、集群远程证明服务等），使得用户可以方便地将集群中多台 TEE 机器当作一个更强大的 TEE 来使用。

• 代码库：*https://github.com/SOFAEnclave/KubeTEE*

Apache Teaclave Java TEE SDK

Apache Teaclave Java TEE SDK(JavaEnclave)是一个面向 Java 生态的机密计算编程框架，它继承Intel SGX SDK所定义的Host-Enclave机密计算分割编程模型。JavaEnclave提供一种十分优雅的模式，对一个完整的Java应用程序进行分割与组织。它将一个Java项目划分成三个子模块，Common子模块定义SPI服务接口，Enclave子模块实现SPI接口并以Provider方式提供服务，Host子模块负责TEE环境的管理和Enclave机密服务的调用。整个机密计算应用的开发与使用模式符合Java经典的SPI设计模式，极大降低了Java机密计算开发门槛。此外，本框架创新性应用Java静态编译技术，将Enclave子模块Java代码编译成Native形态并运行在TEE环境，极大减小了Enclave攻击面，杜绝了Enclave发生注入攻击的风险，实现了极致安全的Java机密计算运行环境。

• 主页: *https://teaclave.apache.org*

• 代码库: *https://github.com/apache/incubator-teaclave-java-tee-sdk*

Gramine

Gramine 是一个轻量级的 LibOS，旨在以最小的主机要求运行单个应用程序。Gramine 可以在一个隔离的环境中运行应用程序。其优点是可定制，易移植，方便迁移，可以媲美虚拟机。 在架构上 Gramine 可以在任何平台上支持运行未修改的 Linux 二进制文件。目前，Gramine 可以在 Linux 和 Intel SGX enclave 环境中工作。

• 主页：*https://gramine.readthedocs.io/*

• 代码库：*https://github.com/gramineproject/gramine*

TDX机密容器&机密虚拟机

Intel Trust Domain Extension (TDX) 基于虚拟化扩展机密计算的隔离能力，通过构建机密虚拟机，为业务负载提供了虚拟机级别的机密计算的运行环境。通过Linux社区对TDX机密虚拟机的生态支持, 基于Linux的业务应用可以方便的迁移到机密计算环境中。此外，Intel TDX Pod 级机密容器将TDX机密虚拟机技术同容器生态无缝集成，以云原生方式运行，保护敏感工作负载和数据的机密性和完整性。在机密虚拟机内部，默认集成了 image-rs 和 attestation-agent 等组件，实现了容器镜像的拉取、授权、验签、解密、远程证明以及秘密注入等安全特性。

机密计算平台

Confidential Computing Platform

海光CSV：海光安全虚拟化技术

项目位置链接

https://gitee.com/anolis/cloud-kernel
https://gitee.com/anolis/hygon-edk2
https://gitee.com/anolis/hygon-qemu
https://github.com/inclavare-containers/librats
https://github.com/inclavare-containers/rats-tls

技术自身介绍

背景

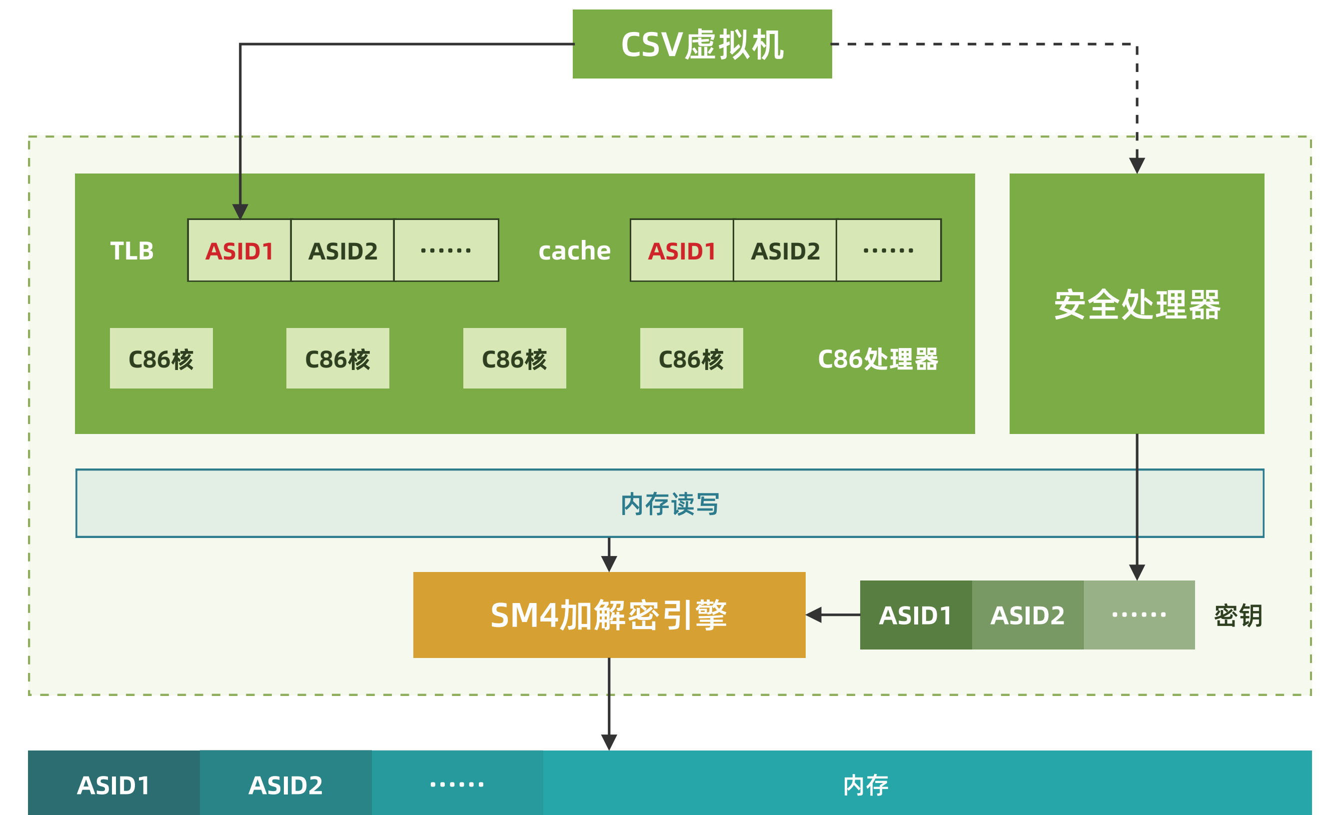
容器技术的出现，使应用程序的打包、分发变得非常简单易用，Kubernetes等容器编排技术的出现，进一步加速了容器生态的普及和发展，目前容器已经逐渐成为云计算的主要运行单元。但是由于传统容器共享操作系统内核，在隔离性和安全性上比传统虚拟机差。为了解决这个问题，Kata容器应运而生，Kata容器运行在轻量级虚拟机里，比起传统容器提供了更好的隔离性和安全性，使Kata容器同时具有容器技术带来的易用性和虚拟机技术带来的安全性。随着机密计算需求的出现，CPU厂商纷纷推出了硬件TEE技术，传统虚拟机技术已无法满足机密计算的需要，Kata容器的安全性需要进一步增强以便应用于机密计算场景。

问题&挑战

虚拟化是云计算的底层基础技术，随着云计算的发展而被广泛应用。由于虚拟机的全部资源被主机操作系统和虚拟机管理器管理和控制，虚拟化本身有较严重的安全缺陷，主机操作系统和虚拟机管理器可任意读取和修改虚拟机资源且虚拟机无法察觉。主机操作系统和虚拟机管理器有权读写虚拟机代码段，虚拟机内存数据，虚拟机磁盘数据，并有权重映射虚拟机内存等。攻击者可利用主机操作系统和虚拟机管理器的安全缺陷获取操作系统的权限后攻击虚拟机，给虚拟机最终用户造成重大损失。

解决方案

CSV是海光自主研发的安全虚拟化技术，采用国密算法实现，CSV虚拟机在写内存数据时CPU硬件自动加密，读内存数据时硬件自动解密，每个CSV虚拟机使用不同的密钥。海光CPU内部使用ASID（Address Space ID）区分不同的CSV虚拟机和主机，每个CSV虚拟机使用独立的Cache、TLB等CPU资源，实现CSV虚拟机、主机之间的资源隔离。CSV虚拟机使用隔离的硬件资源，支持启动度量、远程认证等功能，是安全的硬件可信执行环境。



CSV机密容器技术将安全虚拟化技术与Kata容器技术结合，实现容器运行环境的度量和加密，容器中的程序可以使用远程认证功能实现身份证明。CSV机密容器和普通容器的接口完全兼容，用户可以使用Docker或者Kubernetes启动机密容器，实现对容器数据的隔离和保护。

CSV技术构建了以安全加密虚拟机为基础的可信执行环境。在安全加密虚拟机保证了虚拟机数据机密性的基础上，更进一步保证了虚拟机数据的完整性，主机操作系统和虚拟机管理无法通过改写虚拟机嵌套页表对虚拟机实施重映射攻击。

应用场景

安全加密虚拟化可以保证最终用户数据的机密性和完整性，可用于实施机密计算，适用于云计算和隐私计算场景。

OpenAnolis
龙 蜥 社 区

Intel SGX: Intel安全防护扩展

项目位置链接

https://github.com/intel/linux-sgx
https://github.com/intel/SGXDataCenterAttestationPrimitives

技术自身介绍

背景

机密计算是信息安全行业内一项新兴技术，专注于帮助保护使用中的数据。机密计算旨在加密数据在内存中进行处理，同时降低将其暴露给系统其余部分的风险，从而降低敏感数据暴露的可能性，同时为用户提供更高程度的控制和透明度。在多租户云环境中，机密计算确保敏感数据与系统堆栈的其他特权部分保持隔离。

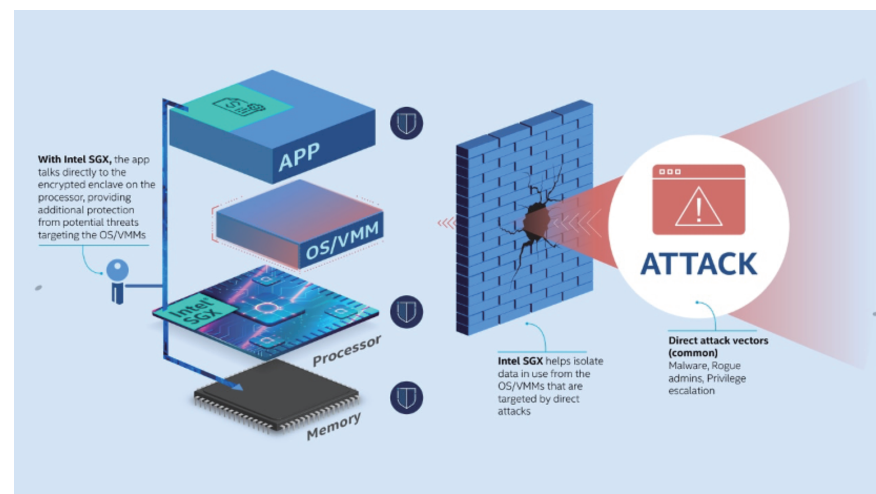
问题&挑战

传统操作系统提供了进程级内存隔离，但是无法保护用户的数据不被特权进程获取； 虚拟化技术基于特权软件Hypervisor对系统资源进行分配与监控，但Hypervisor潜在的软件漏洞有可能会威胁到整个系统; 基于TPM（Trusted Platform Module）的可信架构难以保障程序运行时的可信执行；TrustZone技术为程序提供了两种隔离的执行环境, 但需要硬件厂商的签名验证才能运行在安全执行环境，开发门槛较高。

解决方案

英特尔软件防护扩展（Intel Software Guard Extensions，SGX）是一组安全相关的指令，它被内置于一些现代Intel 中央处理器（CPU）中。它们允许用户态及内核态代码定义将特定内存区域，设置为私有区域，此区域也被称作飞地（Enclaves）。其内容受到保护，不能被本身以外的任何进程存取，包括以更高权限级别运行的进程。CPU对受SGX保护的内存进行加密处理。受保护区域的代码和数据的加解密操作在CPU内部动态完成。因此，处理器可以保护代码不被其他代码窥视或检查。

SGX提供了硬件指令级安全保障，保障了运行时的可信执行环境, 使恶意代码无法访问与篡改其他程序运行时的保护内容。Intel从第六代CPU开始支持SGX，SGX已经成为学术界的热点，各大云厂商也开始在云上部署基于SGX的应用。



应用场景

提供基础机密计算能力的支撑，在Intel SGX和TDX DCAP之上，支撑机密计算运行时，虚拟机，容器等具体的使用，最终让用户方便地将自己的workload运行到一个可信的机密计算环境当中。

用户情况

用户使用情况

对于龙蜥社区的机密计算应用，除了需要硬件上的支持，也需要软件基础架构的支持。所有的机密计算应用都会依赖于相应机密计算技术底层的软件开发包和运行时库。Intel为SGX和TDX Attestation提供了基础的软件架构支撑：主要包括SGX SDK，SGX PSW/(TDX) DCAP安装包的适配，和Anolis的集成。

用户使用效果

在vSGX虚拟机，TDX机密虚拟机，SGX LibOS运行时Occlum和Gramine当中，SGX SDK/PSW/(TDX) DCAP都提供了SGX，SGX 远程证明及TDX远程证明相关的软件支持（如API等）。

后续计划

在Anolis OS发布之前，Intel已经在CentOS和Alinux上支持了SGX SDK/PSW/DCAP软件包的构建和完整的测试。在2022年第二季度之后，我们又将TDX DCAP和SGX DCAP合并到统一的代码仓库。软件主要功能上准备完毕，安装包适配到Anolis OS的过程可以分为四个主要步骤：

1、完成相关安装包（RPM）的构建和测试
2、发布到Intel软件仓库
3、提供RPM Build Spec
4、集成到Anolis软件仓库

用户证言

Intel在龙蜥社区中将机密计算如SGX和TDX技术落地，服务于我们的客户。 Intel目前提供了以SGX SDK/PSW/DCAP, TDX DCAP软件包为代表的TEE基础架构支撑，和基于LibOS的运行时支持Gramine。并以这些为基础支持了其他LibOS运行时如蚂蚁的Occlum，以及更高层次的机密计算应用，例如SGX虚拟化及SGX/TDX机密容器。

Intel TDX: Intel安全虚拟化技术

技术自身介绍

背景

近年来，随着隐私保护的呼声和关注度越来越高，越来越多的人开始关注集中化的云基础设施存在泄露租户隐私和敏感数据的风险。在此背景下，机密计算这一新的计算形态应运而生。

机密计算是通过在基于硬件的可信执行环境（Trusted Execution Environment，TEE）中执行计算过程的全新计算模式，它能够对使用中的数据进行保护与隔离，防止具有特权的云基础设施提供方对租户的应用程序和数据进行未经授权的访问。

问题&挑战

越来越多的租户业务尤其是企业负载需要利用云计算提供的弹性资源进行海量数据处理。这类租户要求CSP能够提供更好的安全和隔离解决方案，尤其是在处理租户敏感数据这一计算阶段的过程中，租户的敏感数据不能以明文形式暴露在内存中，而租户的安全性又不依赖于CSP。

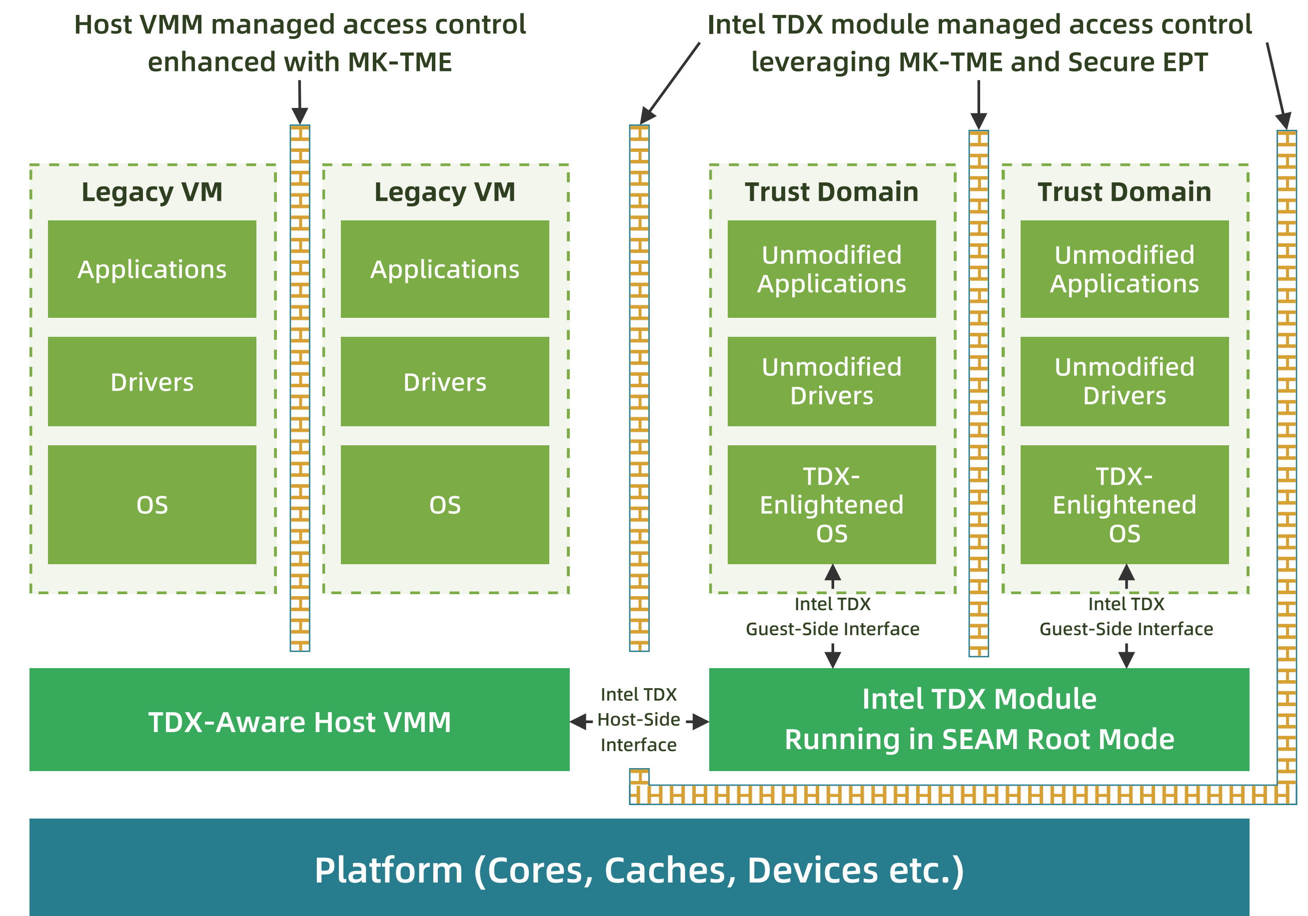
解决方案

Intel Trust Domain Extensions（简称TDX）引入了一种新的、基于硬件隔离的虚拟机工作负载形态，所谓的Trust Domain（简称TD）。TDX是一种典型的机密计算技术，可以在在租户不可信的云基础设施上，为租户的工作负载提供一个系统级（相比Intel SGX机密计算技术提供的应用级粒度）的安全可信的执行环境，同时保证租户运行环境的机密性和完整性。为此，需要将当前租户的TD，与CSP控制的特权级系统组件（比如VMM/hypervisor）、VM以及其他租户的TD都隔离开来，并将它们排除出当前租户的TCB，以确保当前租户的TD不受上述组件的影响。

与VM相比，TD额外增加了以下能力：

- 提供了VM内存的机密性和完整性保护
- 地址转换完整性保护
- CPU状态机密性和完整性保护
- 对安全中断和异常的分发机制
- 远程证明

TDX技术综合了MKTME（多密钥全加密内存）与VMX虚拟化技术，再添加新的指令集、处理器模式和强制实施的访问控制等设计。



用户情况

目前Intel TDX已经在主流云平台上提供相关实例或部署计划：

1、 阿里云8代ECS 提供的TDX机密计算实例已经上线邀测。

2、 Microsoft Azure TDX机密计算实例也计划今年晚些时候提供服务。

OpenAnolis 龙蜥社区

AMD SEV: AMD安全加密虚拟化技术

项目位置链接

https://github.com/AMDESE/AMDSEV

技术自身介绍

背景

机密计算指的是在Secure or Trusted执行环境中进行计算操作，以防止正在做计算的敏感数据被泄露或者修改。

机密计算的核心是Trusted Execution Environment (TEE)，TEE既可基于硬件实现、也可基于软件实现，本文主要专注于基于硬件和密码学原理的实现，相比于纯软件解决方案，具有较高的通用性、易用性、可靠性、较优的性能以及更小的TCB。其缺点是需要引入可信方，即信任芯片厂商。此外由于CPU相关实现属于TCB，侧信道攻击也成为不可忽视的攻击向量，需要关注相关漏洞和研究进展。

问题&挑战

随着越来越多的业务上云，端到端的全链路可信或机密正在慢慢成为公有云基础设施的默认要求而不再是一个特性，需要综合利用加密存储、安全网络传输、机密计算等技术来实现对用户敏感数据全生命周期的保护。

机密计算是当前业界正在补齐的环节，主流的硬件平台已经部分提供或正在实现对机密计算的支持，如AMD SEV，Intel TDX和SGX，Arm CCA，IBM SE和RISC-V的KeyStone等。

解决方案

AMD SEV技术基于AMD EPYC CPU，将物理机密计算能力传导至虚拟机实例，在公有云上打造一个立体化可信加密环境。

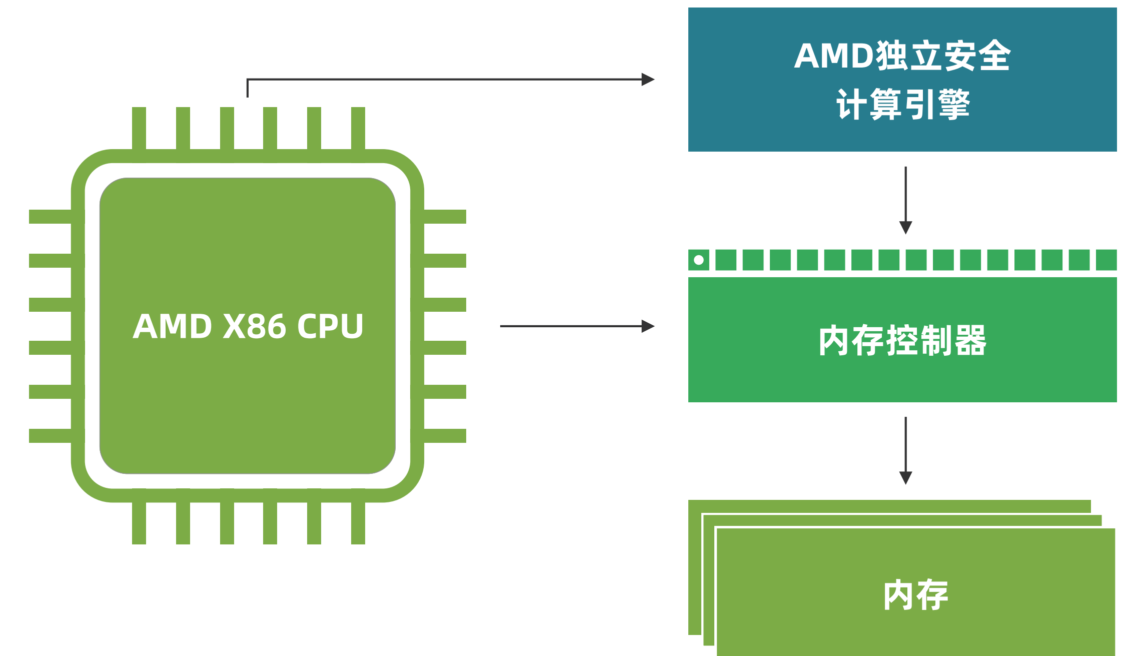
SEV可保证单个虚拟机实例使用独立的硬件密钥对内存加密，同时提供高性能支持。密钥由AMD平台安全处理器 (PSP)在实例创建期间生成，而且仅位于处理器中，云厂商无法访问这些密钥。

AMD SEV提供了硬件级的内存加密方案, 用以实现安全加密的虚拟化:

内存控制器中集成了AES-128硬件加速引擎: 客户操作系统通过页表选择要加密的页, 对终端用户的应用程序没有任何改变。

AMD安全内存加密（SME）: 所有内存由单一的密钥进行加密, 仅仅在BIOS中开启就可以实现（TSME）。

AMD安全加密虚拟化（SEV）: 每台虚拟机都会被分配自己的独立加密密钥, 宿主机和客户虚拟机之间相互加密隔离。



Confidential Virtual Machine也可以称作Secure Virtual Machine (SVM)，是最终实现机密计算的实体，本身作为一个可信域存在，SVM自身和在其中运行的用户程序可以不受来自SVM之外的非可信特权软件和硬件的的攻击，从而实现对用户的机密数据和代码的保护。

因为是整个虚拟机作为一个可信域，所以对运行于其中的用户程序可以做到透明，无需重构用户程序，对最终用户而言可以实现零成本可信 （Trust Native）。

用户情况

目前AMD SEV 已经在主流的云平台上都已经提供相关的实例和部署:

1、阿里云G8ae实例商业化中。

2、Google公有云已经提供2代AMD SEV的机密计算实例。

3、Microsoft Azure公有云已经提供了AMD SEV的机密计算实例和机密计算容器方案。

ARM CCA: Arm安全加密虚拟化技术

项目位置链接

Veracuz: *https://github.com/veracruz-project/veracruz*

VERAISON-VERificAtIon of atteStatiON: *https://github.com/veraison/veraison*

技术自身介绍

问题&挑战

TrustZone是Arm为设备安全提供的一个安全架构，通过硬件隔离和权限分层的方式将系统内分为安全世界（Secure world）和正常世界（Normal / Non-Secure world）。

在安全环境中，通过底层硬件隔离，不同执行级别，安全鉴权方式等方式，从最根本的安全机制上提供基于信任根（Root of Trust）的可信执行环境TEE（Trusted Execution Environment）,通过可信服务（Trusted Services）接口与和通用环境REE（Rich Execution Environment）进行安全通信，可以保护TEE中的安全内容不能被非安全环境的任何软件，包括操作系统底层软件等所访问，窃取，篡改和伪造等。因此一些安全的私密数据，比如一些安全密钥，密码，指纹以及人脸数据等都是可以放在安全世界的数据区中进行保护。当前，Trustzone机制已经非常成熟稳定，并得到大规模的应用，并以开源的方式给业界提供实现参考。可以访问https://www.trustedfirmware.org/ 获取更多信息。

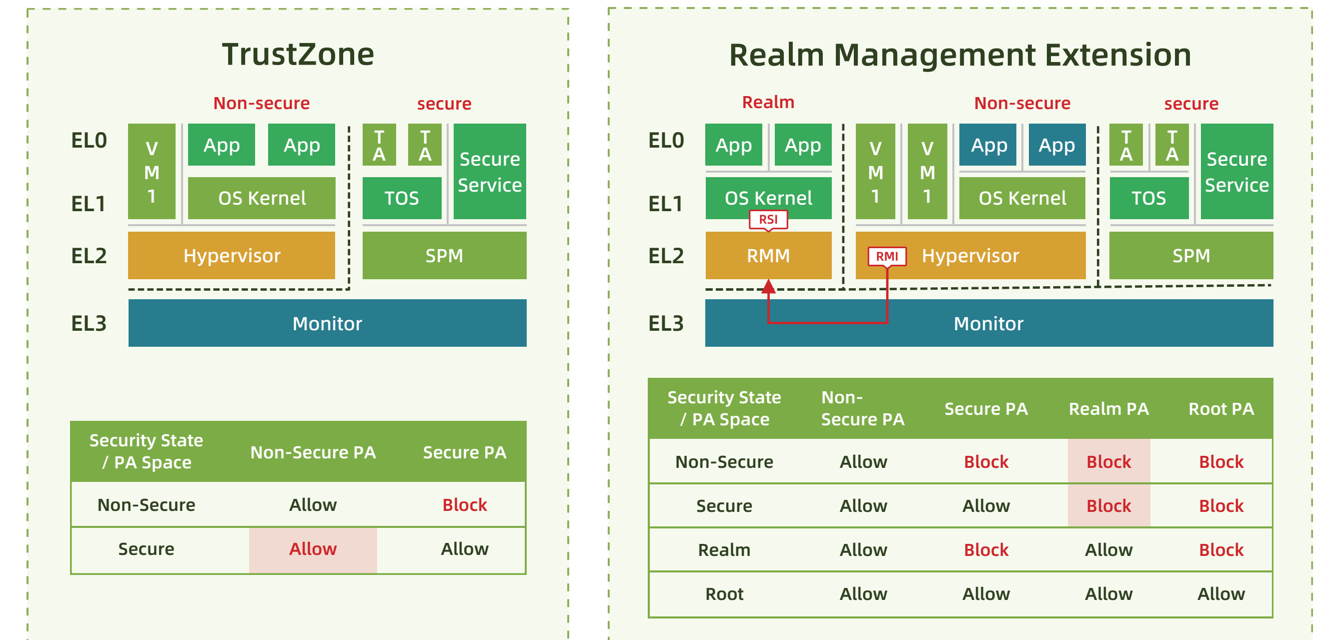
然而，TrustZone所提供的安全机制和TEE环境只能提供硬件级别的安全隔离。通常情况下，安全物理地址空间的内存在系统引导时静态分配，适用于数量有限的平台。对于大规模云服务器的机密计算，旨在允许任何第三方开发人员保护他们的虚拟机（VM）或应用程序，必须能够在运行时保护与VM或应用程序关联的任何内存，而不受限制或分割。

解决方案

Arm CCA引入了一种新的机密计算世界：机密领域（Realm）。在Arm CCA中，硬件扩展被称为Realm Management Extension (RME)，RME 会和被称之为机密领域管理监控器 (Realm Management Monitor, RMM)，用来控制机密领域的专用固件，及在 Exception level 3 中的 Monitor 代码交互。

Realm是一种Arm CCA环境，能被Normal world主机动态分配。主机是指能管理应用程序或虚拟机的监控软件。Realm及其所在平台的初始化状态都可以得到验证。这一过程使Realm的所有者能在向它提供任何机密前就建立信任。因此，Realm不必继承来自控制它的Non-secure hypervisor的信任。主机可以分配和管理资源配置,管理调度Realm虚拟机。然而，主机不可以监控或修改Realm执行的指令。在主机控制下，Realm可以被创建并被销毁。通过主机请求，可以增加或移除页面，这与hypervisor管理任何其他非机密虚拟机的操作方式类似。

Arm Confidential Compute Architecture



Security State / PA Space	Non-Secure PA	Secure PA
Non-Secure	Allow	Block
Secure	Allow	Allow

Security State / PA Space	Non-Secure PA	Secure PA	Realm PA	Root PA
Non-Secure	Allow	Block	Block	Block
Secure	Allow	Allow	Block	Block
Realm	Allow	Block	Allow	Block
Root	Allow	Allow	Allow	Allow

Arm CCA builds on widely adopted TrustZone programming model to protect User Code and Data

Arm CCA技术能够从根本上解决用户敏感应用数据的安全计算问题。它充分利用软硬件实现的信任根提供的数据和程序的物理隔离、保护、通信和认证体系，并在传统TrustZone的基础上，增加了被称为领域（Realm）的隔离区域，从最底层的安全机制和原理上解决用户程序和数据的隔离需求。

当前应用情况

Realm内运行的代码将管理机密数据或运行机密算法，这些代码需要确保正在运行真正的Arm CCA平台，而不是冒充者。这些代码还需要知道自己已经被正确地加载，没有遭到篡改。并且，这些代码还需要知道整个平台或Realm并不处于可能导致机密泄露的调试状态。建立这种信任的过程被称为"证明"。ARM正在与包括机密计算联盟成员在内的主要行业合作伙伴合作，定义这一证明机制的属性，确保在不同的产品和设备上使用常见的平台真实性和来源方法。Arm主导的开源软件Veracuz是一个框架，用于在一组相互不信任的个人之间定义和部署协作的、保护隐私的计算；VERAISON-VERificAtIon of atteStatiON构建可用于证明验证服务的软件组件。

Armv9-A架构引入了Arm CCA的RME功能特性，采用对应架构的芯片也将拥有此项功能。此外，基于CCA的软件支持已经在虚拟硬件平台上进行开发、测试和验证，将在硬件设备问世的同时实现同步支持。更多信息，可以访问 https://arm.com/armcca 获取更多信息。

框架编程

Programming Framework

Intel SGX SDK/PSW/DCAP: Intel SGX软件开发套件和平台软件服务

Intel SGX Platform Software and Datacenter Attestation Primitives （PSW/DCAP） 在龙蜥生态中为数据中心和云计算平台提供 Intel SGX 技术所需的平台软件服务，如远程证明等。Intel SGX SDK在龙蜥生态中为开发者提供使用 Intel SGX 技术所需的软件开发套件，帮助开发者高效便捷地开发机密计算程序和解决方案。本文介绍如何在Anolis OS 8.6当中基于Intel SGX SDK/PSW/DCAP构建SGX机密计算环境，并演示如何运行示例代码以验证SGX功能。

背景信息

Intel® SGX以硬件安全保障信息安全，不依赖固件和软件的安全状态，为用户提供物理级的机密计算环境。Intel® SGX通过新的指令集扩展与访问控制机制实现SGX程序的隔离运行，保障关键代码和数据的机密性与完整性不受恶意软件的破坏。不同于其他安全技术，Intel® SGX的可信根仅包括硬件，避免了基于软件的可信根可能自身存在安全漏洞的缺陷，极大地提升了系统安全保障。

检查 SGX 使能状态

构建SGX机密计算环境前，您可以通过cpuid检查SGX使能状态。

1、安装cpuid。

```
sudo yum install -y cpuid
```

2、检查SGX使能状态。

```
cpuid -1 -l 0x7 | grep SGX
```

说明 SGX被正确使能后，运行SGX程序还需要SGX驱动。

3、检查SGX驱动安装情况。

```
ls -l /dev/{sgx_enclave,sgx_provision}
```

开始构建SGX机密计算环境

为开发SGX程序，您需要在您的机器上安装运行时（runtime）、SDK，并配置远程证明服务。本文以Anolis OS 8.6为例演示构建过程，您也可以直接参考Intel官方提供的Intel® SGX软件安装指南安装所需的驱动、PSW等。

1. 安装SGX运行时

```
mkdir -p $HOME/sgx && \
  cd $HOME/sgx && \
  wget https://download.01.org/intel-sgx/latest/linux-latest/distro/Anolis86/sgx_rpm_local_repo.tgz --no-check-certificate && \
  tar zxvf sgx_rpm_local_repo.tgz && \
  sudo yum install -y yum-utils && \
  sudo yum-config-manager --add-repo file://$HOME/sgx/sgx_rpm_local_repo/ && \
  sudo yum install --nogpgcheck -y sgx-aesm-service libsgx-launch libsgx-urts && \
  rm -rf sgx_rpm_local_repo.tar.gz
```

可按需安装更多库：

```
sudo yum install --nogpgcheck -y libsgx-ae-le libsgx-ae-pce libsgx-ae-qe3 libsgx-ae-qve \
  libsgx-aesm-ecdsa-plugin libsgx-aesm-launch-plugin libsgx-aesm-pce-plugin \
  libsgx-aesm-quote-ex-plugin libsgx-dcap-default-qpl libsgx-dcap-ql \
  libsgx-dcap-quote-verify libsgx-enclave-common libsgx-launch libsgx-pce-logic \
  libsgx-qe3-logic libsgx-quote-ex libsgx-ra-network libsgx-ra-uefi \
  libsgx-uae-service libsgx-urts sgx-ra-service sgx-aesm-service
```

说明 SGX AESM（Architectural Enclave Service Manager）负责管理启动Enclave、密钥配置、远程认证等服务，默认安装路径为/opt/intel/sgx-aesm-service。

2. 安装SGX SDK

```
cd $HOME/sgx && \
  export SGX_VERSION="2.18.100.3" && \
  wget https://download.01.org/intel-sgx/latest/linux-latest/distro/Anolis86/sgx_linux_x64_sdk_$SGX_VERSION.bin --no-check-certificate && \
  chmod +x sgx_linux_x64_sdk_$SGX_VERSION.bin && \
  echo -e 'n\n\/opt/intel\n' | ./sgx_linux_x64_sdk_$SGX_VERSION.bin && \
  rm -rf sgx_linux_x64_sdk_$SGX_VERSION.bin && \
  source /opt/intel/sgxsdk/environment
```

安装Intel® SGX SDK后，您可以参见Intel® SGX Developer Reference开发SGX程序。

说明 Intel® SGX SDK的默认安装目录为 `/opt/intel/sgxsdk/` 。

3. 配置SGX远程证明服务

Intel® SGX ECDSA远程证明服务通过远程证明来获得远程提供商或生产者的信任，为SGX SDK提供下列信息：

1、SGX certificates：SGX证书。

2、Revocation lists：已被撤销的SGX证书列表。

3、Trusted computing base information：可信根信息。

说明 Intel Ice Lake仅支持基于Intel SGX DCAP的远程证明方式，不支持基于Intel EPID的远程证明方式，您可能需要适配程序后才能正常使用远程证明功能。更多远程证明的信息，请参见attestation-service。

配置QPL及PCK缓存服务：

1、安装Quote Provider Library (QPL)。您可以使用自己定制的 QPL 或使用 Intel 提供的默认 QPL (libsgx-dcap-default-qpl)

2、安装 PCK 缓存服务。 PCK Caching Service的安装配置请参考Intel官方PCCS文档：SGXDataCenterAttestationPrimitives

3、确保 PCK 缓存服务由本地管理员或数据中心管理员正确设置。还要确保引用提供程序库的配置文件（/etc/sgx_default_qcnl.conf）与真实环境一致，例如：

```
PCS_URL=https://your_pcs_server:8081/sgx/certification/v3/
```

验证SGX功能示例一：启动Enclave

Intel® SGX SDK中提供了SGX示例代码用于验证SGX功能，默认位于/opt/intel/sgxsdk/SampleCode目录下。

本节演示其中的启动Enclave示例（SampleEnclave），效果为启动一个Enclave，以验证是否可以正常使用安装的SGX SDK。

1. 安装编译工具及相关依赖

```
yum install -y gcc-c++
```

2. 设置SGX SDK相关的环境变量

```
source /opt/intel/sgxsdk/environment
```

3. 编译示例代码SampleEnclave

- 进入SampleEnclave目录

```
cd /opt/intel/sgxsdk/SampleCode/SampleEnclave
```

- 编译SampleEnclave

```
make
```

- 运行编译出的可执行文件

```
./app
```

预期的结果为：

```
Checksum(0x0x7ffd989a81e0, 100) = 0xfffd4143
Info: executing thread synchronization, please wait...
```

```
Info: SampleEnclave successfully returned.
Enter a character before exit ...
```

验证SGX功能示例二：SGX远程证明示例

Intel® SGX SDK中提供了SGX示例代码用于验证SGX功能，默认位于/opt/intel/sgxsdk/SampleCode目录下。

本节演示其中的SGX远程证明示例（QuoteGenerationSample、QuoteVerificationSample），效果为生成和验证Quote。该示例涉及被挑战方（在SGX实例中运行的SGX程序）和挑战方（希望验证SGX程序是否可信的一方），其中QuoteGenerationSample为被挑战方生成Quote的示例代码，QuoteVerificationSample为挑战方验证Quote的示例代码。

1. 安装编译工具及相关依赖

```
yum install -y git
```

2. 设置SGX SDK相关的环境变量

```
source /opt/intel/sgxsdk/environment
```

3. 安装SGX远程证明依赖的包

```
yum install --nogpgcheck -y libsgx-dcap-ql-devel libsgx-dcap-quote-verify-devel
```

4. 编译被挑战方示例代码QuoteGenerationSample

- 进入QuoteGenerationSample目录

```
git clone https://github.com/intel/SGXDataCenterAttestationPrimitives -b DCAP_1.15
cd SGXDataCenterAttestationPrimitives/SampleCode/QuoteGenerationSample
```

- 编译QuoteGenerationSample

```
make
```

- 运行编译出的可执行文件生成Quote

```
./app
```

预期的结果为：

```
sgx_qe_set_enclave_load_policy is valid in in-proc mode only and it is optional: the default
enclave load policy is persistent:
set the enclave load policy as persistent:succeed!

Step1: Call sgx_qe_get_target_info:succeed!
Step2: Call create_app_report:succeed!
Step3: Call sgx_qe_get_quote_size:succeed!
Step4: Call sgx_qe_get_quote:succeed!cert_key_type = 0x5
sgx_qe_cleanup_by_policy is valid in in-proc mode only.

 Clean up the enclave load policy:succeed!
```

5. 编译挑战方示例代码QuoteVerificationSample

- 进入QuoteGenerationSample目录

```
git clone https://github.com/intel/SGXDataCenterAttestationPrimitives -b DCAP_1.15
cd SGXDataCenterAttestationPrimitives/SampleCode/QuoteVerificationSample
```

- 编译QuoteVerificationSample

```
make
```

- 生成签名密钥

```
openssl genrsa -out Enclave/Enclave_private_sample.pem -3 3072
```

- 对QuoteVerificationSample Enclave进行签名

```
/opt/intel/sgxsdk/bin/x64/sgx_sign sign -key Enclave/Enclave_private_sample.pem -enclave
enclave.so -out enclave.signed.so -config Enclave/Enclave.config.xml
```

- 运行编译出的可执行文件验证Quote

```
./app
```

预期的结果为：

```
Info: ECDSA quote path: ../QuoteGenerationSample/quote.dat

Trusted quote verification:
    Info: get target info successfully returned.
    Info: sgx_qv_set_enclave_load_policy successfully returned.
    Info: tee_get_quote_supplemental_data_version_and_size successfully returned.
    Info: latest supplemental data major version: 3, minor version: 1, size: 336
    Info: App: tee_verify_quote successfully returned.
    Info: Ecall: Verify QvE report and identity successfully returned.
    Info: App: Verification completed successfully.
    Info: Supplemental data Major Version: 3
    Info: Supplemental data Minor Version: 1

===========================================

Untrusted quote verification:
    Info: tee_get_quote_supplemental_data_version_and_size successfully returned.
    Info: latest supplemental data major version: 3, minor version: 1, size: 336
```

```
Info: App: tee_verify_quote successfully returned.
  Info: App: Verification completed successfully.
  Info: Supplemental data Major Version: 3
  Info: Supplemental data Minor Version: 1
```


OpenAnolis 龙蜥社区

Intel Homomorphic Encryption: Intel平台同态加密加速框架

项目位置链接

Intel HE Toolkit: *https://github.com/intel/he-toolkit*

Intel HEXL: *https://github.com/intel/hexl*

Intel Paillier Cryptosystem Library (IPCL): *https://github.com/intel/pailliercryptolib*

HEBench: *https://github.com/hebench*

技术自身介绍

背景

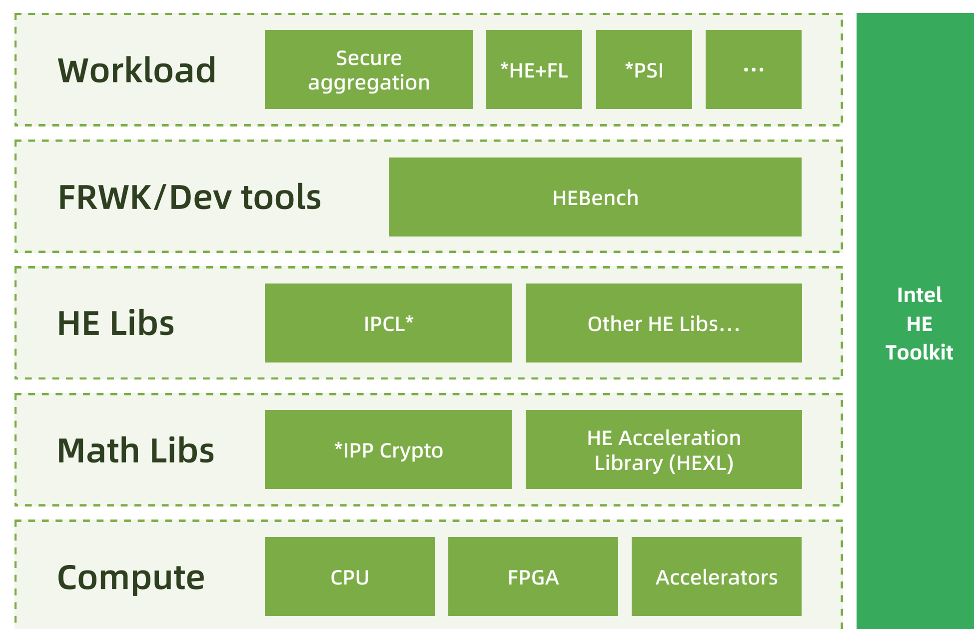
同态加密是一种满足同态运算性质的加密技术，即在对密文进行特定形式的代数运算之后，得到的仍然是加密的结果，将其解密所得到的结果与对原始明文进行运算得出的结果一致。由于运算完全在密文上进行，同态加密可以提供极高的数据安全保证，其被广泛地应用于隐私保护计算、联邦学习和区块链等领域。

问题&挑战

虽然同态加密技术发展了许多年，但由于其技术复杂度高，对计算、存储和网络带宽资源要求高等原因，面临着开发、使用和部署门槛高的问题。为了解决上述问题，Intel提供了一系列工具套件和加速库，包括Intel HE Toolkit、Intel HE Acceleration Library （Intel HEXL）和Intel Paillier Cryptosystem Library（IPCL）。另一方面，同态加密技术正在经历标准化的过程，尽管涌现出了许许多多同态加密应用，但是缺乏一个公认的基准评价体系，针对这个问题，Intel向开源社区贡献了Homomorphic Encryption Benchmarking Framework（ HEBench）。

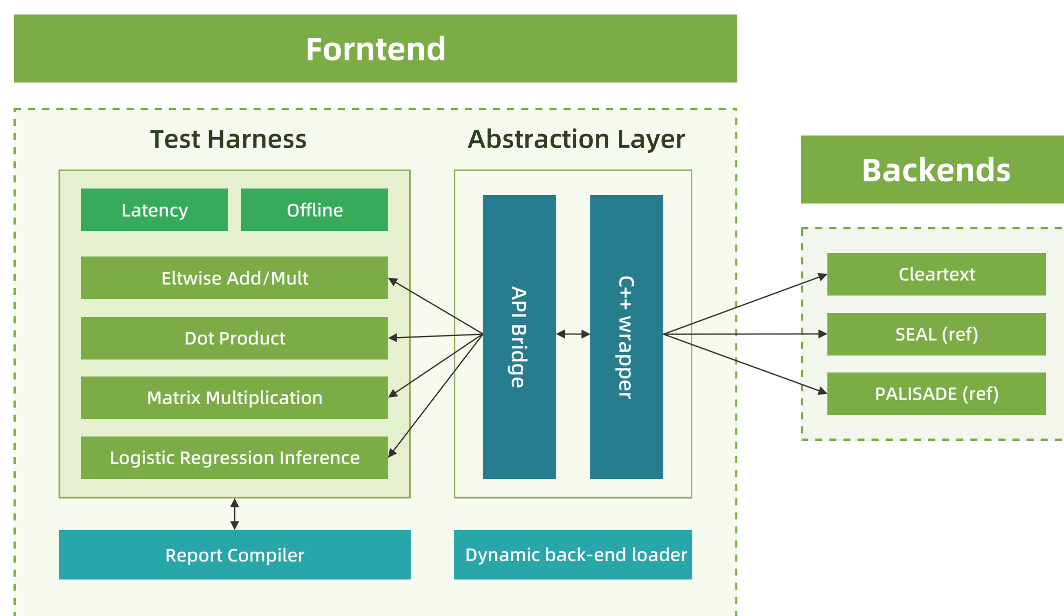
解决方案

如下图所示，Intel提供了对于同态加密技术的全栈式支持。在半同态加密方面，Intel提供了通过ISO认证的Paillier算法库IPCL，它使用了Intel IPP-Crypto库提供的AVX512IFMA指令集加速的能力，在最新的Intel Xeon平台上有优异的性能表现。在Leveled HE方面，依托于Intel Xeon平台最新的指令集加速以及FPGA的硬件加速能力，Intel HEXL封装了底层硬件细节，向上提供了加速的密码学运算接口供第三方同态加密库，如SEAL、Palisade和HELib使用。Intel HEXL在最新的Intel平台上同样采用了AVX512IFMA指令集加速伽罗华域上的大整数运算，并且可以根据CPU型号自动匹配最合适的加速实现，提供向后兼容功能。



围绕 HE 技术创新， Intel HE Toolkit 为社区和行业提供了一个用于HE 应用实验、开发和部署的平台。Intel HE Toolkit目前提供示例内核和示例程序，这些示例程序演示了利用主流的同态加密库构建各种同态加密应用用于保护用户隐私数据的能力。 同时，Intel HE Toolkit通过使用Intel HEXL 加速库展示了使用最新的英特尔硬件功能发挥英特尔处理器在HE领域优势的最佳实践。

此外，HEBench为同态加密方案的开发者和用户提供了一个公允且标准的性能测试基准，旨在推动同态加密技术的演进和应用方案的创新。HEBench由三个部分组成：



1、前端：包括测试框架和后端加载器。测试框架目前支持的运算有向量按元素的加法和乘法、向量点乘、矩阵乘法和逻辑回归推理，主要度量的指标是计算时延和吞吐量。

2、适配层：提供了统一的C接口，桥接了前端的测试框架和后端具体的HE实现，使得后端开发者可以专注于HE各类算法的实现，只需要遵照接口定义就能很容易接入HEBench的测试框架。

3、后端：支持开发者接入各类HE实现。目前HEBench开源项目中自带几个后端的参考实现，包括明文后端、SEAL后端和Palisade后端。

OpenAnolis
龙 蜥 社 区

应用场景

同态加密天然适合云计算场景，因此最先在这一领域得以应用。用户希望借助云服务器强大的计算资源来进行复杂的计算，但又不希望暴露自己的私有数据。在同态加密进入这一领域之前，在信任云服务提供商不会泄露或者窃取用户数据的前提下，用户直接将明文交给云服务器进行计算，这样始终存在一定的安全风险。而有了同态加密之后，用户只需要将私有数据加密之后的密文传输到云端，在云端直接以密文形式参与计算，最终结果也是以密文的形式返回给用户，在用户本地进行解密。云计算在同态加密的加持下，安全性得到了充分的保证。

场景描述

在联邦学习的应用场景中，各参与方通过同态加密技术在保证各自数据隐私的情况下，共同参与训练机器学习模型，提高模型训练精度。在横向联邦学习中，各参与方在本地用各自的数据训练模型，将训练得到的模型参数加密上传参数服务器。参数服务器用Paillier等加法半同态加密技术聚合来自多方的模型参数，最后更新的模型参数仍然以密文形式发送给各参与方。利用同态加密技术可以防止攻击者从上传的模型参数中反推出用户的本地数据。在纵向联邦学习的隐私求交部分，同态加密也能发挥重要作用，从而保证在不暴露各方私有数据的情况下，求取用于联邦学习的训练数据的交集。

在区块链应用中，为了保护链上存证信息的隐私性，可以对这些数据进行同态加密，并将计算过程转化为同态运算过程，节点即可在无需获知明文数据的情况下实现密文计算。

应用效果

微众银行的联邦学习框架FATE采用了IPCL加速的Paillier方案，在第三代Xeon平台上的测试数据显示，对2048比特精度的整型数据进行模幂运算，相比于GMP方案有4. 7倍的性能提升。IPCL的应用极大地提高了用户使用联邦学习方案的效率，有效地减少了TCO。

蚂蚁集团的开源隐私计算框架“隐语”也正式接入 Intel IPCL 加速的 Paillier 方案，在第四代 Xeon 平台上的测试数据显示 IPCL 的应用显著提高了隐私计算的效率。

Intel_HE_Toolkit开发指南

本文首先介绍了如何在Anolis 8.6上分别以源码和二进制包形式编译与安装Intel HE Toolkit，以及如何运行HE相关例程。

源码安装

源码下载

```
sudo yum install -y git
git clone -b v2.0.1 https://github.com/intel/he-toolkit.git
```

安装依赖

- 安装系统依赖

```
sudo yum install -y m4 \
            patchelf \
            cmake \
            gcc-toolset-10 \
            glibc-devel \
            virtualenv \
            autoconf \
            wget \
            bzip2 \
            python38
```

• 安装python依赖

```
cd he-toolkit
sudo pip3.8 install -r requirements.txt
sudo pip3.8 install -r dev_reqs.txt
```

• 安装gmp-6.2.1

```
wget https://gmplib.org/download/gmp/gmp-6.2.1.tar.xz
tar -xf gmp-6.2.1.tar.xz
cd gmp-6.2.1
./configure
make
sudo make install
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

编译安装

• 初始化编译环境

```
sudo update-alternatives --config python3 # 选择python3.8选项
scl enable gcc-toolset-10 bash
cd he-toolkit
./hekit init --default-config
#根据命令提示 source bash_profile
```

• 编译和安装第三方依赖

```
./hekit install recipes/default.toml
```

> 如果遇到找不到某个依赖包的问题，可以通过修改 `default.toml` 中相应的安装路径来解决。比如：如果提示找不到zstd，需要修改default.toml里面的 `export_cmake = "install/lib/cmake/zstd"` 改成 `export_cmake = "install/lib64/cmake/zstd"`

• 查看编译安装的包

```
./hekit list
```

运行示例应用

1、逻辑回归

这是一个用SEAL CKKS实现的逻辑回归推理，可以根据实际需要选择数据和模型都是密文或者其中一方是密文而另一方是明文的配置。通过传递命令行参数 `--linear_regression` ，也可以实现线性回归推理。

• 编译

```
cd he-toolkit/he-samples/examples/logistic-regression
cmake -S . -B build -DSEAL_DIR=$HOME/.hekit/components/seal/v3.7.2/install/lib64/cmake/SEAL-3.7 \
      -DMicrosoft.GSL_DIR=$HOME/.hekit/components/gsl/v3.1.0/install/share/cmake/Microsoft.GSL \
      -Dzstd_DIR=$HOME/.hekit/components/zstd/v1.4.5/install/lib64/cmake/zstd \
      -DHEXL_DIR=$HOME/.hekit/components/hexl/1.2.3/install/lib/cmake/hexl-1.2.3
cmake --build build
```

• 运行

```
cd build
./lr_test
```

• 预期输出结果

```
INFO: Wed Nov 2 03:21:03 2022: Loading EVAL dataset
INFO: Wed Nov 2 03:21:03 2022: dataLoader: datasets/lrtest_mid_eval.csv
INFO: Wed Nov 2 03:21:04 2022: Loading EVAL dataset complete Elapsed(s): 0.102
INFO: Wed Nov 2 03:21:04 2022: Input data size: (samples) 2000 (features) 80
INFO: Wed Nov 2 03:21:04 2022: Loading Model
INFO: Wed Nov 2 03:21:04 2022: weightsLoader: datasets/lrtest_mid_lrmodel.csv
INFO: Wed Nov 2 03:21:04 2022: Loading Model complete Elapsed(s): 0
INFO: Wed Nov 2 03:21:04 2022: Encode/encrypt weights and bias
INFO: Wed Nov 2 03:21:04 2022: Encode/encrypt weights and bias complete Elapsed(s): 0.041
INFO: Wed Nov 2 03:21:04 2022: HE LR
INFO: Wed Nov 2 03:21:04 2022: # of batches: 1 Batch size: 4096
INFO: Wed Nov 2 03:21:04 2022: Transpose data
INFO: Wed Nov 2 03:21:04 2022: - Transpose data complete Elapsed(s): 0.005
INFO: Wed Nov 2 03:21:04 2022: Encode/encrypt data
INFO: Wed Nov 2 03:21:04 2022: - Encode/encrypt data complete! Elapsed(s): 0.026
INFO: Wed Nov 2 03:21:04 2022: Logistic Regression HE: 1 batch(es)
INFO: Wed Nov 2 03:21:04 2022: - LR HE complete! Elapsed(s): 0.085
INFO: Wed Nov 2 03:21:04 2022: Decrypt/decoding LRHE result
INFO: Wed Nov 2 03:21:04 2022: - Decrypt/decode complete! Elapsed(s): 0.003
INFO: Wed Nov 2 03:21:04 2022: HE inference result - accuracy: 0.855
```

2、安全查询

这个例子演示了如何用同态加密技术（SEAL BFV scheme)实现安全数据库查询。在这个例子中，数据查询条件和数据库本身都是加密的状态。查询客户端负责初始化加密上下文、产生同态加密密钥对、加密查询条件和解密查询结果。查询服务端负责存储加密数据库，并实现加密数据库查询算法。

• 编译

```
cd he-toolkit/he-samples/examples/logistic-regression
cmake -S . -B build -DSEAL_DIR=$HOME/.hekit/components/seal/v3.7.2/install/lib64/cmake
/SEAL-3.7 \
-DMicrosoft.GSL_DIR=$HOME/.hekit/components/gsl/v3.1.0/install/share/cmake/
  Microsoft.GSL \
-Dzstd_DIR=$HOME/.hekit/components/zstd/v1.4.5/install/lib64/cmake/zstd \
-DHEXL_DIR=$HOME/.hekit/components/hexl/1.2.3/install/lib/cmake/hexl-1.2.3
cmake --build build
```

• 运行

```
cd build
./secure-query
```

• 预期输出结果

```
Initialize SEAL BFV scheme with default parameters[Y(default)|N]:
SEAL BFV context initialized with following parameters
Polymodulus degree: 8192
Plain modulus: 17
Key length: 8
Input file to use for database or press enter to use default[us_state_capitals.csv]:
Number of database entries: 50
Encrypting database entries into Ciphertexts
Input key value to use for database query:Oregon
Querying database for key: Oregon
Decoded database entry: Salem

Total query elapsed time(seconds): (Time in seconds for database query)
Records searched per second: (number of records searched per second)
```

二进制包安装

除了直接通过源代码编译安装，Intel HE Toolkit即将在下一个Anolis版本中支持通过二进制包的形式安装。

参考

Intel HE Toolkit: *https://github.com/intel/he-toolkit*

Apache Teaclave Java TEE SDK：面向Java生态的机密计算编程框架

项目位置链接

https://github.com/apache/incubator-teaclave-java-tee-sdk

技术自身介绍

背景

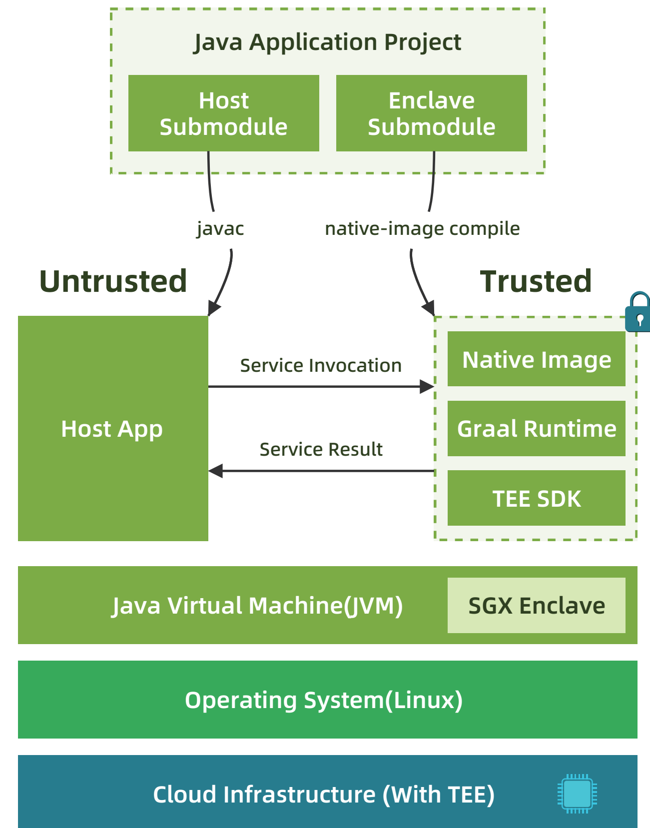
数据在存储和传输状态的安全性通过加解密得到了很好的解决，但数据在运行时是以明文的方式参与计算的，很容易出现泄漏，造成不可估量的风险。机密计算技术正是为了解决运行时数据安全问题而生，它通过处理器提供一个基于芯片的可信执行环境(TEE)，将敏感数据和代码放置在该TEE内执行，TEE对整个计算过程进行严格保护，有效阻止TEE之外的组件(包括操作系统)获取或篡改TEE内的代码和数据，保证敏感代码和数据的安全性。

问题&挑战

Intel SGX技术提供了极高安全等级的可信执行环境，但使用该技术需要对已有的应用代码进行改造，SGX SDK只提供了对C语言生态的支持，此外用户需要用.edl文件定义服务接口，开发过程繁琐，对开发者的编程习惯冲击较大，造成开发门槛很高，阻碍了该技术的发展与应用。

解决方案

Teaclave Java TEE SDK提供基于Intel SGX技术的Java生态机密计算开发框架。采用Java静态编译技术将Enclave Module编译成Native代码并运行在SGX TEE中，实现对高级语言的支持并最大限度保持较低的系统TCB。屏蔽底层交互细节，用户无须定义edl接口文件。给用户提供一个Pure Java的机密计算开发框架和编译构建工具链，极大降低Intel SGX的开发门槛。



将机密计算从C/C++应用生态扩展到Java生态，在不牺牲机密安全性的前提下，极大提升开发效率和用户体验。

应用场景

场景描述

Teaclave Java TEE SDK可应用于对数据和算法敏感的领域。比如政府部门、金融、区块链、医疗和联邦计算等；

在阿里云DataTrust隐私增强计算平台，Teaclave Java TEE SDK应用在SQL安全审计和文件转加密等两个微服务模块中；

应用效果

基于Teaclave Java TEE SDK帮助用户开发Pure Java机密计算应用，并保证系统安全性和性能，提升机密计算应用开发效率与体验。

竞品分析

Intel SGX SDK与OpenEnclave仅支持C/C++生态机密计算应用开发，且开发门槛高；

Occlum LibOS技术降低了机密计算开发与部署难度，但系统TCB很大，牺牲了应用部分机密性。

Apache_Teaclave_Java_TEE_SDK最佳实践

1. 背景信息

Intel SGX技术提供了极高安全等级的可信执行环境，但使用该技术需要对已有的应用代码进行改造，SGX SDK只提供了对C语言生态的支持，此外用户需要用.edl文件定义服务接口，开发过程繁琐，对开发者的编程习惯冲击较大，造成开发门槛很高，阻碍了该技术的发展与应用。

Apache Teaclave Java TEE SDK提供基于Intel SGX技术的Java生态机密计算开发框架。采用Java静态编译技术将Enclave Module编译成Native代码并运行在SGX TEE中，实现对高级语言的支持并最大限度保持较低的系统TCB。屏蔽底层交互细节，用户无须定义edl接口文件。给用户提供一个Pure Java的机密计算开发框架和编译构建工具链，极大降低Intel SGX的开发门槛。

2. 环境准备

构建SGX机密计算环境前，您可以通过cpuid检查SGX使能状态。

1、安装cpuid。

```
sudo yum install -y cpuid
```

2、检查SGX使能状态。

```
cpuid -1 -l 0x7 | grep SGX
```

说明 SGX被正确使能后，运行SGX程序还需要SGX驱动。

3、检查SGX驱动安装情况。

```
ls -l /dev/{sgx_enclave,sgx_provision}
```

3. 运行项目test/benchmark/sample

3.1 进入容器环境

目前Teaclave-Java-Tee-SDK支持两种容器环境ubuntu18.04和anolis8.6.

```
docker run -it --privileged --network host -v /dev/sgx_enclave:/dev/sgx/enclave -v /dev/sgx
_provision:/dev/sgx/provision teaclave/teaclave-java-tee-sdk:v0.1.0-ubuntu18.04 /bin/bash
```

或

```
docker run -it --privileged --network host -v /dev/sgx_enclave:/dev/sgx/enclave -v /dev/sgx
_provision:/dev/sgx/provision teaclave/teaclave-java-tee-sdk:v0.1.0-ubuntu18.04 /bin/bash
```

3.2 运行samples

```
cd /opt/javaenclave/samples
```

运行 HelloWorld Sample: `cd helloworld && ./run.sh`

OpenAnolis
龙 蜥 社 区

结果如下即代表运行成功：

```
Downloading ...........

[INFO]
[INFO] helloworld ........................................ SUCCESS [  0.758 s]
[INFO] common ............................................ SUCCESS [  5.766 s]
[INFO] enclave ........................................... SUCCESS [ 55.070 s]
[INFO] host .............................................. SUCCESS [ 25.544 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:27 min
[INFO] Finished at: 2023-02-16T12:27:18Z
[INFO] ------------------------------------------------------------------------
Hello World
Hello World
Hello World
Hello World
```

运行 Springboot Sample: `cd springboot && ./run.sh`

结果如下即代表运行成功：

```
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for springboot 1.0-SNAPSHOT:
[INFO]
[INFO] springboot ........................................ SUCCESS [ 22.343 s]
[INFO] common ............................................ SUCCESS [06:46 min]
[INFO] enclave ........................................... SUCCESS [02:52 min]
[INFO] host .............................................. SUCCESS [02:58 min]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14:38 min
[INFO] Finished at: 2023-02-16T11:28:49Z
[INFO] ------------------------------------------------------------------------


 .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
```

若因网络原因无法下载依赖，可通过配置Maven镜像源解决

3.3 运行tests

`cd /opt/javaenclave/test && ./run.sh`

```
...........

.enter test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave
exit test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave
.enter test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave
exit test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave
.enter test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave
exit test case: org.apache.teaclave.javasdk.test.host.TestSMEnclave

Time: 109.616

OK (16 tests)

Teaclave java sdk ut result: true
```

3.4 运行benchmark

`cd /opt/javaenclave/benchmark`

运行 Guomi Benchmark: `cd guomi && ./run.sh`

```
REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.


Benchmark                (enclaveServiceInstance)  (smAlgo)  Mode  Cnt   Score    Error  Units
GuoMiBenchMark.smBenchMark          MOCK_IN_JVM          SM2  avgt   4  39.032 ? 39.471  ms/op
GuoMiBenchMark.smBenchMark          MOCK_IN_JVM          SM3  avgt   4   8.656 ?  0.302  ms/op
GuoMiBenchMark.smBenchMark          MOCK_IN_JVM          SM4  avgt   4   3.410 ?  0.153  ms/op
GuoMiBenchMark.smBenchMark          MOCK_IN_SVM          SM2  avgt   4  31.899 ?  1.003  ms/op
GuoMiBenchMark.smBenchMark          MOCK_IN_SVM          SM3  avgt   4  10.755 ?  2.616  ms/op
GuoMiBenchMark.smBenchMark          MOCK_IN_SVM          SM4  avgt   4   4.515 ?  0.303  ms/op
GuoMiBenchMark.smBenchMark          TEE_SDK              SM2  avgt   4  36.113 ?  2.337  ms/op
GuoMiBenchMark.smBenchMark          TEE_SDK              SM3  avgt   4  11.331 ?  1.379  ms/op
GuoMiBenchMark.smBenchMark          TEE_SDK              SM4  avgt   4  10.292 ?  0.896  ms/op
GuoMiBenchMark.smBenchMark          EMBEDDED_LIB_OS      SM2  avgt   4  32.058 ? 14.033  ms/op
GuoMiBenchMark.smBenchMark          EMBEDDED_LIB_OS      SM3  avgt   4  10.380 ?  0.741  ms/op
GuoMiBenchMark.smBenchMark          EMBEDDED_LIB_OS      SM4  avgt   4   9.190 ?  0.488  ms/op

Benchmark result is saved to guomi_benchmark.json
```

运行 String Benchmark: `cd string && ./run.sh`

```
REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

Benchmark                    (enclaveServiceInstance)  (stringOpt)  Mode  Cnt  Score   Error  Units
StringBenchMark.stringBenchMark        MOCK_IN_JVM        regex    avgt   4  2.788 ? 0.090  ms/op
StringBenchMark.stringBenchMark        MOCK_IN_JVM        concat   avgt   4  2.692 ? 0.139  ms/op
StringBenchMark.stringBenchMark        MOCK_IN_JVM        split    avgt   4  2.698 ? 0.129  ms/op
StringBenchMark.stringBenchMark        MOCK_IN_SVM        regex    avgt   4  4.558 ? 0.084  ms/op
StringBenchMark.stringBenchMark        MOCK_IN_SVM        concat   avgt   4  4.879 ? 0.200  ms/op
StringBenchMark.stringBenchMark        MOCK_IN_SVM        split    avgt   4  5.595 ? 0.186  ms/op
StringBenchMark.stringBenchMark        TEE_SDK            regex    avgt   4  5.377 ? 0.176  ms/op
StringBenchMark.stringBenchMark        TEE_SDK            concat   avgt   4  5.269 ? 0.119  ms/op
StringBenchMark.stringBenchMark        TEE_SDK            split    avgt   4  6.171 ? 0.403  ms/op
StringBenchMark.stringBenchMark        EMBEDDED_LIB_OS    regex    avgt   4  4.104 ? 0.992  ms/op
StringBenchMark.stringBenchMark        EMBEDDED_LIB_OS    concat   avgt   4  6.140 ? 0.278  ms/op
StringBenchMark.stringBenchMark        EMBEDDED_LIB_OS    split    avgt   4  4.305 ? 0.585  ms/op

Benchmark result is saved to string_benchmark.json
```

4. HelloWorld Demo演示

4.1 进入容器环境

与《运行项目test/benchmark/sample》的进入容器环境步骤相同。

4.2 创建JavaEnclave工程框架

利用Teaclave-Java-Tee-SDK提供的脚手架创建JavaEnclave工程框架：

```
mvn archetype:generate -DgroupId=com.sample -DartifactId=helloworld -Darchetype-GroupId=org.apache.teaclave.javasdk -DarchetypeArtifactId=javaenclave-archetype -Darche-typeVersion=0.1.0 -DinteractiveMode=false
```

该工程包括三个子工程，分别是host、common和enclave。

4.3 定义服务接口(common)

在common子模块中定义服务接口：

```
cd helloworld/common/src/main/java/com/sample/
mkdir -p helloworld/common
```

创建Service.java，定义服务接口：

```
package com.sample.helloworld.common;

import org.apache.teaclave.javasdk.common.annotations.EnclaveService;

@EnclaveService
public interface Service {
    String sayHelloWorld();
}
```

4.4 服务接口实现(enclave)

在enclave子模块实现所定义的服务接口：

```
cd helloworld/enclave/src/main/java/com/sample/
mkdir -p helloworld/enclave
```

创建ServiceImpl.java，实现服务接口：

```
package com.sample.helloworld.enclave;

import com.sample.helloworld.common.Service;
import com.google.auto.service.AutoService;

@AutoService(Service.class)
public class ServiceImpl implements Service {
    @Override
    public String sayHelloWorld() {
        return "Hello World";
    }
}
```

4.5 管理服务实现(host)

在host子模块实现对enclave的管理与服务加载.

```
cd helloworld/host/src/main/java/com/sample/
mkdir -p helloworld/host
```

创建Main.java，实现机密计算服务管理：

```
package com.sample.helloworld.host;

import org.apache.teaclave.javasdk.host.Enclave;
import org.apache.teaclave.javasdk.host.EnclaveFactory;
import org.apache.teaclave.javasdk.host.EnclaveType;

import com.sample.helloworld.common.Service;

import java.util.Iterator;
```

```
public class Main {
    public static void main(String[] args) throws Exception {
        EnclaveType[] enclaveTypes = {
                EnclaveType.MOCK_IN_JVM,
                EnclaveType.MOCK_IN_SVM,
                EnclaveType.TEE_SDK};

        for (EnclaveType enclaveType : enclaveTypes) {
            Enclave enclave = EnclaveFactory.create(enclaveType);
            Iterator<Service> services = enclave.load(Service.class);
            System.out.println(services.next().sayHelloWorld());
            enclave.destroy();
        }
    }
}
```

4.6 编译和运行

回到工程根目录并编译工程: `mvn -Pnative clean package`

编译成功后, 运行该Demo: `$JAVA_HOME/bin/java -cp host/target/host-1.0-SNAPSHOT-jar-with-dependencies.jar:enclave/target/enclave-1.0-SNAPSHOT-jar-with-dependencies.jar com.sample.helloworld.host.Main`

```
[INFO] Building jar: /opt/temp/helloworld/host/target/host-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for helloworld 1.0-SNAPSHOT:
[INFO]
[INFO] helloworld ........................................ SUCCESS [  0.111 s]
[INFO] common ............................................ SUCCESS [ 14.584 s]
[INFO] enclave ........................................... SUCCESS [01:29 min]
[INFO] host .............................................. SUCCESS [ 51.027 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:35 min
[INFO] Finished at: 2023-02-16T12:01:09Z
[INFO] ------------------------------------------------------------------------
# $JAVA_HOME/bin/java -cp host/target/host-1.0-SNAPSHOT-jar-with-dependencies.jar:enclave/target/enclave-1.0-SNAPSHOT-jar-with-dependencies.jar com.sample.helloworld.host.Main
Hello World
Hello World
Hello World
```


RATS-TLS：跨机密计算平台的双向传输层安全协议

项目位置链接

https://github.com/inclavare-containers/rats-tls

技术自身介绍

背景

机密计算指使用基于硬件的可信执行环境（Trusted Execution Environment，TEE）对使用中的数据提供保护。 通过使用机密计算，我们现在能够针对 “使用中” 的数据提供保护。RATS-TLS是将对HW-TEE的远程证明和TLS通信协议绑定的一项创新性的项目， 有助于租户通过RATS-TLS验证正在运行的程序是在可信的环境中， 并和程序之间建立安全通道传输加密的数据。

问题&挑战

硬件可信执行环境中运行的应用都是可信的，包括TLS库，因此似乎不用对HW-TEE内的TLS库进行特别的考虑，但与HW-TEE通信的对端必须要验证工作负载是否运行在真实的HW-TEE中。常规的远程证明工程化实践只是对远程证明底层原语进行了封装，实际产出就是让通信双方能够安全可信地协商出共享secret，并将如何基于共享secret创建安全通信信道的工作留给了应用开发者。

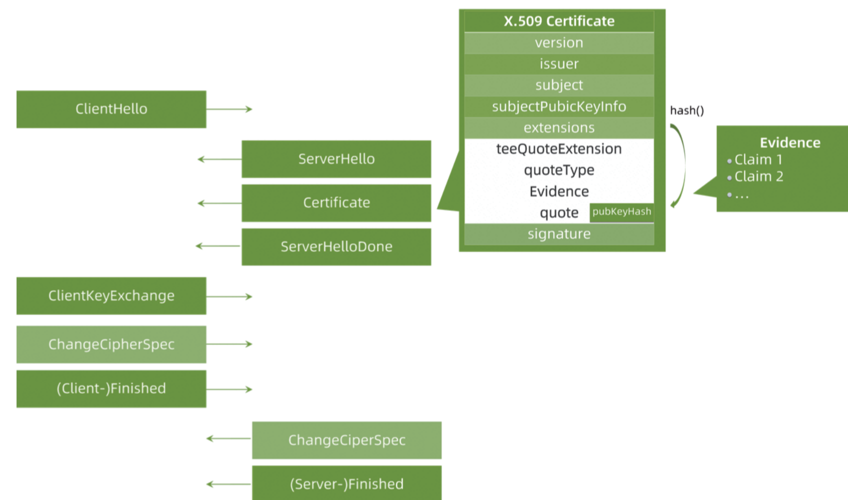
远程证明和建立安全信道的逻辑必须要深度结合，否则易受中间人攻击的影响。即使不将RA与TLS这么复杂的协议结合，至少也要能实现仅通过RA协议来协商出作为安全信道基础的共享密钥这一基本功能。但是仅做到共享秘密信息的程度是无法完全实现安全信道的。虽然基于共享秘密的方式能够实现安全信道，但是基于RA协议实现的安全信道的通信效率太低。解决方法是将RA与能够提供安全信道的标准协议相结合，比如TLS协议。

Thomas Knauth 提出将Intel SGX远程证明和TLS结合的方案，但是该方案面临着以下挑战：

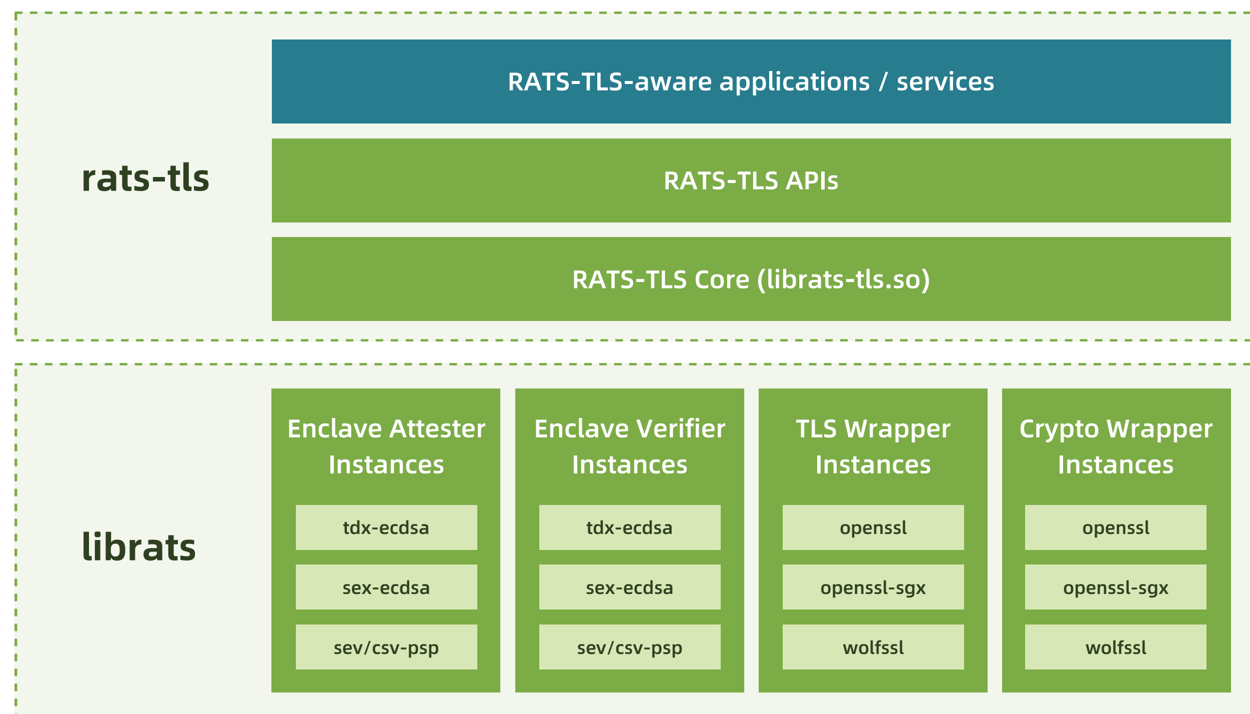
- 如何灵活地支持不同的TLS库？
- 如何适配不同的HW-TEE环境？
- 如何让两个不同类型的HW-TEE互相验证对方的Evidence？

解决方案

为了解决以上挑战，我们提出了Rats-TLS，一种支持异构硬件可执行环境的双向传输层安全性协议。如下图所示，Rats-TLS在TLS的基础上增加了将TLS证书中的公钥与HW-TEE attestation quote绑定的能力，基于HW-TEE硬件为信任根，即可证明了对方是在可信平台上，又可以高效的传输数据。



Rats-TLS的架构如下图所示，Rats TLS 提供了API给上层的应用和服务使用，API能够实现可信安全信道的建立和数据的传输。Rats TLS API的实现依赖于核心层与四类实例的实现。



为了提供更好的安全性，核心层与四类实例插件都运行在HW TEE环境中。由于Rats-TLS架构需要支持和灵活选择多种可能性（例如：TLS库，加密库，Enclave形态等），因此根据功能逻辑区分出不同的实例插件是必不可少的。下面简单介绍一下核心层和实例插件的作用。

- 核心层: 负责整体的控制作用，控制数据流的流向。
- TLS Wrapper实例: 负责完成真正的TLS session管理和网络传输。
- Attester 实例: 从本地平台运行环境中收集证明材料，通常要有Enclave配合生成quote数据。
- Verifier 实例: 负责验证收到的各种格式的quote数据，可能收到来自另一个不同的机密计算硬件平台发来的quote。也可能存在verifier与attester必须是相同平台的情况，比如verifier使用SGX ECDSA QVE的情况：强制要求当前运行环境必须支持SGX ECDSA，否则无法启动QVE来验证SGX ECDSA quote。
- Crypto实例: 负责配合其他实例完成与密码学算法有关的操作。例如，它可以生成自签名证书并将quote装到证书扩展中。

Rats-TLS具有以下优势:

- 能够支持不同的HW-TEE类型。
- 能够支持不同的TLS库。
- 能够支持不同的密码学算法库。
- 能够支持不同类型HW-TEE间的双向TLS认证。

OpenAnolis
龙 蜥 社 区

运行时底座

Runtime Foundation

CSV机密容器

介绍

Kata Containers 是一个使用虚拟化来提供隔离层的开源安全容器项目。Kata支持机密计算硬件技术，通过利用可信执行环境(Trusted Execution Environments)来保护客户的高度敏感的工作负载，以防止不受信任的实体（例如：云服务商）访问租客的敏感数据。

在kata container中，您能够在在机密虚拟机中运行POD和容器，将主机/owner/管理员/CSP软件栈从kata 的TCB中移除，从而构建一个更强大的云原生多租户架构。具体做法是：在为每个租户使用的容器加密镜像远程Provisioning解密密钥前，先认证pod或容器是否已经运行在了经过认证的环境中。

海光CPU支持安全虚拟化技术CSV(China Secure Virtualization)，CSV的设计目标是通过CSV虚拟机提供可信执行环境，适用的场景包括云计算、机密计算等。海光2号支持CSV1技术，提供虚拟机内存加密能力，采用国密SM4算法，不同的CSV虚拟机使用不同的加密密钥，密钥由海光安全处理器管理，主机无法解密虚拟机的加密内存。

海光CSV加密虚拟机支持两种远程认证方式：

- pre-attestation指需要在启动TEE之前能够执行attestation过程。在云上的CSV虚拟机启动的时候，对加载的ovmf，kernel，initrd，cmdline进行静态度量，生成measurement。线下的远程证明验证者对measurement进行验证，确保启动的CSV虚拟机是符合预期的。
- runtime-attestation：在云上的CSV虚拟机运行的时候，产生带有硬件可执行环境的Quote的TLS证书。线下的远程证明验证者对TLS证书进行验证，确保CSV虚拟机运行在TEE中。

基于runtime-attestation使用机密容器

本文主要为您介绍如何在kata环境中基于海光安全加密虚拟化功能CSV(China Secure Virtualization)技术，通过runtime-attestation 认证方式，启动一个租户的加密容器镜像。

前提条件

请使用安装Hygon CPU的硬件设备，硬件信息参考如下：

- CPU型号：Hygon C86 7291 32-core Processor
- 固件版本：1600及以上
- BIOS设置：开启SME

BIOS选项SMEE用来控制是否打开内存加密功能，SMEE=Enable表示在BIOS中打开内存加密功能，SMEE=Disable表示在BIOS中关闭内存加密功能。

1. 安装Anolis 8.4 操作系统

请参考Anolis 8.4 GA说明文档安装anolis 8.4 GA。

2. 升级kernel到5.10

Anlois 8.4 的默认内核版本是4.19，5.10的内核上支持CSV远程证明功能。请升级kernel 到5.10版本。

1、请参考以下命令，添加Anolis的Experimental repo，并将kernel升级至5.10。

```
yum-config-manager --add-repo https://mirrors.openanolis.cn/anolis/8/kernel-5.10/x
86_64/os/ && \
yum update kernel
```

2、配置bootloader。

```
grubby --update-kernel=ALL --args="mem_encrypt=on kvm_amd.sev=1"
```

3、重启机器，请输入以下命令查看内核版本

```
uname -r
```

预期输出：

```
5.10.134-12.an8.x86_64
```

注意！！

如果您使用的是Anolis 8.6 GA镜像，可能会碰到使能SEV之后，机器Hang住无法进入系统的情况。请参考以下步骤降级grub2-efi之后，可以正常启动这个特性

```
yum downgrade grub2-efi
```

3. 检查CSV使能状态

1、在操作系统内执行：

```
dmesg | grep -i sev
```

出现如下结果时，表示CSV已经使能。

```
[    9.179127  ] ccp 0000:05:00.2: sev enabled
[   10.297951] ccp 0000:05:00.2: SEV API:1.2 build:0
[   10.311454] SEV supported: 11 ASIDs
```

1、检查kvm_amd和ccp模块是否成功安装。

```
lsmod | grep kvm
```

4. 使用hag检查固件版本信息

1、安装hag

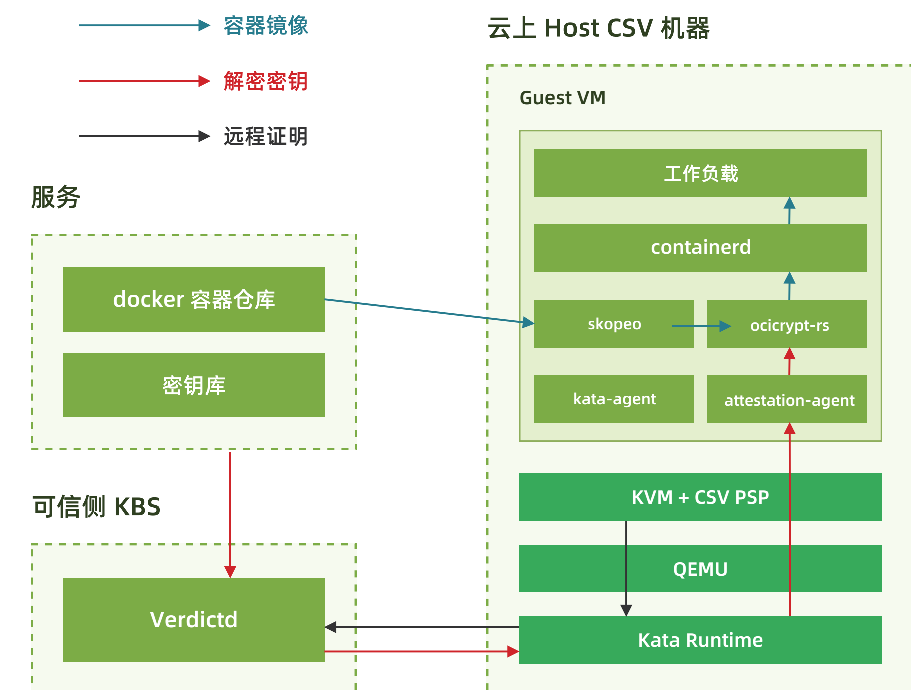
```
yum-config-manager --add-repo https://mirrors.openanolis.org/inclavare-containers/ano
lis8.4 && \
rpm --import https://mirrors.openanolis.org/inclavare-containers/anolis8.4/RPM-GPG-KEY
-rpm-sign && \
 yum install -y hag
```

1、通过hag获得平台状态

```
sudo hag --platform_status
api_major:              1
api_minor:              2
platform_state:        CSV_STATE_INIT
owner:                  PLATFORM_STATE_SELF_OWN
chip_secure:            SECURE
fw_enc:                 ENCRYPTED
fw_sign:                SIGNED
es:                     CSV ES
build id:               1600
bootloader version:     0.0.0
guest_count:            0
supported csv guest:  11
platform_status command successful
```

注意：固件build id要大于等于1600才可以支持远程证明。若版本低于1600，请联系BIOS厂商提供PI版本>=2.1.0.2的BIOS。

背景信息



1、CSV VM启动；

2、下载加密镜像时才会通过attestation-agent将通过vm-attestation hypercall获取的包括attestation-report 、chip-id等内容的CSV VM evidence发送给verdictd server校验；

3、校验通过后virdictd才与attestation-agent建立基于rats-tls的可信硬件环境的安全通道、并将加密镜像的解密key通过该安全通道发送给attestation-agent；

4、CSV VM利用步骤3获得的解密key解密镜像，运行工作负载

步骤一：配置权限景信息

1. 关闭firewall

Linux系统下面自带了防火墙iptables，iptables可以设置很多安全规则。但是如果配置错误很容易导致各种网络问题。此处建议关闭firewall。执行如下操作：

```
sudo service firewalld stop
```

2. 关闭selinux

Security-Enhanced Linux（SELinux）是一个在内核中强制访问控制（MAC）安全性机制。

为避免出现权限控制导致的虚拟机启动、访问失败等问题，此处建议关闭selinux。 执行如下操作：

```
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

步骤二：安装kata 环境

Kata Containers是一个开源的、致力于用轻量级虚拟机构建一个安全的容器运行时的实现，这些虚拟机在感觉和执行上与容器类似，但使用硬件虚拟化技术作为第二层防御，提供了更强的工作负载隔离。

关于项目的更多信息，请参见kata-container。

1. 安装kata-containers

1、请执行以下命令，安装kata-containers。

```
yum install -y kata-static
```

2、运行以下命令，查看kata-containers是否安装成功。

```
tree /opt/kata/
```

2. 安装qemu

此处使用的qemu基于6.2.0构建。

```
yum install -y qemu-system-x86_64
```

3. 安装guest kernel，initrd，ovmf

ccv0-guest中包含kata运行CSV VM所需的guest kernel、initrd、OVMF、cmdline等文件。 其中：guest的rootfs和kernel，需使用efi_secret的内核模块以支持向文件系统中注入secret，加入AA并修改AA设置，自行构建请参考guest Rootfs and Kernel； 这里提供的OVMF是基于f0f3f5aae7c4d346ea5e24970936d80dc5b60657 进行构建的，也可以使用edk2-stable202108后的版本自行构建，以支持CSV。

```
yum install -y ccv0-guest
```

cmdline中记录了CSV VM启动时所需的参数信息，需根据实际使用情况进行修改。可参考以下命令：

```
cat <<EOF | sudo tee /opt/csv/ccv0-guest/cmdline
tsc=reliable no_timer_check rcupdate.rcu_expedited=1 i8042.direct=1 i8042.dumbkbd=1
i8042.nopnp=1 i8042.noaux=1 noreplace-smp reboot=k console=hvc0 console=hvc1 crypto
mgr.notests net.ifnames=0 pci=lastbus=0 quiet panic=1 nr_cpus=`cat /proc/cpuinfo| grep
processor | wc -l` scsi_mod.scan=none agent.config_file=/etc/agent-config.toml
EOF
```

4. 安装kata-runtime

kata-runtime运行CSV VM。

```
yum -y install kata-runtime
```

5. 配置kata-runtime

执行以下命令，配置kata 运行时： 这里修改了kata-runtime默认配置中的qemu、guest kernel && initrd && OVMF路径； 使能confidential-guest选项并加入attestation-agent-config配置； 将默认内存大小由2048调整为8000； 将共享文件系统由"virtio-fs"调整为"virtio-9p"。

```
mkdir -p /etc/kata-containers/ && \
cp /opt/kata/share/defaults/kata-containers/configuration.toml /etc/kata-containers/ && \
cd /etc/kata-containers/ && \
sed -i 's/opt\/kata\/bin\/qemu-system-x86_64/opt\/qemu\/bin\/qemu-system-x86_64/' configura
tion.toml && \
sed -i 's/kata\/share\/kata-containers\/vmlinux.container/csv\/ccv0-guest\/vmlinuz-5.15.0-rc5+/'
configuration.toml && \
sed -i 's/image = \"\/opt\/kata\/share\/kata-containers\/kata-containers/initrd = \"\/opt\/csv\/c
cv0-guest\/initrd.run/' configuration.toml && \
sed -i 's/\# confidential_guest/confidential_guest/' configuration.toml && \
sed -i 's/kernel_params = \"\"/kernel_params = \"agent.config_file=\/etc\/agent-config.toml\"/' configu
ration.toml && \
sed -i 's/firmware = \"\"/firmware = \"\/opt\/csv\/ccv0-guest\/OVMF.fd\"/' configuration.toml && \
sed -i 's/default_memory = 2048/default_memory = 8000/' configuration.toml && \
sed -i 's/shared_fs = \"virtio-fs\"/shared_fs = \"virtio-9p\"/' configuration.toml && \
sed -i 's/\#service_offload/service_offload/' configuration.toml
```

步骤三：安装containerd

Containerd是一个行业标准的容器运行时，强调简单性、健壮性和可移植性。它可以作为Linux和Windows的守护进程，可以管理其主机系统的完整容器生命周期:图像传输和存储、容器执行和监督、底层存储和网络附件等。 更多信息请参考containerd

1、执行以下命令，安装containerd

```
sudo yum install -y containerd
```

2、启动containerd

```
sudo systemctl enable /etc/systemd/system/containerd.service
sudo systemctl daemon-reload
sudo service containerd restart
```

预期输出类似如下：

```
● containerd.service - containerd container runtime
  Loaded: loaded (/etc/systemd/system/containerd.service; disabled; vendor preset: disabled)
  Active: active (running) since Thu 2022-03-31 16:52:43 CST; 1s ago
    Docs: https://containerd.io
 Process: 1884520 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
Main PID: 1884522 (containerd)
   Tasks: 34
  Memory: 46.8M
  CGroup: /system.slice/containerd.service
          └─1884522 /usr/bin/containerd
```

步骤四：搭建kubernetes运行环境

请参考kubernetes官方指南安装Kubernetes cluster。搭建kubernetes运行环境。

步骤五：安装并启动verdictd

Verdictd是一种远程认证实现，由一组构建块组成，这些构建块利用Intel/AMD的安全特性来发现、验证和支持关键的基础安全和机密计算用例。它依靠RATS-TLS应用远程认证基础和标准规范来维护平台数据收集服务和高效的验证引擎来执行全面的信任评估。这些信任评估可用于管理应用于任何给定工作负载的不同信任和安全策略。 更多信息请参考verdictd项目文档。

1. 请执行以下命令，安装verdictd

```
yum install -y verdictd
```

2. 配置CSV OPA文件

获得measurement

CSV 机器容器启动的时候，要对kernel,intird,ovmf和cmdline进行度量，来确保云上启动的guest VM的确是符合预期的CSV VM。

默认情况下，请使用/opt/csv/calculate_hash.py计算measurement。

为确保/opt/csv/calculate_hash.py运行正常，请安装snowland-smx模块，以pip3为例：

```
pip3 install snowland-smx
```

安装完成后执行：

```
yum install -y gop

/opt/csv/calculate_hash.py --ovmf  /opt/csv/ccv0-guest/OVMF.fd --kernel /opt/csv/ccv0-guest/vmlinuz-5.15.0-rc5+  --initrd /opt/csv/ccv0-guest/initrd.run.img --cmdline/opt/csv/ccv0-guest/cmdline
```

输出结果类似如下：

```
Calculating hash of kernel at /opt/csv/ccv0-guest/vmlinuz-5.15.0-rc5+
Calculating hash of initrd at /opt/csv/ccv0-guest/initrd.run.img
Calculating hash of kernel params (/opt/csv/ccv0-guest/cmdline)
Firmware Digest:  OJXIhq3PHbknNmpAIy8YpUHOpY0wvGRXULOW8djVAZA=
```

注意：如果您修改了Kata的 配置文件（/etc/kata-containers/configuration.toml），可能会影响cmdline的内容。

请您用　ps -ef | grep qemu　输出qemu实际启动VM的命令参数，然后参考　-append　的参数获得guest VM的cmdline. 最后重新使用/opt/csv/calculate_hash.py脚本计算measurement.

例如：

```
# ps -ef
/opt/qemu/bin/qemu-system-x86_64
...
-kernel /opt/csv/ccv0-guest/vmlinuz-5.15.0-rc5+ \
-initrd /opt/csv/ccv0-guest/initrd.run.img \
-append tsc=reliable no_timer_check rcupdate.rcu_expedited=1 i8042.direct=1 i8042.dumbkbd=1 i8042.nopnp=1 i8042.noaux=1 noreplace-smp reboot=k console=hvc0 console=hvc1 cryptomgr.notests net.ifnames=0 pci=lastbus=0 debug panic=1 nr_cpus=96 scsi_mod.scan=none agent.log=debug agent.debug_console agent.debug_console_vport=1026 agent.config_file=/etc/agent-config.toml agent.log=debug initcall_debug \
-pidfile /run/vc/vm/0d134059d36e2c099363d0c48d176e18ae9133dcb4ce25094079cfc1fd5de3a5/pid
...
```

则guest VM 的cmdline 为

```
tsc=reliable no_timer_check rcupdate.rcu_expedited=1 i8042.direct=1 i8042.dumbkbd=1 i8042.nopnp=1 i8042.noaux=1 noreplace-smp reboot=k console=hvc0 console=hvc1 cryptomgr.notests net.ifnames=0 pci=lastbus=0 debug panic=1 nr_cpus=96 scsi_mod.scan=none agent.log=debug agent.debug_console agent.debug_console_vport=1026 agent.config_file=/etc/agent-config.toml agent.log=debug initcall_debug
```

配置csvData

```
mkdir -p /opt/verdictd/opa/ && cat <<EOF | sudo tee /opt/verdictd/opa/csvData
{
   "measure": ["OJXIhq3PHbknNmpAIy8YpUHOpY0wvGRXULOW8djVAZA="]
}
EOF
```

配置csvPolicy.rego

```
cat <<EOF | sudo tee /opt/verdictd/opa/csvPolicy.rego

package policy

# By default, deny requests.
default allow = false

allow {
    measure_is_grant
}


measure_is_grant {
    count(data.measure) == 0
}

measure_is_grant {
    count(data.measure) > 0
    input.measure == data.measure[_]
}
EOF
```

步骤六：制作加密镜像

1. 制作加密镜像

可参考Generate encrypted container image制作加密镜像。 在本例中以 docker.io/zhouliang121/alpine-84688df7-2c0c-40fa-956b-29d8e74d16c1-gcm:latest 为例进行测试。

注意事项：在机密计算场景中，加密镜像是在guest VM中由imgae-rs 组件负责拉取，而不是在host进行拉取。 如果您出于研究的目的，想查看加密镜像的内容。请注意由于镜像是加密的，用常规的 docker ，ctr 和 crictl 都无法正常拉取。请使用skopeo工具进行镜像的拉取。参考命令如下：

```
skopeo --insecure-policy copy docker://docker.io/zhouliang121/alpine-84688df7-2c0c-40fa-956b-29d8e74d16c1-gcm:latest oci:test
```

2. 部署镜像密钥

```
mkdir -p /opt/verdictd/keys/ && echo 1111111111111111111111111111111 > /opt/verdictd/keys/84688df7-2c0c-40fa-956b-29d8e74d16c1
```

3. 修改镜像policy

以docker.io/zhouliang121/alpine-84688df7-2c0c-40fa-956b-29d8e74d16c1-gcm:latest 为例：

```
cat <<EOF | sudo tee /opt/verdictd/image/policy.json
{
    "default": [{"type": "insecureAcceptAnything"}],
    "transports": {
        "docker": {
            "docker.io/zhouliang121/":
                [{"type": "insecureAcceptAnything"}]
            }
        }
}
EOF
```

4. 启动verdictd

```
verdictd --listen 0.0.0.0:20002 --verifier csv --attester nullattester --client-api 127.0.0.1:20001 --mutual
```

当Verdictd启动后，Verdictd在端口监听地址0.0.0.0:20002监听来自attestation agent的远程证明请求。

注意：verdictd 启动的时候有一个报错。原因是在注册SGX相关的 instance时出错，在CSV平台上可以忽略。

```
[ERROR] failed on dlopen(): libsgx_dcap_quoteverify.so.1: cannot open shared object file: No such file or directory
```

步骤七：部署加密镜像

1. 创建RuntimeClass对象kata

用户可以使用RuntimeClass为pod指定不同的运行时，这里使用kata作为验证时使用的运行时。

在集群中执行以下命令，创建RuntimeClass对象kata。

```
cat <<-EOF | kubectl apply -f -
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata
handler: kata
EOF
```

2. 部署pod

如果 pod 的 runtimeClassName 设置为 kata，CRI 插件会使用 Kata Containers 运行时运行 pod。 执行以下命令，部署名称为alpine的pod。

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-sandbox
spec:
  runtimeClassName: kata
  containers:
  - image: docker.io/zhouliang121/alpine-84688df7-2c0c-40fa-956b-29d8e74d16c1-gcm:latest
    command:
      - top
```

```
imagePullPolicy: IfNotPresent
name: alpine
restartPolicy: Never
EOF
```

3. 测试加密镜像是否部署成功

执行以下命令，查看加密镜像是否部署成功：

```
kubectl get pods
```

预期输出：

```
NAME            READY   STATUS    RESTARTS   AGE
nginx-sandbox   1/1     Running   0          30s
```

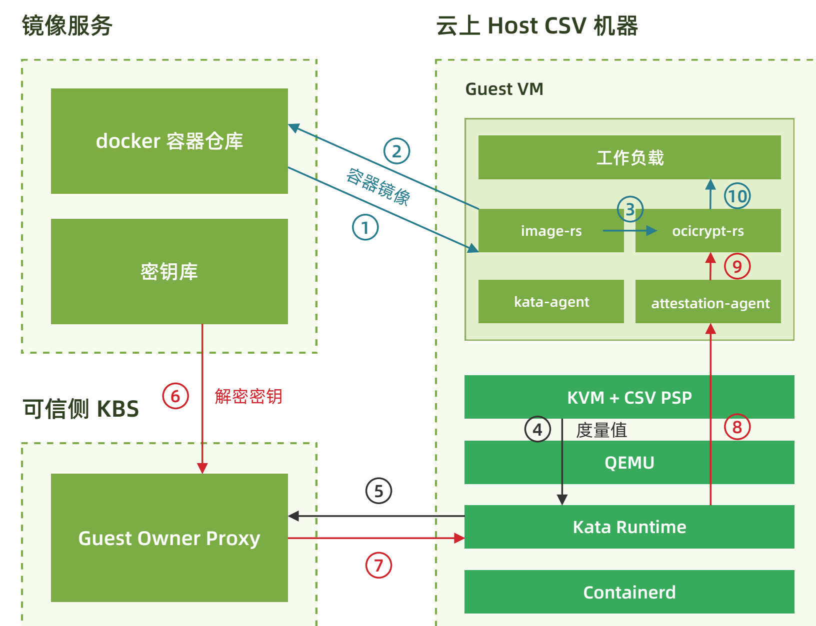
基于pre-attestation使用机密容器

本文主要为您介绍如何在kata环境中基于海光安全加密虚拟化功能CSV(China Secure Virtualization)技术，通过pre-attestaion 认证方式，启动一个租户的加密容器镜像。

前提条件

请参考《基于runtime-attestation使用机密容器》指南的前提条件一节，完成对系统环境的检查和配置。

背景信息



①②③：containerd 调用 kata-runtime创建CSV VM，kata-runtime调用 image-rs 下载加密镜像；

④⑤：kata-runtime 获取 CSV VM 的 launch_measurement发送给 gop-server；

⑥⑦：gop-server 调 用 hag 对 launch_measurement 计算和校验；验证 gop-server 调用 hag 对 key 进行加密、形成 secret 和 secret_header，发送给 kata-runtime

⑧：kata-runtime 调用 qemu 向 CSV VM 注入 secret；CSV VM 利用固件解密 secret 获取 key 存入指定位置；

⑨：CSV VM 中的 attestation-agent 获取 key，image-rs 使用 key 解密镜像；

⑩：启动镜像运行 workload

步骤一：配置权限

请参考《基于runtime-attestation使用机密容器》指南的步骤一，完成配置权限。

步骤二：安装kata 环境

Kata Containers是一个开源的、致力于用轻量级虚拟机构建一个安全的容器运行时的实现，这些虚拟机在感觉和执行上与容器类似，但使用硬件虚拟化技术作为第二层防御，提供了更强的工作负载隔离。

关于项目的更多信息，请参见kata-container。

1. 安装kata-containers

请执行以下命令，安装kata-containers。

```
yum-config-manager --add-repo https://mirrors.openanolis.org/inclavare-containers/anolis8.4 && \
 rpm --import https://mirrors.openanolis.org/inclavare-containers/anolis8.4/RPM-GPG-KEY-rpm-sign && \
 yum install -y kata-static
```

2. 安装qemu

pre-attestation过程中，CSV VM的measurement计算需要qemu支持kernel-hashes，在qemu 6.2.0之后开始支持kernel-hashes标志，此处提供的qemu是基于6.2.0版本构建。

```
yum install -y qemu-system-x86_64
```

3. 安装guest kernel，initrd，OVMF

ccv0-guest中包含kata运行CSV VM所需的guest kernel、initrd、OVMF、cmdline等文件。 其中：

guest的rootfs和kernel，需使用efi_secret的内核模块以支持向文件系统中注入secret，加入AA并修改AA设置，自行构建请参考guest Rootfs and Kernel；

这里提供的OVMF是基于f0f3f5aae7c4d346ea5e24970936d80dc5b60657 进行构建的，也可以使用edk2-stable202108后的版本自行构建，以支持CSV。

```
yum -y install ccv0-guest
```

cmdline中记录了CSV VM启动时所需的参数信息，需根据实际使用情况进行修改。可参考以下命令：

```
cat <<EOF | sudo tee /opt/csv/ccv0-guest/cmdline
tsc=reliable no_timer_check rcupdate.rcu_expedited=1 i8042.direct=1 i8042.dumbkbd=1
i8042.nopnp=1 i8042.noaux=1 noreplace-smp reboot=k console=hvc0 console=hvc1
cryptomgr.notests net.ifnames=0 pci=lastbus=0 quiet panic=1 nr_cpus=`cat /proc/cpuinfo| grep
processor | wc -l` scsi_mod.scan=none agent.config_file=/etc/agent-config.toml
EOF
```

4. 安装kata-runtime

pre-attestation使用的kata-runtime，需要基于confidential-containers 项目的 kata-containers-CCv0的CCv0分支，并加入CCv0: SEV prelaunch attestation by jimcadden · Pull Request #3025 · kata-containers/kata-containers · GitHub上的pre-attestation 的支持patch f682220d 和 d2c3f372； 同时也需要使用修改后的CPUID 以支持Hygon CPU。

```
yum -y install kata-runtime
```

5. 配置kata-runtime

执行以下命令，配置kata 运行时。

这里修改了kata-runtime默认配置中的qemu、guest kernel && initrd && OVMF路径；

使能confidential-guest选项并加入guest_attestation_proxy、guest_attestation_keyset、attestation-agent-config等配置；

将默认内存大小由2048调整为8000；

将共享文件系统由"virtio-fs"调整为"virtio-9p"。

```
mkdir -p /etc/kata-containers/ && \
cp /opt/kata/share/defaults/kata-containers/configuration.toml /etc/kata-containers/ && \
cd /etc/kata-containers/ && \
sed -i 's/opt\/kata\/bin\/qemu-system-x86_64/opt\/qemu\/bin\/qemu-system-x86_64/'
configuration.toml && \
sed -i 's/kata\/share\/kata-containers\/vmlinux.container/csv\/ccv0-guest\
/vmlinuz-5.15.0-rc5+/' configuration.toml && \
sed -i 's/image = \"\/opt\/kata\/share\/kata-containers\/kata-containers\/initrd =
\"\/opt\/csv\/ccv0-guest\/initrd.pre/' configuration.toml && \
sed -i 's/\# confidential_guest/confidential_guest/' configuration.toml && \
sed -i '/confidential_guest/a\guest_attestation = true\nguest_attestation_proxy = \"local
host:50051\"\nguest_attestation_keyset = \"KEYSET-1\"' configuration.toml && \
sed -i 's/kernel_params = \"\"/kernel_params = \"agent.config_file=\/etc\/agent-
config.toml\"/' configuration.toml && \
sed -i 's/firmware = \"\"/firmware = \"\/opt\/csv\/ccv0-guest\/OVMF.fd\"/'
configuration.toml && \
sed -i 's/default_memory = 2048/default_memory = 8000/' configuration.toml && \
sed -i 's/shared_fs = \"virtio-fs\"/shared_fs = \"virtio-9p\"/' configuration.toml && \
sed -i 's/\#service_offload/service_offload/' configuration.toml
```


步骤三：安装containerd

请参考《基于runtime-attestation使用机密容器》指南的步骤三来安装containerd。

步骤四：搭建运行环境

请参考kubernetes官方指南安装Kubernetes cluster。搭建kubenetes运行环境。

步骤五：安装并启动GOP KBS

GOP（guest owner proxy）用于验证CSV VM的启动，并有条件地提供一个secret，是一项非常复杂的操作。该工具设计用于在可信环境中运行，提供验证和sercret注入服务。

1. 请执行以下命令，安装GOP KBS

```
yum install -y gop
```

初始化：

```
cd /opt/csv/guest-owner-proxy
./make_grpc.sh
```

2. 安装hag

hag是海光提供的工具，主要用于建立和管理服务器的csv证书链，确保csv虚拟机在安全可靠的环境下执行，不受主机干扰。

```
yum install -y hag
```

3. 使用hag生成证书链

```
rm -rf /opt/csv/*cert /opt/csv/*txt /opt/csv/*bin /tmp/csv-guest-owner-proxy/ && \
 cd /opt/csv && \
 hag  --pdh_cert_export
```

4. 修改脚本适配本机环境

将keysets.json中的min-api-major、min-api-minor、allowed-build-ids分别改为hag --platform_status显示的api_major、api_minor、build_id

将gop-client.py中hw_api_major、hw_api_minor、hw_build_id分别改为hag --platform_status显示的api_major、api_minor、build_id

将gop-server.py中的build_id值改为与gop-client.py中hw_build_id相同

5. 启动GOP server

```
cd /opt/csv/guest-owner-proxy/ && \
 ./make_grpc.sh && \
 ./gop-server.py
```

步骤六：制作加密镜像

1. 制作加密镜像

请参考指南制作加密镜像。 在本例中可以使用 docker.io/haosanzi/busybox-v1:encrypted 作为镜像进行测试。

2. 将加密key保存到GOP的keys.json文件中

GOP的keys.json文件路径：`/opt/csv/guest-owner-proxy/keys.json` 。

步骤七：部署加密镜像

1. 创建RuntimeClass对象kata

用户可以使用RuntimeClass为pod指定不同的运行时，这里使用kata作为验证时使用的运行时。

在集群中执行以下命令，创建RuntimeClass对象kata。

```
cat <<-EOF | kubectl apply -f -
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata
handler: kata
EOF
```

2. 部署pod

用如果 pod 的 runtimeClassName 设置为 kata，CRI 插件会使用 Kata Containers 运行时运行 pod。

执行以下命令，部署名称为busybox的pod。

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-sandbox
spec:
  runtimeClassName: kata
  containers:
  - image: docker.io/haosanzi/busybox-v1:encrypted
    command:
      - top
    imagePullPolicy: IfNotPresent
    name: alphine-haosanzi
  restartPolicy: Never
EOF
```

3. 测试加密镜像是否部署成功

执行以下命令，查看加密镜像是否部署成功：

```
kubectl get pods
```

从上述输出信息可知已部署成功。

```
NAME            READY    STATUS     RESTARTS    AGE
nginx-sandbox   1/1      Running    0           50s
```

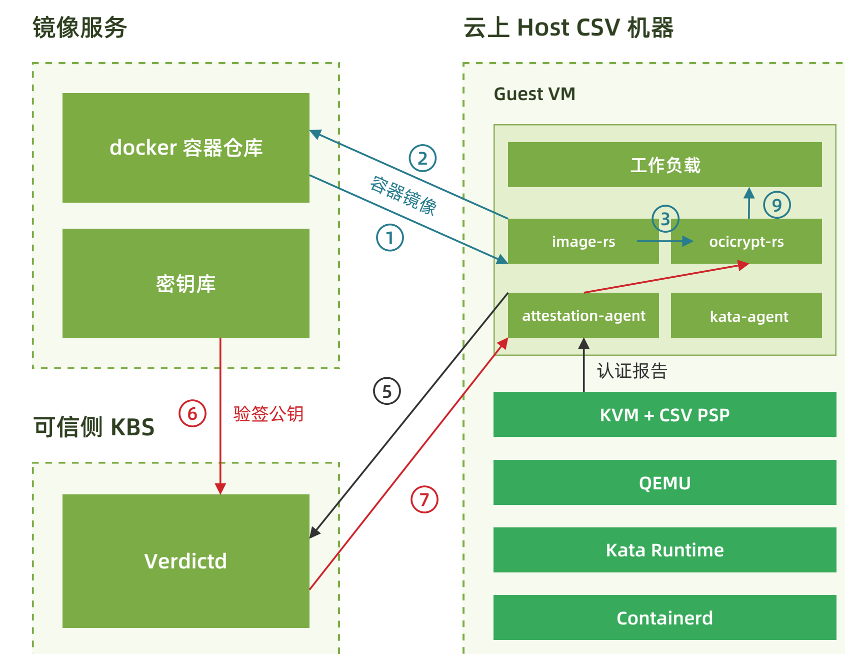
基于runtime-attestation使用签名容器

本文主要为您介绍如何在kata环境中基于海光安全加密虚拟化功能CSV(China Secure Virtualization)技术，启动一个租户的签名容器镜像。

前提条件

请参考《基于runtime-attestation使用机密容器》指南的前提条件一节，完成对系统环境的检查和配置。

背景信息



相较于Confidential Container v0，Confidential Container v1增加支持镜像签名和验签，并使用image-rs取代umoci和skopeo来完成镜像的相关管理功能。更多相关信息，请参考ccv1_image_security_design 。

①②③：containerd 调用 kata-runtime 启动 CSV VM；kata-runtime调用 image-rs 下载加密镜像；

④：attestation-agent 通过 vm-attestation hypercall 获取包括 attestation-report 、chip-id 等内容的 CSV VM evidence；

⑤：attestation-agent 发送 CSV VM evidence 给 verdictd server；verdictd 验证 CSV VM evidence 的证书链、签名、digest 等内容；与 attestation-agent 建立基于 rats-tls 的可信硬件环境的安全通道；

⑥⑦：attestation-agent 向verdictd server 请求验签 key和image policy，verdictd 作为 KBS 通过安全通道发送 policy 文件和验签key发送给 attestation-agent；

⑧：image-rs 使用 key 验签镜像；

⑨：启动镜像运行 workload

步骤一：配置权限

请参考《基于runtime-attestation使用机密容器》指南的步骤一，完成配置权限。

步骤二：制作签名的应用容器镜像

1.生成GPG密钥对

GPG：GNU Privacy Guard ， 用于加密、签名通信内容及管理非对称密钥等。 这里假设用户名为 testing ，邮箱为test@higon.cn，请按gpg2提示输入用户名、邮箱、密码等信息。

```
gpg2 --gen-key
```

上述执行完成后，执行 gpg2 --list-keys 列举生成的密钥： 可看见类似如下内容：

```
[root@localhost test]# gpg2 --list-keys
/root/.gnupg/pubring.kbx
------------------------
pub   rsa2048 2022-05-11 [SC] [expires: 2024-05-10]
      B0801A210472A10CC22090C8B27A004C4E80D3E5
uid           [ultimate] testing <test@hygon.cn>
sub   rsa2048 2022-05-11 [E] [expires: 2024-05-10]
```

由于verdictd不支持kbx格式pubkey，还需执行如下操作生成pubring.gpg

```
cd /root/.gnupg/
gpg2 --armor --export --output pubring.gpg test@hygon.cn
```

2. 制作签名的应用容器镜像

以docker.io/library/busybox:latest这个镜像为例，假设用户名为test，则远端的个人仓库为docker.io/test/busybox。

> 注意：在实际操作中，应将docker.io/test更名为实际操作的用户名，docker.io/xxxx。

首先配置 /etc/containers/registries.d/default.yaml 文件内容如下:

```
default-docker:
 sigstore-staging: file:///var/lib/containers/sigstore
```

然后用skopeo copy --sign-by进行签名， 以用户test为例：执行成功后在https://hub.docker.com/r/test/busybox/tags 上可观察到sig-test的tag。

```
skopeo login docker.io --username <$username> --password <$password>

skopeo copy --remove-signatures --sign-by test@hygon.cn \
 docker://docker.io/library/busybox:latest \
 docker://docker.io/test/busybox:sig-test

skopeo logout docker.io/test
```

以用户test为例：在本地可观察到 /var/lib/containers/sigstore 目录下多出了test目录（这样就完成了签名的准备工作）：

```
tree /var/lib/containers/sigstore

/var/lib/containers/sigstore
 └── test
    └── busybox@
sha256=52f431d980baa76878329b68ddb69cb124c25efa6e206d8b0bd797a828f0528e
        └── signature-1
```

步骤三：安装kata 环境

请参考《基于runtime-attestation使用机密容器》指南的步骤二来安装kata环境，并参考以下步骤更新initrd文件和kata 配置文件。

更新initrd文件

解压initrd image

```
cp initrd.run.img ./initrd.img.gz
gunzip initrd.img.gz
mkdir initrd
cd initrd/
cpio -ivmd < ../initrd.img
```

配置待验证镜像签名文件

将 /var/lib/containers/sigstore 目录下的所有内容 复制到initrd/var/lib/containers/sigstore：

```
sudo cp -rf /var/lib/containers/sigstore/* initrd/var/lib/containers/sigstore
```

重新打包initrd

```
cd initrd/
find . | cpio -o -H newc > ../initrd.new.img
gzip ../initrd.new.img
cd ../ && mv initrd.new.img.gz initrd.new.img
cp initrd.new.img initrd.run.img
```

步骤四：安装containerd

请参考《基于runtime-attestation使用机密容器》指南的步骤三来安装containerd。

步骤五：搭建kubenetes运行环境

请参考《基于runtime-attestation使用机密容器》指南的步骤四来搭建kubenetes运行环境。

步骤六： 安装并启动Verdictd

Verdictd是一种远程认证实现，由一组构建块组成，这些构建块利用Intel/AMD的安全特性来发现、验证和支持关键的基础安全和机密计算用例。它依靠RATS-TLS应用远程认证基础和标准规范来维护平台数据收集服务和高效的验证引擎来执行全面的信任评估。这些信任评估可用于管理应用于任何给定工作负载的不同信任和安全策略。 更多信息请参考verdictd项目文档。

1. 安装verdicted

```
yum remove verdictd -y
rm -rf /usr/share/rats-tls /usr/local/lib/rats-tls/ /usr/local/bin/verdict*
yum install -y verdictd
```

2. 配置verdicted

2.1 将步骤二中的pubring.gpg复制到 /opt/verdictd/gpg/

```
cp /root/.gnupg/pubring.gpg  /opt/verdictd/gpg/keyring.gpg
```

2.2 将policy.json写入到 /opt/verdictd/image/

> 在实际操作中，应将用户docker.io/test更名为实际操作的docker.io用户名，docker.io/xxxx。

```
cat>/opt/verdictd/image/policy.json <<EOF
{
  "default": [
   {
     "type": "reject"
   }
  ],
  "transports": {
   "docker": {
    "docker.io/test": [
     {
       "type": "signedBy",
       "keyType": "GPGKeys",
       "keyPath": "/run/image-security/simple_signing/pubkey.gpg"
     }
    ]
   }
  }
}
EOF
```

2.3 将sigstore.yaml写入到 /opt/verdictd/image/

```
cat>/opt/verdictd/image/sigstore.yaml <<EOF
default-docker:
 sigstore: file:///var/lib/containers/sigstore

docker:
docker.io/test:
  sigstore: file:///var/lib/containers/sigstore
EOF
```

2.4 配置CSV OPA文件

使用/opt/csv/calculate_hash.py计算measure：

```
yum install -y gop
/opt/csv/calculate_hash.py --ovmf  /opt/csv/ccv0-guest/OVMF.fd --kernel /opt/csv/ccv0
guest/vmlinuz-5.15.0-rc5+  --initrd /opt/csv/ccv0-guest/initrd.run.img --cmdlin/opt/csv/
ccv0-guest/cmdlinee
```

输出结果类似如下：

```
Calculating hash of kernel at /opt/csv/ccv0-guest/vmlinuz-5.15.0-rc5+
Calculating hash of initrd at /opt/csv/ccv0-guest/initrd.run.img
Calculating hash of kernel params (/opt/csv/ccv0-guest/cmdline)
Firmware Digest:  OJXIhq3PHbknNmpAIy8YpUHOpY0wvGRXULOW8djVAZA=
```

配置csvData，需确保csvData中的measure与前一步计算的Firmware Digest一致。

```
mkdir -p /opt/verdictd/opa/ && cat <<EOF | sudo tee /opt/verdictd/opa/csvData
{

   "measure": ["OJXIhq3PHbknNmpAIy8YpUHOpY0wvGRXULOW8djVAZA="]
}
EOF
```

配置csvPolicy.rego：

```
cat <<EOF | sudo tee /opt/verdictd/opa/csvPolicy.rego

package policy

# By default, deny requests.
default allow = false
```

```
allow {
   measure_is_grant
}


measure_is_grant {
   count(data.measure) == 0
}

measure_is_grant {
   count(data.measure) > 0
   input.measure == data.measure[_]
}
EOF
```

3. 启动verdictd

使用以下命令启动verdictd，从而于Attestation-Agent建立基于CSV的安全信道。

```
verdictd --listen 0.0.0.0:20002 --verifier csv --attester nullattester --client-api 127.0.0.1:20001
--mutual
```

注意：verdictd 启动的时候有一个报错。原因是在注册SGX相关的 instance时出错，在CSV平台上可以忽略。

```
[ERROR] failed on dlopen(): libsgx_dcap_quoteverify.so.1: cannot open shared object file: No
such file or directory
```

步骤七：启动并验证带签名的镜像

1. 创建RuntimeClass对象kata

用户可以使用RuntimeClass为pod指定不同的运行时，这里使用kata作为验证时使用的运行时。 在集群中执行以下命令，创建RuntimeClass对象kata。

```
cat <<-EOF | kubectl apply -f -
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata
handler: kata
EOF
```

2. 部署pod

如果 pod 的 runtimeClassName 设置为 kata，CRI 插件会使用 Kata Containers 运行时运行 pod。 执行以下命令，部署名称为sig-test的pod。

> 注意：在实际操作中，应将用户docker.io/test更名为实际操作的用户名，docker.io/xxxx。

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: sig-test
spec:
  runtimeClassName: kata
  containers:
  - image: docker.io/test/busybox:sig-test
    command:
      - top
    imagePullPolicy: IfNotPresent
    name: csv-sig-test
  restartPolicy: Never
EOF
```

3. 测试签名镜像是否部署成功

执行以下命令，查看加密镜像是否部署成功：

```
kubectl get pods
```

预期输出类似如下：

```
NAME       READY   STATUS    RESTARTS   AGE
sig-test   1/1     Running   0          23s
```

海光CSV机密虚拟机

本文主要为您介绍如何在 CSV裸金属/物理机环境中启动一个 CSV 虚拟机。

背景信息

海光CPU支持安全虚拟化技术CSV(China Secure Virtualization)，CSV的设计目标是通过CSV虚拟机提供可信执行环境，适用的场景包括云计算、机密计算等。海光2号支持CSV1技术，提供虚拟机内存加密能力，采用国密SM4算法，不同的CSV虚拟机使用不同的加密密钥，密钥由海光安全处理器管理，主机无法解密虚拟机的加密内存。

步骤一： 安装edk2-ovmf

请执行如下命令安装edk2-ovmf。OVMF是一个基于EDK II的项目，用于虚拟机的UEFI支持，更多详细信息请参考OVMF

```
yum install -y edk2-ovmf
```

步骤二：安装qemu-kvm

请执行以下命令安装 qemu-kvm。Qemu是一款开源的模拟器和虚拟机监视器，可用于虚拟机管理、加速，可以用来虚拟X86、Power、Arm、MIPS等平台，具有高速、跨平台的特性。qemu广泛应用于虚拟机管理、仿真等领域。更多详细信息请参考qemu

```
yum install -y qemu-kvm
```

步骤三：安装libvirt

请执行以下命令安装libvirt。libvirt是一套用于管理虚拟化的开源API、守护进程与管理工具。此套组件可用于管理KVM、Xen、VMware ESXi、QEMU及其他虚拟化技术。libvirt内置的API广泛用于云解决方案开发中的虚拟机监视器编排层（Orchestration Layer）。更多详细信息请参考libvirt

```
yum install -y libvirt
```

步骤四：下载Guest OS镜像

- 请参考以下命令从龙蜥社区下载 Anolis 8.6 的 Guest OS 镜像：

```
cd /tmp

wget https://anolis.oss-cn-hangzhou.aliyuncs.com/anolisos_8_6_x64_20G_anck_uefi_
ommunity_alibase_20220817.vhd
```

- 将vhd格式镜像转换为qcow2格式：

```
qemu-img convert -p -f vpc anolis_8_4_x64_20G_uefi_anck_community_alibase_20220418.
vhd-O qcow2 test.qcow2
```

步骤五：启动CSV guest虚拟机

启动方式一：使用 QEMU 命令行启动

请参考如下qemu命令行启动CSV guest虚拟机：

```
sudo /usr/libexec/qemu-kvm -enable-kvm -cpu host -smp 4 -m 4096 -drive if=pflash,
format=raw,unit=0,file=/usr/share/edk2/ovmf/OVMF_CODE.cc.fd,readonly=on  -hda
test.qcow2 -object sev-guest,id=sev0,policy=0x1,cbitpos=47,reduced-phys-bits=5 -machine
memory-encryption=sev0 -name test -monitor stdio
```

启动方式二：使用virsh启动

virsh 是用于管理 虚拟化环境中的客户机和 Hypervisor 的命令行工具，与 virt-manager 等工具类似，它也是通过 libvirt API 来实现虚拟化的管理。virsh 是完全在命令行文本模式下运行的用户态工具，它是系统管理员通过脚本程序实现虚拟化自动部署和管理的理想工具之一。

配置libvirt

请将/etc/libvirt/qemu.conf中的user和group设置为root，以免出现权限问题和报错：

```
442 # Some examples of valid values are:
443 #
444 #       user = "qemu"   # A user named "qemu"
445 #       user = "+0"     # Super user (uid=0)
446 #       user = "100"    # A user named "100" or a user with uid=100
447 #
448 user = "root"
449
450 # The group for QEMU processes run by the system instance. It can be
451 # specified in a similar way to user.
452 group = "root"
453
```

重启libvirtd服务

```
systemctl daemon-reload
service libvirtd restart
```

OpenAnolis 龙 蜥 社 区

创建CSV guest配置文件

以下是CSV虚拟机的参考配置文件csv_launch.xml，在使用过程中，请根据实际需求，修改对应的配置字段。更多配置请参考Launch security with AMD SEV

```
<domain type = 'kvm'  xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>csv_launch</name>
  <memory unit='GiB'>4</memory>
  <vcpu>4</vcpu>
  <os>
    <type arch = 'x86_64' machine = 'pc'>hvm</type>
    <boot dev = 'hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset = 'utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type = 'file' device = 'disk'>
      <driver name = 'qemu' type = 'qcow2' cache = 'none'/>
      <source file = '/tmp/test.qcow2'/>
      <target dev = 'hda' bus = 'ide'/>
    </disk>
    <memballoon model='none'/>
    <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0' keymap='en-us'>
      <listen type='address' address='0.0.0.0'/>
    </graphics>
  </devices>

  <launchSecurity type='sev'>
    <policy>0x0001</policy>
    <cbitpos>47</cbitpos>
    <reducedPhysBits>5</reducedPhysBits>
  </launchSecurity>

  <qemu:commandline>
    <qemu:arg value="-drive"/>
    <qemu:arg value="if=pflash,format=raw,unit=0,file=/usr/share/edk2/ovmf
/OVMF_CODE.cc.fd,readonly=on"/>
  </qemu:commandline>
</domain>
```

启动CSV虚拟机

```
sudo virsh create csv_launch.xml
```

步骤六：检查guest的CSV使能状态

- 请使用vnc或其他远程工具连接guest
- anolis镜像默认用户名anuser，密码anolisos

```
localhost login: anuser
Password: anolisos
```

登录虚拟机后，执行：

```
dmesg | grep -i sev
```

显示内容应类似如下，则证明CSV虚拟机启动成功：

```
[    0.129692] AMD Secure Encrypted Virtualization (SEV) active
[    1.886794] software IO TLB: SEV is active and system is using DMA bounce buffers
```

OpenAnolis
龙 蜥 社 区

Intel TDX机密容器

本文主要为您介绍如何基于Intel信任域(TD)的硬件隔离虚拟化功能TDX(Intel Trust Domain Extensions)（*https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html*）技术，通过远程证明启动一个租户的加密签名容器镜像。

前提条件

1. 安装Anolis 8.6 操作系统

请在支持INTEL TDX CPU的硬件设备上，参考Anolis 8.6 GA说明文档安装anolis 8.6 GA。

2. 升级内核到5.10

由于 Anlois 8.6 的默认内核版本是4.19，请升级kernel 到5.10版本。

• 添加 yum 源配置参数，添加Anolis 的 Experimental repo。

```
yum install yum-utils
yum-config-manager --add-repo https://mirrors.openanolis.cn/anolis/8/kernel-5.10/x86_64
/os/
```

• 升级内核

```
yum update kernel
```

• 重启机器，并重新查看机器的操作系统发行编号。

```
reboot
uname -r
```

• 预期结果如下：

```
5.10.134-13_rc2.an8.x86_64
```

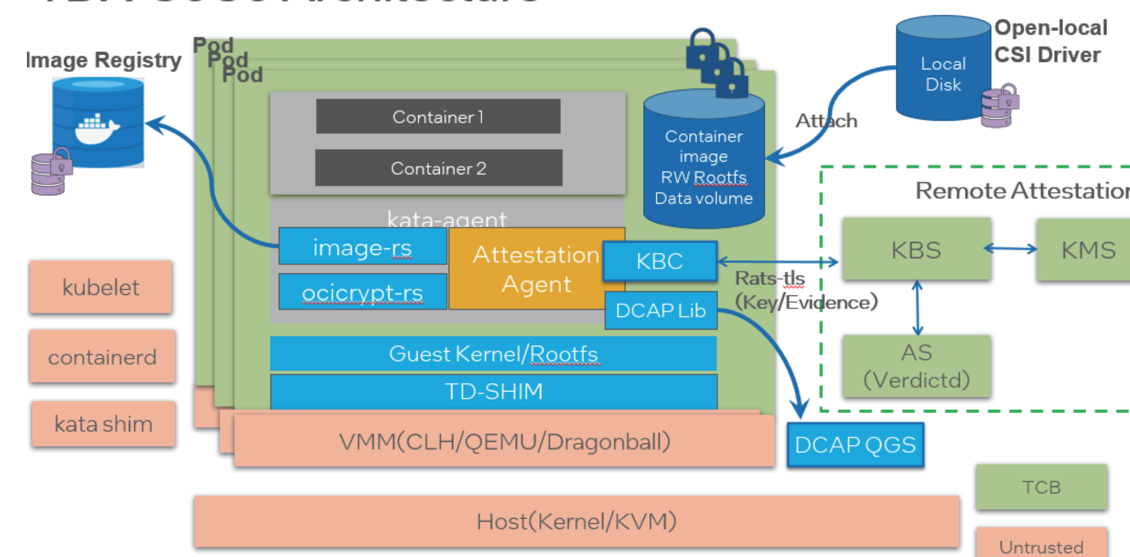
3. 使能TDX

• 重启后，请检查机器的tdx使能状态。

• 预期结果如下

```
[    1.481148] seam: Loading TDX P-SEAMLDR intel-seam/np-seamldr.acm.
[    1.482177] seam: Successfully loaded TDX P-SEAMLDR.
[    1.482195] tdx: Build all system memory blocks as TDX memory.
[    5.919528] tdx: Loaded TDX module via P-SEAMLDR.
[    5.919763] tdx: TDX SEAM module: attributes 0x0 vendor_id 0x8086 build_date 20220420
build_num 0x156 minor_version 0x0 major_version 0x1.
[    6.379342] tdx: Successfully initialized TDX module
```

背景信息



英特尔 TDX Pod 级机密容是将TDX硬件安全虚拟化技术同容器生态无缝集成，以云原生方式运行，保护敏感工作负载和数据的机密性和完整性。是目前机密容器社区（https://github.com/confidential-containers）主要支持的底层硬件之一。相对于传统基于虚拟机的容器技术，机密容器做了很多安全的增强，比如trusted boot, restricted API, encrypted/signed image, remote attestation，这些技术都集成为机密容器runtime stack的一部分。在TEE保护的guest 内部，默认集成了 image-rs 和 attestation-agent 等组件，它们负责实现容器镜像的拉取、授权、验签、解密、远程证明以及秘密注入等安全特性。机密容器的基本运行过程为：

• 用户使用标准工具制作一个签名和/或加密的受保护的容器镜像，并上传到容器镜像仓库中。

• 用户命令 Kubernetes 启动这个受保护的容器镜像。kubelet 会向 containerd 发起创建 Pod 的 CRI 请求，containerd 则把请求转发给 kata-runtime，kata-runtime 启动一个TDX 保护的轻量级VM, 目前支持TDX 的VMM 有Dragonball，Cloud Hypervisor, Qemu。

• kubelet 向 containerd 发起 Image Pulling 的 CRI 请求，containerd 则把请求转发给 kata-runtime，最终 kata-agent 收到请求并通过 image-rs 子模块提供的容器镜像管理功能，在下载image之前，image-rs 会跟attestation-agent进行远程认证。

• Attestation-agent 与 Key broker service（verdictd）建立安全会话，并进行基于runtime stack启动过程中的度量值进行远程认证，确保runtime没有被篡改。只有通过远程认证，image解密key，签名的policy和证书才会传送到TEE保护的guest内部。

• 通过远程证明后，拿到image policy文件，签名证书，解密image key，image-rs进行验签、解密、unpack 以及挂载容器镜像的操作，最终顺利启动container。

步骤一：部署测试集群

本步骤为您提供快速部署单节点测试集群的步骤。您可以根据您的需求，灵活部署集群。

配置权限

关闭firewall

Linux系统下面自带了防火墙 iptables ， iptables 可以设置很多安全规则。但是如果配置错误很容易导致各种网络问题。此处建议关闭 firewall 。 执行如下操作：

```
sudo service firewalld stop
```

检查 firewall 状态：

```
service firewalld status
```

预期结果如下：

```
Redirecting to /bin/systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
  Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
  Active: inactive (dead)
    Docs: man:firewalld(1)
```

关闭selinux

Security-Enhanced Linux（SELinux）是一个在内核中实施的强制存取控制（MAC）安全性机制。为避免出现权限控制导致的虚拟机启动、访问失败等问题，此处建议关闭selinux。执行如下操作：

```
setenforce 0
```

预期结果如下：

```
setenforce: SELinux is disabled
```

配置containerd

自动生成默认的config.toml

```
containerd config default > /etc/containerd/config.toml
```

由于默认的 config.toml 使用的是国外的镜像，国内有可能无法访问。请参考以下命令修改为国内镜像。

```
cd /etc/containerd
sed -i 's#registry.k8s.io/pause:3.6#registry.cn-hangzhou.aliyuncs.com/google_containers/pause:3.1#g' config.toml
```

启动 containerd

```
systemctl containerd start
```

部署单节点的Kubernetes cluster

- 请参考kubernetes官方指南安装Kubernetes cluster。最低 Kubernetes 版本应为 1.24。
- 确保集群中至少有一个 Kubernetes 节点具有标签 node-role.kubernetes.io/worker=

```
kubectl label node <node-name> node-role.kubernetes.io/worker=
```


步骤二：安装Confidential computing Operator

Confidential computing Operator 提供了一种在 Kubernetes 集群上部署和管理 Confidential Containers Runtime 的方法。具体信息请参考指南。

前提条件

1、确保 Kubernetes 集群节点至少有 8GB RAM 和 4 个 vCPU

2、当前 CoCo 版本仅支持基于 containerd 运行时的 Kubernetes 集群

3、确保 SELinux 被禁用或未强制执行 (confidential-containers/operator#115)

部署Operator

Operator目前有3个版本，这里默认安装最新版v0.3.0版本。 通过运行以下命令部署Operator：

```
kubectl apply -k github.com/confidential-containers/operator/config/release?ref=v0.3.0
```

cc-operator-controller-manager 资源依赖国外的镜像，可能拉不下来，请参考以下步骤对镜像进行修改：

```
kubectl edit deploy cc-operator-controller-manager -n confidential-containers-system

# 将gcr.io/kubebuilder/kube-rbac-proxy:v0.13.0替换成
image: quay.io/brancz/kube-rbac-proxy:v0.13.0
```

查看节点状态：

```
kubectl get pods -n confidential-containers-system --watch
```

预期结果如下。注意这三个pod都要存在，且STATUS都要为Running。

```
NAME                                              READY   STATUS    RESTARTS   AGE
cc-operator-controller-manager-56cb4d5ff5-lqd9x   2/2     Running   0          167m
cc-operator-daemon-install-rg8s9                  1/1     Running   0          154m
cc-operator-pre-install-daemon-7jhnw              1/1     Running   0          154m
```

创建custom resource

创建 custom resource 会将所需的 CC runtime安装到集群节点中并创建 RuntimeClasses。操作如下：

```
kubectl apply -k github.com/confidential-containers/operator/config/samples/ccruntime/default?ref=v0.3.0
```

检查创建的 RuntimeClasses。

```
kubectl get runtimeclass
```

预期结果如下：

```
NAME        HANDLER     AGE
kata        kata        154m
kata-clh    kata-clh    154m
```

```
kata-clh-tdx      kata-clh-tdx      154m
kata-qemu         kata-qemu         154m
kata-qemu-sev     kata-qemu-sev     154m
kata-qemu-tdx     kata-qemu-tdx     154m
```

卸载Operator（非必要步骤）

如果您想更新Operator的版本，或者您的安装出现问题，可以先卸载，再回到上面的步骤重新安装。具体操作请参考链接。

```
kubectl delete -k github.com/confidential-containers/operator/config/samples/ccruntime/default?ref=<RELEASE_VERSION>
kubectl delete -k github.com/confidential-containers/operator/config/release?ref=${RELEASE_VERSION}
```

步骤三：安装并启动verdictd

Verdictd是一种远程认证实现，由一组构建块组成，这些构建块利用Intel/AMD的安全特性来发现、验证和支持关键的基础安全和机密计算用例。它依靠RATS-TLS应用远程认证基础和标准规范来维护平台数据收集服务和高效的验证引擎来执行全面的信任评估。这些信任评估可用于管理应用于任何给定工作负载的不同信任和安全策略。 更多信息请参考verdictd项目文档。

1. 请执行以下命令，安装verdictd

```
yum install -y verdictd
```

2. 部署镜像加密密钥

```
mkdir -p /opt/verdictd/keys/

cat <<- EOF > /opt/verdictd/keys/84688df7-2c0c-40fa-956b-29d8e74d16c0
12345678901234567890123456789
EOF
```

3. 部署镜像签名密钥

```
# 安装镜像签名工具cosign：https://github.com/sigstore/cosign#installation
wget https://github.com/sigstore/cosign/releases/download/v2.0.0/cosign-linux-amd64
sudo install -D --owner root --group root --mode 0755 cosign-linux-amd64 /usr/local/bin/cosign

# 生成新的密钥对
cosign generate-key-pair
ls
cosign.key  cosign.pub
mkdir -p /opt/verdictd/image/
cp cosign.pub /opt/verdictd/image/cosign.key
```

4. 部署镜像policy

注意：在实际操作中，应将用户docker.io/test更名为实际操作的用户名，docker.io/xxxx。

```
cat <<EOF | sudo tee /opt/verdictd/image/policy.json
{
  "default": [
    {
      "type": " insecureAcceptAnything"
    }
  ],
"transports": {
"docker": {
      " docker.io/test/": [
        {
          "type": "sigstoreSigned",
          "keyPath": "/run/image-security/cosign/cosign.pub"
        }
      ]
    }
  }
}
EOF
```

5. 启动verdictd

```
verdictd --listen 0.0.0.0:20002 --verifier tdx --attester nullattester --client-api 127.0.0.1:20001
--mutual
```

当Verdictd启动后，Verdictd在端口监听地址0.0.0.0:20002监听来自attestationagent的远程证明请求。

6. 制作加密image

可参考Generate encrypted container image制作加密镜像。

注意事项：在机密计算场景中，加密镜像是在guest VM中由imgae-rs 组件负责拉取，而不是在host进行拉取。 如果您出于研究的目的，想查看加密镜像的内容。请注意由于镜像是加密的，用常规的docker，ctr 和 crictl 都无法正常拉取。请使用skopeo工具进行镜像的拉取。参考命令如下：

```
# 下载一个明文image
skopeo copy docker://docker.io/library/alpine:latest oci:alpine

# 为skopeo生成一个keyprovider配置文件
$ sudo mkdir -p /etc/containerd/ocicrypt/
$ cat <<- EOF | sudo tee "/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf"
{
    "key-providers": {
        "attestation-agent": {
          "grpc": "127.0.0.1:20001"

        }
    }
```

```
}
EOF

export OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.conf

# 生成加密image并保存在远端docker registry
```

注意：在实际操作中，应将用户docker.io/test更名为实际操作的用户名，docker.io/xxxx。

```
skopeo copy --encryption-key provider:attestation-agent:84688df7-2c0c-40fa-956b-29d8e74d16c0 oci:
alpine docker://docker.io/test/alpine-encrypted
{map[attestation-agent:{<nil> 127.0.0.1:50001}]}
&{map[attestation-agent:{<nil> 127.0.0.1:50001}]}
[[97 116 116 101 115 116 97 116 105 111 110 45 97 103 101 110 116 58 56 52 54 56 56 100 102 55 45 50 99 48 99 45 52 48 102 97 45 57 53 54 98 45 50 57 100 56 101 55 52 100 49 54 99 48]]
attestation-agent:84688df7-2c0c-40fa-956b-29d8e74d16c0
idx: 17
map[attestation-agent:[[56 52 54 56 56 100 102 55 45 50 99 48 99 45 52 48 102 97 45 57 53 54 98 45 50 57 100 56 101 55 52 100 49 54 99 48]]]
Getting image source signatures
&{map[attestation-agent:[[56 52 54 56 56 100 102 55 45 50 99 48 99 45 52 48 102 97 45 57 53 54 98 45 50 57 100 56 101 55 52 100 49 54 99 48]]] {map[]}}
Copying blob 63b65145d645 done
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures

# verdictd 日志
[2023-02-27T06:02:18Z INFO  verdictd::client_api::key_provider] wrap_command: KeyProviderInput { op: "keywrap", keywrapparams: KeyWrapParams { ec: Some(Ec { Parameters: {"attestation-agent": ["ODQ2ODhkZjctMmMwYy00MGZhLTk1NmItMjlkOGU3NGQxNmMw"]}, DecryptConfig: Dc { Parameters: {} } }), optsdata: Some("eyJzeW1rZXkiOiIycEVxWk9jNVhyUmN0WXdyQzl1UlJmSkZ5WGM2ZnV2SWZUckhnMHEyM0RrPSIsImRpZ2VzdCI6InNoYTI1Njo2M2I2NTE0NWQ2NDVjMTI1MGMzOTFiMmQxNmViZTUzYjM3NDdjMjk1Y2E4YmEyZmNiNmIwY2YwNjRhNGRjMjFjIiwiY2lwaGVyb3B0aW9ucyI6eyJub25jZSI6IkNURk5UZ2hZL0pkRkd0eGNKYzc5dUE9PSJ9fQ==") }, keyunwrapparams: KeyUnwrapParams{ dc: None, annotation: None } }
[2023-02-27T06:02:18Z INFO  verdictd::resources::directory_key_manager] get key from keyFile: /opt/verdictd/keys/84688df7-2c0c-40fa-956b-29d8e74d16c0
[2023-02-27T06:02:18Z INFO  verdictd::client_api::key_provider] key: [49, 50, 51, 52, 53, 54, 55, 56, 57,48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 10]
```

7、制作签名image

```
# 下载一个明文image
cosign sign --key cosign.key docker.io/test/alpine-encrypted
tlog entry created with index: 14409560
Pushing signature to: docker.io/test/alpine-encrypted
```


步骤四：启动并验证带签名的加密镜像

1. 配置TDX CoCo runtime

attestation agent 支持TDX平台的KBC为：eaa_kbc， 远端verdictd service用于提供验证和image解密key，签名的policy和密钥信息。

```
vim /opt/confidential-containers/share/defaults/kata-containers/configuration-qemu-tdx.toml
# EAA KBC is specified as: eaa_kbc::host_ip:port
kernel_params = "<default kernel params> agent.aa_kbc_params=eaa_kbc::verdictd_ip_address:20002
agent.enable_signature_verification=true"
```

2. 部署Pod

注意：在实际操作中，应将用户docker.io/test更名为实际操作的用户名，docker.io/xxxx。

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: test-tdx-alpine
spec:
  runtimeClassName: kata-qemu-tdx
  containers:
  - image: docker.io/test/alpine-encrypted
    command:
      - top
    imagePullPolicy: Always
    name: test-tdx-alpine
  restartPolicy: Never
EOF
```

• 查看 pod 是否启动成功：

```
kubectl get po
```

• 预期结果如下：

```
NAME              READY   STATUS    RESTARTS   AGE
test-tdx-alpine   1/1     Running   0          31h

kubectl describe pod test-tdx-alpine
  Normal  Pulling    5s    kubelet          Pulling image " docker.io/test/alpine-encrypted "
  Normal  Pulled     3s    kubelet          Successfully pulled image " docker.io/test/alpine-encrypted " in 1.682344015s (1.682349283s including waiting)
  Normal  Created    3s    kubelet          Created container test-tdx-alpine
  Normal  Started    3s    kubelet          Started container test-tdx-alpine
.
# verdictd 日志
```

```
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::rats_tls] response: {"data":{"base64
size":"452"},"status":"OK"}
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::protocol] Request: Object {"command":
String("Get Policy"), "optional": Object {}}
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::rats_tls] response: ewogICAgImRlZmF
1bHQiOiBbCiAgICAgICAgewogICAgICAgICAgICAidHlwZSI6ICJyZWplY3QiCiAgICAgICAgfQogICAgXSw
KICAgICJ0cmFuc3BvcnRzIjogewogICAgICAgICJkb2NrZXIiOiB7CiAgICAgICAgICAgICJyZWdpc3RyeS5kb
21haW4ubG9jYWwiOiBbCiAgICAgICAgICAgICAgICB7CiAgICAgICAgICAgICAgICAgICAgInR5cGUiOiAic2
lnc3RvcmVTaWduZWQiLAogICAgICAgICAgICAgICAgICAgICJrZXlQYXRoIjogIi9ydW4vaW1hZ2Utc2VjdXJ
pdHkvY29zaWduL2Nvc2lnbi5wdWIiCiAgICAgICAgICAgICAgICB9CiAgICAgICAgICAgIF0KICAgICAgICB9
CiAgICB9Cn0K
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::protocol] Request: Object {"command":
String("Get Resource Info"), "name": String("Cosign Key")}
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::rats_tls] response: {"data":{"base64
size":"240"},"status":"OK"}
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::protocol] Request: Object {"command":
String("Get Cosign Key"), "optional": Object {}}
[2023-03-01T08:19:45Z INFO  verdictd::attestation_agent::rats_tls] response: LS0tLS1CRUdJTiBQVU
JMSUMgS0VZLS0tLS0KTUZrd0V3WUhLb1pJemowQ0FRWUlLb1pJemowRE
FRY0RRZ0FFZ3h6NWhEVXl6VnpFd2RVcnhZb1JQVE1pN0ZveQovVEI4OTVlbmt
MdzE4RHNLczR1MnFidHg1L1hJNVlKaUJ4TDhyZG9NL3A5clBQSHVDVV
dpSkxBSFVnPT0KLS0tLS1FTkQgUFVCTElDIEtFWS0tLS0tCg==EOF
```


OpenAnolis 龙 蜥 社 区

Intel vSGX：Intel SGX虚拟化

项目位置链接

https://gitee.com/anolis/cloud-kernel.git

技术自身介绍

背景

SGX 是Intel基于硬件TEE的技术实现方式之一，硬件提供飞地 (Enclave) 的构建能力，使运行在飞地中的用户代码数据具备加密性和完整性保护的特点。同时通过远程证明的自证手段，为用户提供SGX 飞地运行环境和软件组件的可信度。SGX作为成熟的硬件TEE技术，为联邦学习，多方计算，区块链，安全数据库等诸多场景中提供技术底座的支撑。

问题&挑战

目前云服务提供商售卖的计算资源以虚拟机，安全容器的形式最为常见。硬件的能力通过虚拟化的方式暴露给虚拟机和安全容器，以SGX为例，如何通过虚拟化的方式，能够在虚拟机和安全容器中透传SGX的硬件能力，让联邦学习，多方计算，区块链等业务形态方便易用的使用具备SGX功能的云上计算资源。

解决方案

SGX的能力主要针对用于飞地的专用内存EPC (Enclave Page Cache) 进行加密和完成行保护来实现。SGX的虚拟化实现一下三方面技术支撑：

1、由虚拟机的CPUID 来枚举SGX的能力

2、通过为虚拟机模拟构建EPC的区域使虚拟机感知SGX EPC

3、捕捉虚拟机中SGX飞地构建的特殊指令，提供SGX飞地中附加功能的支持

SGX虚拟化的支持已在Linux上游社区和龙蜥的5.10内核中支持。

用户情况

后续计划

SGX 2.0 中的EDMM 功能提供动态得为SGX 飞地增加EPC资源的能力，在后续的龙蜥版本中，通过龙蜥机密计算SIG，为用户提供该功能。

Intel SGX虚拟机最佳实践

本文主要为您介绍如何在 SGX 裸金属/物理机环境中启动一个 SGX 虚拟机，以及如何在 SGX 虚拟机中构建 SGX 加密计算环境，并演示如何运行示例代码以验证 SGX 功能。

背景信息

Intel® SGX 是一组用于内存访问的指令和机制，以便为敏感应用程序和数据提供安全访问。 SGX 允许应用程序将其特定的地址空间用作 enclave，这是一个受保护的区域，即使在存在特权恶意软件的情况下也能提供机密性和完整性。 阻止任何不驻留在 enclave 中的软件访问 enclave 内存区域，包括来自特权软件的访问。

用户可以使用 Linux 内核中 KVM 虚拟化模块和 QEMU 在支持英特尔 SGX 的硬件上创建一个虚拟机，该虚拟机可以在 Guest 操作系统中使用Intel SGX and Intel SGX Data Center Attestation Primitives (Intel SGX DCAP) 。

前提条件

1. 检查 BIOS 是否使能 SGX

请使用 BIOS 使能 SGX 功能的 SGX 裸金属。

以浪潮 NF5280M6 SGX 机型为例，请参考下图的步骤检查 SGX 是否使能。

1.1. Total Memory Encription

Main menu->Socket Configuration->Processor Configuration->Total Memory Encryption ->Enabled

1.2. SW Guard Extensions(SGX)

Main menu->Socket Configuration->Processor Configuration-> SW Guard Extensions(SGX) ->En

abled

1.3. PRMRR Size

Main menu->Socket Configuration->Processor Configuration-> SW Guard Extensions(SGX) ->PRMRR Size 推荐设置为 32 G （PRMRR 决定单 socket 上预留 EPC 的大小）

2. 安装 Anolis 8.6 操作系统

请参考 Anolis 8.6 GA 说明文档安装 Anolis 8.6 GA。

3. 升级 kernel 到 5.10

• 安装 kernel 5.10

```
yum-config-manager --add-repo https://mirrors.openanolis.cn/anolis/8/kernel-5.10/x86_64/os/ && \
 yum update kernel-5.10.134-12.1.an8
```

提示： 若提示-bash: yum-config-manager: command not found ，请参考以下命令安装yum-utils。

```
yum install -y yum-utils
```

• 重启机器之后，请输入以下命令查看内核版本。

```
uname -r
```

4. 检查 Host 侧的 SGX 使能状态

• 检查 SGX 使能状态。

```
dmesg | grep -i sgx
```

以下输出表示 SGX 已经被正确使能。

```
[    2.326326] sgx: EPC section 0x80c000000-0xfff7fffff
[    2.391710] sgx: EPC section 0x180c000000-0x1ffffffff
[    2.456785] sgx: EPC section 0x280c000000-0x2ffffffff
[    2.522183] sgx: EPC section 0x380c000000-0x3ffffffff
```

• 检查 SGX 驱动安装情况。

```
ls /dev/sgx*
```

以下输出表示已经安装 SGX 驱动。

```
/dev/sgx_enclave  /dev/sgx_provision  /dev/sgx_vepc
```

步骤一：安装QEMU

如果系统 QEMU 的版本低于 v6.2.0，请执行以下命令安装 QEMU v6.2.0。更多详细信息请参考qemu。

```
export version=6.2.0-11.0.1.module+an8.6.0+10830+f21fb638.5
yum install -y qemu-img-$version \
   qemu-guest-agent-$version \
   qemu-kvm-$version \
   qemu-kvm-common-$version \
```

```
qemu-guest-agent
```

步骤二：下载 Guest OS镜像

• 请参考以下命令从龙蜥社区下载 Anolis 8.6 的 Guest OS 镜像。

```
cd $HOME/vsgx/ && \
wget https://mirrors.openanolis.cn/anolis/8.6/isos/GA/x86_64/AnolisOS-8.6-x86_64-ANCK.qcow2
```

• 修改 Guest 镜像的登陆密码.

说明：以下命令行将密码设置为123456， 请根据需求自行设置登陆密码。

```
yum install -y libguestfs-tools-c
export LIBGUESTFS_BACKEND=direct
virt-customize -a AnolisOS-8.6-x86_64-ANCK.qcow2 --root-password password:123456
```

• 结果显示：

```
[   0.0] Examining the guest ...
[   6.5] Setting a random seed
[   6.5] Setting passwords
[   7.6] Finishing off
```

步骤三：启动 SGX Guest

启动方式一：使用 QEMU 命令行启动

请输入以下 QEMU 命令来启动 SGX Guest。

```
/usr/libexec/qemu-kvm \
  -enable-kvm \
  -cpu host,+sgx-provisionkey -smp 8,sockets=1 -m 16G -no-reboot \
  -drive file=$HOME/vsgx/AnolisOS-8.6-x86_64-ANCK.qcow2,if=none,id=disk0,format=qcow2 \
  -device virtio-scsi-pci,id=scsi0,disable-legacy=on,iommu_platform=true -device scsi-hd,drive=disk0 -nographic \
  -monitor pty -monitor unix:monitor,server,nowait \
  -object memory-backend-epc,id=mem1,size=64M,prealloc=on \
  -M sgx-epc.0.memdev=mem1,sgx-epc.0.node=0
```

请输入步骤二设置的用户名和密码，进入SGX Guest。

```
localhost login: root
Password: 123456
```

启动方式二：使用 virsh 启动 sgx guest

virsh 是用于管理 虚拟化环境中的客户机和 Hypervisor 的命令行工具，与 virt-manager 等工具类似，它也是通过 libvirt API 来实现虚拟化的管理。virsh 是完全在命令行文本模式下运行的用户态工具，它是系统管理员通过脚本程序实现虚拟化自动部署和管理的理想工具之一。

安装 libvirt

说明：目前 Anolis 的 libvirt 不支持 sgx 功能，需要打上 intel 的 6 个 patch。目前基于intel 提供的代码自行打包，由 inclavare-containers repo 进行管理。

```
cd $HOME/vsgx && \
  wget https://mirrors.openanolis.cn/inclavare-containers/bin/anolis8.6/libvirt-8.5.0/libvirt-with-vsgx-support.tar.gz && \
  tar zxvf libvirt-with-vsgx-support.tar.gz && \
  cd libvirt-with-vsgx-support && \
  yum localinstall -y *.rpm
```

在 libvirt 中配置 QEMU

在 QEMU 中构建英特尔 SGX 环境，需要访问以下设备：

- `/dev/sgx_enclave` 启动enclave
- `/dev/sgx_provision` 启动供应认证enclave (PCE)
- `/dev/sgx_vepc` 分配 EPC 内存页

libvirt 默认启用的 cgroup 控制器将拒绝访问这些设备文件。编辑 `/etc/libvirt/qemu.conf` 并更改 `cgroup_device_acl` 列表已包括所有三个：

```
cgroup_device_acl = [
  "/dev/null", "/dev/full", "/dev/zero",
  "/dev/random", "/dev/urandom",
  "/dev/ptmx", "/dev/kvm",
  "/dev/rtc","/dev/hpet",
  "/dev/sgx_enclave", "/dev/sgx_provision", "/dev/sgx_vepc"
]
```

QEMU 还需要读取和写入 `/dev/sgx_vepc` 设备，该设备由 root 拥有，文件模式为 600。这意味着您必须将 QEMU 配置为以 root 身份运行。 请编辑 `/etc/libvirt/qemu.conf` ，并设置用户参数。

```
user = “root”
```

进行这些更改后，您需要重新启动 libvirtd 服务：

```
systemctl restart libvirtd
```

配置 NAT 网络

重启 libvirtd 服务之后，请检查当前的网络设置。

```
# virsh net-list --all
 名称      状态    自动开始   持久
-----------------------------------------
 default   活动    否        否
```

libvirt 默认使用了一个名为 default 的 NAT 网络，这个网络默认使用 virbr0 作为桥接接口，使用 dnsmasq 来为使用 nat 网络的虚拟机提供 dns 及 dhcp 服务。

如果您需要自定义 libvirt 虚拟网络，请参考libvirt 网络管理。

创建 SGX Guest xml 文件

以下是 SGX 虚拟机的参考 xml 文件 vsgx.xml, 在使用过程中，请根据实际需求，修改对应的配置字段。

说明：假设 guest image 的位置为/root/vsgx/AnolisOS-8.6-x86_64-ANCK.qcow2, 使用名为default 的 NAT 网络

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
<name>vsgx</name>
<memory unit='KiB'>16777216</memory>
<currentMemory unit='KiB'>16777216</currentMemory>
<vcpu placement='static'>8</vcpu>
<os>
  <type arch='x86_64'>hvm</type>
  <boot dev='hd'/>
</os>
<features>
  <acpi/>
  <apic/>
  <pae/>
</features>
<clock offset='localtime'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<pm>
  <suspend-to-mem enabled='no'/>
  <suspend-to-disk enabled='no'/>
</pm>
<qemu:commandline>
   <qemu:arg value='-cpu'/>
   <qemu:arg value='host,+sgx-provisionkey'/>
   <qemu:arg value='-object'/>
   <qemu:arg value='memory-backend-epc,id=mem1,size=64M,prealloc=on'/>
   <qemu:arg value='-M'/>
   <qemu:arg value='sgx-epc.0.memdev=mem1,sgx-epc.0.node=0'/>
</qemu:commandline>
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/root/vsgx/AnolisOS-8.6-x86_64-ANCK.qcow2'/>
    <target dev='vda' bus='virtio'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
  </disk>
  <controller type='ide' index='0'>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
   </controller>
   <memballoon model='virtio'>
     <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
   </memballoon>
   <console type='pty'>
     <target type='serial' port='0'/>
   </console>
   <interface type='network'>
     <source network='default'/>
     <model type='virtio'/>
     <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
   </interface>
  </devices>

<feature>
  <sgx supported='yes'>
    <flc>yes</flc>
    <epc_size unit='KiB'>1048576</epc_size>
  </sgx>
</feature>
</domain>
```

启动虚拟机

```
# virsh create vsgx.xml
Domain 'vsgx' created from vsgx.xml
```

成功启动虚拟机之后，请输入一下命令列出虚拟机实例。

```
# virsh list
 Id   名称   状态
----------------------
 1    vsgx   running
```

进入虚拟机控制台

```
 # virsh  console vsgx
Connected to domain 'vsgx'
Escape character is ^] (Ctrl + ])
# 这里要敲一下回车键
Anolis OS 8.6
Kernel 4.19.91-26.an8.x86_64 on an x86_64

Activate the web console with: systemctl enable --now cockpit.socket

localhost login:
```

请输入步骤二设置的用户名和密码，进入SGX guest。

```
localhost login: root
Password: 123456
```

检查 Guest 的 SGX 使能状态

不管是用 QEMU 命令行直接启动的 SGX Guest，还是使用 virsh 启动的 SGX Guest，在启动之后，都需要检查 Guest 中对 SGX 是否支持。

在 Guest 中使用 SGX 需要支持 SGX 的内核/操作系统。可以通过以下方式在 Guest 中确定支持：

• 检查 SGX 使能状态。

```
dmesg | grep -i sgx
```

以下输出表示 SGX 已经被正确使能。

```
[    0.489460] sgx: EPC section 0x440000000-0x443ffffff
```

• 检查 SGX 驱动安装情况。

```
ls /dev/sgx_*
```

以下输出表示 SGX 已经被正确使能。

```
/dev/sgx_enclave  /dev/sgx_provision
```

步骤四：构建 SGX 加密计算环境

为开发 SGX 程序，您需要在 SGX 虚拟机上安装 SGX SDK，PSW 和 DCAP 进而构建 SGX 加密计算环境。

• 安装 SGX SDK

```
mkdir -p $HOME/vsgx && \
  wget https://mirrors.openanolis.cn/inclavare-containers/bin/anolis8.6/sgx-2.17/sgx_linux_
  x64_sdk_2.17.100.3.bin && \
  chmod +x sgx_linux_x64_sdk_2.17.100.3.bin && \
echo -e 'n\n\/opt/intel\n' | ./sgx_linux_x64_sdk_*.bin && \
rm -rf sgx_linux_x64_sdk_*.bin
```

• 安装 SGX PSW/DCAP

```
cd $HOME/vsgx && \
  wget https://mirrors.openanolis.cn/inclavare-containers/bin/anolis8.6/sgx-2.17/sgx_rpm
  _local_repo.tar.gz && \
  tar zxvf sgx_rpm_local_repo.tar.gz && \
  yum install -y yum-utils && \
  yum-config-manager --add-repo file://$HOME/vsgx/sgx_rpm_local_repo/ && \
  yum install --nogpgcheck -y sgx-aesm-service libsgx-launch libsgx-urts && \
  rm -rf sgx_rpm_local_repo.tar.gz
```


OpenAnolis
龙 蜥 社 区

步骤五：验证 SGX 功能示例一：启动 Enclave

Intel SGX SDK 中提供了 SGX 示例代码用于验证 SGX 功能，默认位于 opt/intel/sgxsdk/SampleCode 目录下。

本节演示其中的启动 Enclave 示例（SampleEnclave），效果为启动一个 Enclave，以验证是否可以正常使用安装的 SGX SDK。

• 安装编译工具

```
yum install -y gcc-c++
```

• 设置 SGX SDK 相关的环境变量。

```
source /opt/intel/sgxsdk/environment
```

• 编译示例代码 SampleEnclave

```
cd /opt/intel/sgxsdk/SampleCode/SampleEnclave && \
  make
```

• 运行编译出的可执行文件。

```
./app
```

预期输出：

```
Checksum(0x0x7ffc732e8d20, 100) = 0xfffd4143
Info: executing thread synchronization, please wait...
Info: SampleEnclave successfully returned.
Enter a character before exit ...
```

步骤六：停止虚拟机

QEMU 停止虚拟机

在 SGX Guest 里输入exit 退出 Guest 虚拟机，然后 CTRL+C 终止 QEMU 进程。

vrish 停止虚拟机

停止虚拟机

请在 Host 机器上，输入以下命令停止虚拟机。

```
# virsh shutdown vsgx
Domain 'vsgx' is being shutdown
```

删除虚拟机

请在 Host 机器上，输入以下命令删除虚拟机。

```
# virsh undefine vsgx
Domain 'vsgx' has been undefined
```

强制停止虚拟机

> 说明：仅限 shutdown 不工作时才使用 destroy。

```
virsh destroy vsgx
```


OpenAnolis 龙蜥社区

AMD SEV机密容器

本文主要为您介绍如何基于AMD安全加密虚拟化功能SEV(AMD Secure Encrypted Virtualization)技术，通过远程证明启动一个租户的加密容器镜像。

前提条件

1. 安装Anolis 8.6 操作系统

请在支持AMD CPU的硬件设备上，参考Anolis 8.6 GA说明文档安装anolis 8.6 GA。

2. 升级内核到5.10

由于 Anlois 8.6 的默认内核版本是4.19，请升级kernel 到5.10版本。

- 添加 yum 源配置参数，添加Anolis 的 Experimental repo。

```
yum install yum-utils
yum-config-manager --add-repo https://mirrors.openanolis.cn/anolis/8/kernel-5.10/x86_64
/os/
```

- 升级内核

```
yum update kernel
```

- 重启机器，并重新查看机器的操作系统发行编号。

```
reboot
uname -r
```

- 预期结果如下：

```
5.10.134-13_rc2.an8.x86_64
```

3. 使能SEV

注意：在 Anolis 8.6 中，grub 版本默认为1:2.02-123.0.2.an8_6.8。此版本存在BUG，如果直接进行使能sev的操作，会导致机器重启后无法进入系统的情况。请采用降级grub的workround方法。

- 修改 yum 源，在 Anolis 8.5 中才有低版本的 grub。

```
cd /etc/yum.repos.d
sed -i 's/$releasever/8.5/' AnolisOS-BaseOS.repo
```

- 降级 grub。

```
um downgrade grub2-efi
```

• 查看 grub 的版本，预期结果如下：

```
# yum list | grep grub
grub2-common.noarch           1:2.02-106.0.1.an8         @BaseOS
grub2-efi-x64.x86_64          1:2.02-106.0.1.an8         @BaseOS
grub2-pc.x86_64               1:2.02-106.0.1.an8         @BaseOS
grub2-pc-modules.noarch       1:2.02-106.0.1.an8         @BaseOS
grub2-tools.x86_64            1:2.02-106.0.1.an8         @BaseOS
grub2-tools-efi.x86_64        1:2.02-106.0.1.an8         @BaseOS
grub2-tools-extra.x86_64      1:2.02-106.0.1.an8         @BaseOS
grub2-tools-minimal.x86_64    1:2.02-106.0.1.an8         @BaseOS
```

使能SEV和SME

默认情况下SEV和SME没有使能，请输入以下内核命令行进行使能：

```
grubby --update-kernel=ALL --args="mem_encrypt=on kvm_amd.sev=1"
```

重启机器。

```
reboot
```

重启后，请检查机器的sev使能状态。

```
dmesg | grep -i sev
```

预期结果如下：

```
[    6.747923] ccp 0000:4b:00.1: sev enabled
[    6.842676] ccp 0000:4b:00.1: SEV firmware update successful
[    6.997400] ccp 0000:4b:00.1: SEV API:1.42 build:42
[    7.522437] SEV supported: 255 ASIDs
```

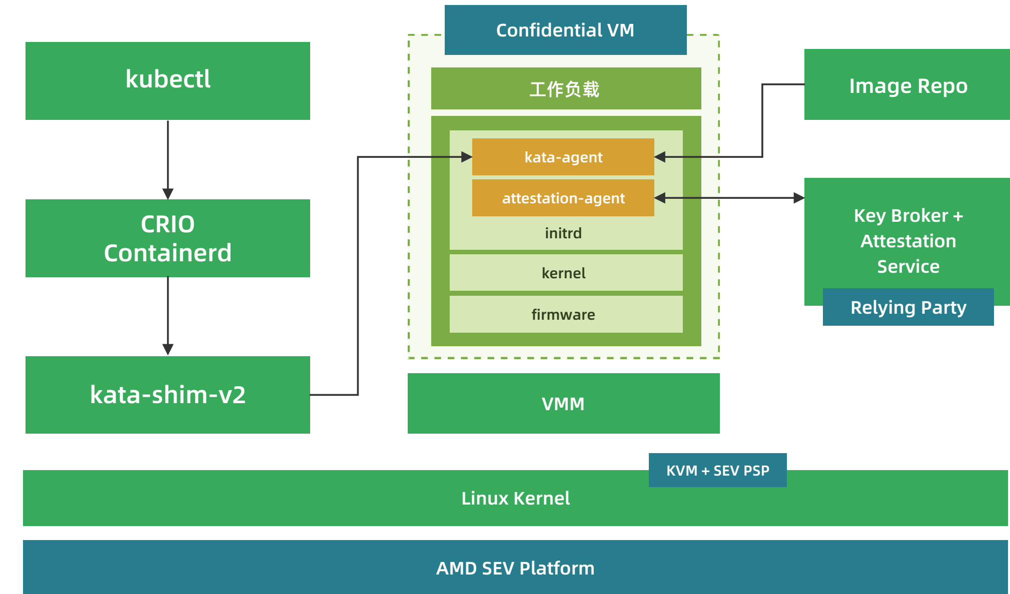
最后检查 SME(Secure Memory Encryption) 的状态。

```
dmesg | grep -i sme
```

预期结果如下：

```
[    1.863927] AMD Memory Encryption Features active: SME
```


背景信息



AMD SEV Pod 级机密容器架构基于 Kata Containers 项目，最大区别是将基于普通虚拟化技术实现的轻量级 Sandbox Pod替换为基于机密计算技术实现的轻量级 TEE Pod，目的是将特定租户的整个 Pod 以及其中的容器运行在受 CPU TEE 保护的执行环境中。除此之外，TEE Pod 内部还额外集成了 image-rs 和 attestation-agent 等组件，它们负责实现容器镜像的拉取、授权、验签、解密、远程证明以及秘密注入等安全特性。机密容器的基本运行过程为：

• 用户使用标准工具制作一个签名和/或加密的受保护的容器镜像，并上传到容器镜像仓库中。

• 用户命令 Kubernetes 启动这个受保护的容器镜像。kubelet 会向 containerd 发起创建 Pod 的 CRI 请求，containerd 则把请求转发给 kata-runtime。

• kata runtime 与 Key broker service（simple kbs）建立安全会话，并进行基于CPU TEE 硬件的身份认证与授权。KBS基于安全可信信道发送敏感数据给kata runtime。kata runtime 调用QEMU 将秘密信息注入到 guest userland中。之后再调用 QEMU 启动 Pod。

• CPU TEE 执行初始化，最终启动 kata-agent 监听后续请求。

• kubelet 向 containerd 发起 Image Pulling 的 CRI 请求，containerd 则把请求转发给 kata-runtime，最终 kata-agent 收到请求并通过 image-rs 子模块提供的容器镜像管理功能，在 TEE 内安全地执行拉取、验签、解密、unpack 以及挂载容器镜像的操作。

步骤一：部署测试集群

本步骤为您提供快速部署单节点测试集群的步骤。您可以根据您的需求，灵活部署集群。

配置权限

关闭firewall

Linux系统下面自带了防火墙 iptables ，iptables 可以设置很多安全规则。但是如果配置错误很容易导致各种网络问题。此处建议关闭 firewall 。 执行如下操作：

```
sudo service firewalld stop
```

检查 firewall 状态：

```
service firewalld status
```

预期结果如下：

```
Redirecting to /bin/systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset:enabled)
    Active: inactive (dead)
    Docs: man:firewalld(1)
```

关闭selinux

Security-Enhanced Linux（SELinux）是一个在内核中实施的强制存取控制（MAC）安全性机制。为避免出现权限控制导致的虚拟机启动、访问失败等问题，此处建议关闭selinux。执行如下操作：

```
setenforce 0
```

预期结果如下：

```
setenforce: SELinux is disabled
```

安装operator-sdk

operator SDK 项目是 Operator Framework 的一个组件，Operator Framework 是一个开源工具包，用于以有效、自动化和可扩展的方式管理 Kubernetes 原生应用程序，称为 Operators。具体信息，请参考 operator SDK。

```
wget -O /usr/local/bin/operator-sdk https://github.com/operator-framework/operator-sd
k/releases/download/v1.23.0/operator-sdk_linux_amd64
sudo chmod +x /usr/local/bin/operator-sdk
```

启动本地 docker registry

• 请执行下列脚本安装 docker

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y containerd.io docker-ce docker-ce-cli
systemctl start docker.service
```

• 执行如下命令，启动本地 docker registry，该registry 用于存储operator images。

```
docker run -itd -p 5000:5000 docker.io/library/registry:latest
```

检查 docker 容器是否启动成功：

```
docker ps
```

预期结果如下，注意，状态（STATUS）应该是Up的。

```
CONTAINER ID   IMAGE            COMMAND                   CREATED          STATUS
a7cc49ee1d19   registry:latest  "/entrypoint.sh /etc…"    38 minutes ago   Up 38 minutes
0.0.0.0:5000->5000/tcp, :::5000->5000/tcp  nostalgic_montalcini
```

配置containerd

自动生成默认的config.toml

```
containerd config default > /etc/containerd/config.toml
```

由于默认的 config.toml 使用的是国外的镜像，国内有可能无法访问。请参考以下命令修改为国内镜像。

```
cd /etc/containerd
sed -i 's#registry.k8s.io/pause:3.6#registry.cn-hangzhou.aliyuncs.com/google_containers/
pause:3.1#g' config.toml
```

启动 containerd

```
systemctl containerd start
```

部署单节点的Kubernetes cluster

• 请参考kubernetes官方指南安装Kubernetes cluster。最低 Kubernetes 版本应为 1.24。

• 确保集群中至少有一个 Kubernetes 节点具有标签 node-role.kubernetes.io/worker=

```
kubectl label node <node-name> node-role.kubernetes.io/worker=
```

步骤二：安装Confidential computing Operator

Confidential computing Operator 提供了一种在 Kubernetes 集群上部署和管理 Confidential Containers Runtime 的方法。具体信息请参考指南。

前提条件

1、确保 Kubernetes 集群节点至少有 8GB RAM 和 4 个 vCPU

2、当前 CoCo 版本仅支持基于 containerd 运行时的 Kubernetes 集群

3、确保 SELinux 被禁用或未强制执行 (confidential-containers/operator#115)

部署Operator

Operator目前有3个版本，这里默认安装最新版v0.3.0版本。 通过运行以下命令部署Operator：

```
kubectl apply -k github.com/confidential-containers/operator/config/release?ref=v0.3.0
```

cc-operator-controller-manager 资源依赖国外的镜像，可能拉不下来，请参考以下步骤对镜像进行修改：

```
kubectl edit deploy cc-operator-controller-manager -n confidential-containers-system

# 将gcr.io/kubebuilder/kube-rbac-proxy:v0.13.0替换成
image: quay.io/brancz/kube-rbac-proxy:v0.13.0
```

查看节点状态：

```
kubectl get pods -n confidential-containers-system --watch
```

预期结果如下。注意这三个pod都要存在，且STATUS都要为Running。

```
NAME                                            READY   STATUS    RESTARTS   AGE
cc-operator-controller-manager-56cb4d5ff5-lqd9x  2/2     Running   0          167m
cc-operator-daemon-install-rg8s9                1/1     Running   0          154m
cc-operator-pre-install-daemon-7jhnw            1/1     Running   0          154m
```

创建custom resource

创建 custom resource 会将所需的 CC runtime安装到集群节点中并创建 RuntimeClasses。操作如下：

```
kubectl apply -k github.com/confidential-containers/operator/config/samples/ccruntime/
default?ref=v0.3.0
```

检查创建的 RuntimeClasses。

```
kubectl get runtimeclass
```

预期结果如下：

```
NAME             HANDLER          AGE
kata             kata             154m
kata-clh         kata-clh         154m
kata-clh-tdx     kata-clh-tdx     154m
kata-qemu        kata-qemu        154m
kata-qemu-sev    kata-qemu-sev    154m
kata-qemu-tdx    kata-qemu-tdx    154m
```

卸载Operator（非必要步骤）

如果您想更新Operator的版本，或者您的安装出现问题，可以先卸载，再回到上面的步骤重新安装。具体操作请参考链接。

```
kubectl delete -k github.com/confidential-containers/operator/config/samples/ccruntime/
default?ref=<RELEASE_VERSION>

kubectl delete -k github.com/confidential-containers/operator/config/release?ref=$
{RELEASE_VERSION}
```

步骤三：启动Simple KBS

simple kbs是一个密钥代理服务，可以存储并向 workload 提供 secret 。对于 SEV 加密容器示例来说，需要从simple kbs 中获取 secret ，并用于解密已加密的容器。 在步骤四的示例二中，本文提供了一个简单的加密镜像(docker.io/haosanzi/busybox-v1:encrypted)，该镜像使用 simple kbs 已经存在的密钥来解密，同时对 policy 不进行校验。此加密镜像只作为测试使用，如您想用于自己的生产用例中，请参考指南制作一个新的加密镜像并部署。

要了解有关创建 policy 的更多信息，请参考指南。

• 安装 docker-compose 后，才能在 docker 容器中运行 simple-kbs 及其数据库，数据库中存放了 secret 等信息：

```
dnf install docker-compose-plugin
```

• 下载 simple-kbs 的代码：

```
simple_kbs_tag="0.1.1"
git clone https://github.com/confidential-containers/simple-kbs.git
cd simple-kbs && git checkout -b "branch_${simple_kbs_tag}" "${simple_kbs_tag}"
```

• 使用 docker-compose 运行 simple-kbs ：

```
cd simple-kbs
sudo docker compose up -d
```

步骤四：运行workload

示例一：运行一个未加密的容器镜像

为了验证主机上不存在容器镜像，应该登录到 k8s 节点并确保以下命令返回空结果：

```
crictl  -r  unix:///run/containerd/containerd.sock image ls | grep bitnami/nginx
```

启动POD

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: bitnami/nginx:1.22.0
    name: nginx
  dnsPolicy: ClusterFirst
  runtimeClassName: kata
EOF
```

预期结果：

```
pod/nginx created
```

查看 pod 状态：

```
kubectl get pods
```

预期结果如下，注意， STATUS 要是 Running 。

```
NAME    READY   STATUS    RESTARTS    AGE
nginx   1/1     Running   0           3m50s
```

示例二：运行一个加密容器

attestation agent 支持两种SEV平台相关的KBC：offline_sev_kbc 和 online_sev_kbc。

• offline sev KBC 在运行时不会与 Simple KBS 进行通信，而是使用在VM Boot时期通过QEMU注入的secret。该机制的缺点是对注入的 secret 长度有限制。

• online sev KBC 在offline sev KBC的基础上，支持在运行时发出请求。online sev KBC 在VM Boot时期通过QEMU注入connection。注入的connection包含一个对称密钥，用于加密和验证 KBC 发出的在线请求。 该连接受 SEV(-ES) 秘密注入过程保护，该过程提供机密性、完整性并防止重放攻击。 simple-kbs 为每个连接生成一个新的对称密钥。 KBC 要求每个在线secret都带有随机 guid 以防止重放攻击。

> 注意：offline_sev_kbc 和 online_sev_kbc 是两种option，用户只需要采用一种KBC方式运行镜像即可。

导出SEV证书链

sevctl 是 SEV 平台的命令行管理工具，Kata 机密容器需要 SEV 证书链从而与guest owner建立安全会话。请按照以下步骤安装 sevctl：

```
dnf install sevctl
```

SEV 证书链必须放在 /opt/sev 中，使用以下命令导出 SEV 证书链：

```
mkdir -p /opt/sev
sevctl export --full /opt/sev/cert_chain.cert
```

基于online KBC运行机密容器

• 请下载支持online sev kbc 的 initrd：

```
wget https://mirrors.openanolis.cn/inclavare-containers/confidential-containers-demo/bin/ccv3-sev/initrd.run.online-sev.img -O /opt/confidential-containers/share/kata-containers/kata-containers-initrd-sev.img
```

• 自定义 policy， 请参考附录部分。

• 编辑 kata 配置文件：

```
kbs_ip="$(ip -o route get to 8.8.8.8 | sed -n 's/.*src \([0-9.]\+\).*/\1/p')"
sed -i 's/agent.enable_signature_verification=false /&agent.aa_kbc_params=online_sev_kbc::'$kbs_ip':44444/' /opt/confidential-containers/share/defaults/kata-containers/configuration-qemu-sev.toml
```

• 启动 Pod

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: test-en-online
  name: test-en-online
spec:
  containers:
  - image: docker.io/haosanzi/busybox-v1:encrypted
    name: test-en-online
    imagePullPolicy: Always
  dnsPolicy: ClusterFirst
  restartPolicy: Never
  runtimeClassName: kata-qemu-sev
EOF
```

查看 pod 是否启动成功：

```
kubectl get po
```

预期结果如下：

```
NAME             READY   STATUS    RESTARTS   AGE
test-en-online   1/1     Running   0          146m
```

基于offline KBC运行加密容器

• 请下载支持offline KBC的initrd。

```
wget https://mirrors.openanolis.cn/inclavare-containers/confidential-containers-demo/bin/ccv3-sev/initrd.run.online-sev.img -O /opt/confidential-containers/share/kata-containers/kata-containers-initrd-sev.img
```

• 编辑 kata 配置文件：

```
kbs_ip="$(ip -o route get to 8.8.8.8 | sed -n 's/.*src \([0-9.]\+\).*/\1/p')"
sed -i 's/agent.enable_signature_verification=false /&agent.aa_kbc_params=online_sev_kbc::'$kbs_ip':44444/' /opt/confidential-containers/share/defaults/kata-containers/
```

• 自定义 policy， 请参考附录部分。

• 启动 Pod

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
```

```
metadata:
  labels:
    run: test-en-offline
  name: test-en-offline
spec:
  containers:
  - image: docker.io/haosanzi/busybox-v1:encrypted
    name: test-en-offline
    imagePullPolicy: Always
  dnsPolicy: ClusterFirst
  restartPolicy: Never
  runtimeClassName: kata-qemu-sev
EOF
```

查看 pod 是否启动成功：

```
kubectl get po
```

预期结果如下：

```
NAME             READY   STATUS    RESTARTS   AGE
test-en-offline   1/1    Running   0          31h
```

附录

制作一个新的加密镜像并部署

请参考指南制作一个新的加密镜像并部署。

自定义simpel KBS 的policy

• sev-snp-measure是一个实用程序，用于使用提供的 ovmf、initrd、kernel、cmdline等作为参数来计算 SEV guest固件测量值。下载sev-snp-measure：

```
git clone https://github.com/IBM/sev-snp-measure.git
cd sev-snp-measure
```

• 根据ovmf、kernel和initrd_path的地址设置参数。

ovmf、kernel和initrd_path的地址请参考kata 的配置文件

kata 的配置文件路径：/opt/confidential-containers/share/defaults/kata-containers/configuration-qemu-sev.toml。

```
ovmf_path="/opt/confidential-containers/share/ovmf/OVMF.fd"
kernel_path="/opt/confidential-containers/share/kata-containers/vmlinuz-sev.container"
initrd_path="/opt/confidential-containers/share/kata-containers/kata-containers-initrd.img"
```

• 计算内核的append值

```
duration=$((SECONDS+30))
```


OpenAnolis
龙 蜥 社 区

```
set append

while [ $SECONDS -lt $duration ]; do
  qemu_process=$(ps aux | grep qemu | grep append || true)
  if [ -n "${qemu_process}" ]; then
    append=$(echo ${qemu_process} \
      | sed "s|.*-append \(.*$\)|\1|g" \
      | sed "s| -.*$||")
    break
  fi
  sleep 1
done

echo "${append}"
```

• 使用 sev-snp-measure 来计算 SEV guest 的Launch digest。

```
 measurement=$(./sev-snp-measure.py --mode=sev --output-format=base64 \
--ovmf "${ovmf_path}" \
  --kernel "${kernel_path}" \
  --initrd "${initrd_path}" \
  --append "${append}" \
)
```

• 设置simple kbs 数据库参数

```
KBS_DB_USER="kbsuser"
KBS_DB_PW="kbspassword"
KBS_DB="simple_kbs"
KBS_DB_TYPE="mysql"
KBS_DB_HOST=$(docker network inspect simple-kbs_default \
  | jq -r '.[].Containers[] | select(.Name | test("simple-kbs[_-]db.*")).IPv4Address' \
  | sed "s|/.*$||g")
```

• 由于本文使用的加密镜像(docker.io/haosanzi/busybox-v1:encrypted)，是采用 simple kbs 已经存在的密钥来解密，该镜像的 enc_key 值如下。用户需要根据加密镜像按需设置enc_key。

```
enc_key=RcHGava52DPvj1uoIk/NVDYlwxi0A6yyIZ8ilhEX3X4=
```

• 将 自定义policy 注入 mysql 中。

policy的组成包括：digests、policies、api_major、api_minor、build_ids等信息。详情请参考链接。

我们以digests为例子，向用户展示如何注入自定义policy 。用户可以根据需求自定义Policy。

```
mysql -u${KBS_DB_USER} -p${KBS_DB_PW} -h ${KBS_DB_HOST} -D ${KBS_DB} <<EOF
  REPLACE INTO secrets VALUES (10, 'key_id1', '${enc_key}', 10);
  REPLACE INTO keysets VALUES (10, 'KEYSET-1', '["key_id1"]', 10);
  REPLACE INTO policy VALUES (10, '["${measurement}"]', '[]', 0, 0, '[]', now(), NULL, 1);
EOF
```

AMD SEV机密虚拟机

项目位置链接

AMD SEV 技术基于AMD EPYC CPU，将物理机密计算能力传导至虚拟机实例，在公有云上打造一个立体化可信加密环境。 SEV可保证单个虚拟机实例使用独立的硬件密钥对内存加密，同时提供高性能支持。密钥由AMD 平台安全处理器 (PSP)在实例创建期间生成，而且仅位于处理器中，云厂商无法访问这些密钥。

测试环境

硬件配置

CPU: AMD EPYC 7763 *1

Memory: DDR4 3200 32G *16

软件信息

OS: Anolis 8.5.0-10.0.1

Kernel: 5.10.134-12.1.an8.x86_64

第一步 开启 SME和 SEV

开启 SME

将 mem_encrypt=on 添加到kernel的引导参数

Enable SEV

将 kvm_amd.sev=1 kvm.添加到kernel的引导参数中

确保Kernel的引导参数生效

1、vim /etc/default/grub

2、增加 "kvm_amd.sev=1 mem_encrypt=on" 到 "GRUB_CMDLINE_LINUX_DEFAULT=" 这一行

3、grub2-mkconfig -o /boot/efi/EFI/anolis/grub.cfg

第二步 重启服务器进入BIOS开启SEV相关选项

BIOS 配置项如下

1、Advanced->AMD CBS->CPU Common Options->SMEE->Enable

2、Advanced->AMD CBS->NBIO Common Options->IOMMU->Enabled

3、Advanced->AMD CBS->NBIO Common Options->SEV-SNP Support->Enable

Anolis OS 对SEV的配置进行确认

1、cat /proc/cmdline

确保输出有"mem_encrypt=on kvm_amd.sev=1"

类似输出如下

```
BOOT_IMAGE=(hd1,gpt2)/vmlinuz-5.10.134-12.1.an8.x86_64 root=/dev/mapper/ao-root ro crashkernel=auto resume=/dev/mapper/ao-swap rd.lvm.lv=ao/root rd.lvm.lv=ao/swap rhgb quiet mem_encrypt=on kvm_amd.sev=1 kvm_amd.sev_es=1
```

2、dmesg | grep -i SEV

确保输出有 "SEV supported"

类似输出如下

```
[    5.145496] ccp 0000:47:00.1: sev enabled
[    5.234221] ccp 0000:47:00.1: SEV API:1.49 build:6
[    5.445958] SEV supported: 253 ASIDs
```

3、cat /sys/module/kvm_amd/parameters/sev

确保输出值是 "1" 或者 "Y"

第三步 启动虚拟机相关的准备工作

1、yum update && yum upgrade

2、yum install libvirt-daemon virt-manager libvirt-client qemu-kvm epel-release cloud-utils virt-install

3、virsh domcapabilities

确保输出有 "sev supported='yes'"

类似输出如下

```
<sev supported='yes'>
    <cbitpos>51</cbitpos>
    <reducedPhysBits>1</reducedPhysBits>
    <maxGuests>253</maxGuests>
    <maxESGuests>0</maxESGuests>
</sev>
```

VM 网络环境配置

1、创建默认的网络环境配置 default.xml

```
<network>
    <name>default</name>
    <forward mode='nat'/>
    <bridge name='virbr0' stp='on' delay='0'/>
```

```
<mac address='52:54:00:0a:cd:21'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

1、virsh net-define --file default.xml

2、virsh net-start default

3、virsh net-autostart --network default

第四步 VM 镜像的配置

ubuntu 作为VM

1、wget *https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img*

2、qemu-img convert focal-server-cloudimg-amd64.img /var/lib/libvirt/images/sev-guest.img

3、创建VM镜像用户和密码配置文件 cloud-config

```
#cloud-config
   ssh_pwauth: True
password: 123456
chpasswd: { expire: False }

chpasswd:
  list: |
     root:123456
     ubuntu:123456
expire: False
```

4、cloud-localds /var/lib/libvirt/images/init-passwd.iso cloud-config

第五步 启动VM

virsh 安装的方式

```
virt-install \
      --name sev-guest \
            --memory 4096 \
            --memtune hard_limit=4563402 \
            --boot uefi \
            --disk /var/lib/libvirt/images/sev-guest.img,device=disk,bus=scsi \
            --disk /var/lib/libvirt/images/init-passwd.iso,device=cdrom \
            --os-type linux \
            --os-variant centos8 \
            --import \
            --controller type=scsi,model=virtio-scsi,driver.iommu=on \
            --controller type=virtio-serial,driver.iommu=on \
            --network network=default,model=virtio,driver.iommu=on \
            --memballoon driver.iommu=on \
            --graphics none \
           --launchSecurity sev
```

virsh 用xml文件 启动

• 创建sev.xml 文件,内容如下

```
<domain type = 'kvm'  xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
 <name>csv_launch</name>
   <memory unit='GiB'>4</memory>
   <vcpu>4</vcpu>
   <os>
      <type arch = 'x86_64' machine = 'pc'>hvm</type>
      <boot dev = 'hd'/>
   </os>
   <features>
      <acpi/>
      <apic/>
      <pae/>
   </features>
   <clock offset = 'utc'/>
   <on_poweroff>destroy</on_poweroff>
   <on_reboot>restart</on_reboot>
   <on_crash>destroy</on_crash>
   <devices>
      <emulator>/usr/libexec/qemu-kvm</emulator>
      <disk type = 'file' device = 'disk'>
         <driver name = 'qemu' type = 'qcow2' cache = 'none'/>
         <source file = '/tmp/test.qcow2'/>
         <target dev = 'hda' bus = 'ide'/>
      </disk>
      <memballoon model='none'/>
      <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0' keymap='en-us'>
         <listen type='address' address='0.0.0.0'/>
      </graphics>
   </devices>

   <launchSecurity type='sev'>
      <policy>0x0001</policy>
      <cbitpos>51</cbitpos>
      <reducedPhysBits>1</reducedPhysBits>
   </launchSecurity>

   <qemu:commandline>
      <qemu:arg value="-drive"/>
      <qemu:arg value="if=pflash,format=raw,unit=0,file=/usr/share/edk2/ovmf/OVMF_
```

```
CODE.cc.fd,readonly=on"/>
  </qemu:commandline>
</domain>
```

• 导入sev-guest虚拟机

virsh define sev.xml

• 开启虚拟机

virsh start sev-guest

第六步 检查SEV 在虚拟机中是否开启

1、virsh console sev-guest

2、dmesg | grep SEV

```
[    0.374549] AMD Memory Encryption Features active: SEV
```


Occlum: 基于Intel SGX的轻量级LibOS

项目位置链接

https://github.com/occlum/occlum

技术自身介绍

背景

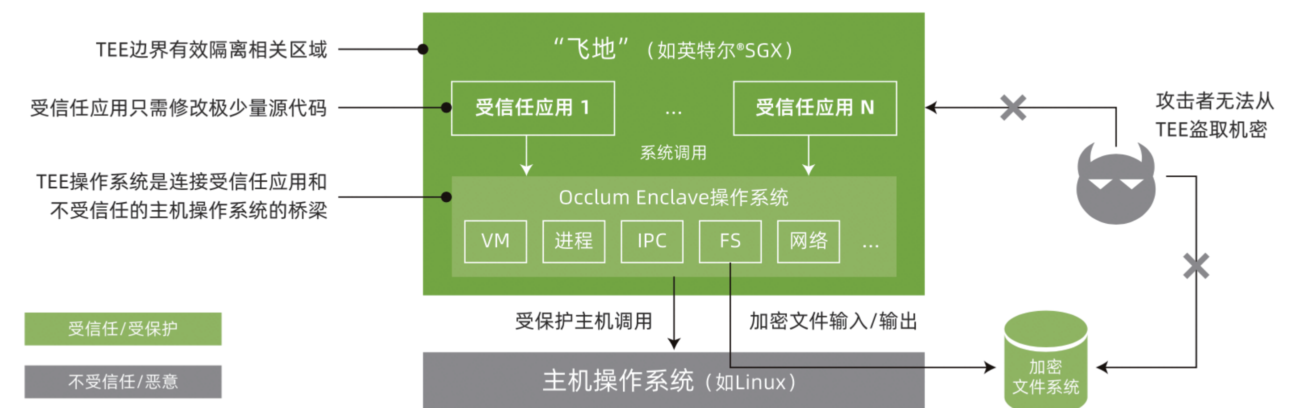
数据在使用中的隐私安全问题越来越受到业界的广泛关注。其中，基于硬件的TEE技术，尤其是英特尔® 软件防护扩展（英特尔® SGX）是TEE技术里应用最广泛的。它提供基于硬件的内存加密，隔离内存中的特定应用代码和数据，能够更有效地抵御多种类型的攻击。它可显著加强数据安全，满足对于机密计算的广泛需求。英特尔® SGX 允许为用户级代码分配专用内存区域（Encalve，安全飞地），以免受到拥有更高权限的进程的影响。英特尔® SGX 经过了严格测试，是业界广泛部署的基于硬件的数据中心可信执行环境 (TEE)，大幅减少了系统中的攻击面。

问题&挑战

传统的基于SGX SDK的编程方式，对应用开发者来说是一个极大的挑战。它需要应用开发者在熟悉TEE/SGX SDK的基础上，重新设计，分割，编译已有的应用，对广发应用TEE技术于各种应用来说，难度极大。

解决方案

Occlum是基于SGX基础上实现的一套轻量级LibOS，大大简化了应用开发者的难度。使用 Occlum 后，机器学习工作负载等只需修改极少量（甚至无需修改）源代码即可在英特尔® SGX 上运行，以高度透明的方式保护了用户数据的机密性和完整性。



应用场景

场景描述

Occlum为蚂蚁摩斯（TEE 服务平台，将 TEE 能力作为一种 SaaS 服务开放出来）提供技术底座。 Occlum为Intel的一个分布式的隐私保护机器学习（Privacy Preserving Machine Learning, PPML）平台提供TEE技术底座。

应用效果

总体上来说，Occlum强大高效的LibOS使得用户的应用可以很方便的运行在TEE环境里，大大推动了TEE机密计算生态的发展。无论是在机密数据库（MySQL，PosrgreSQL，MongoDB），还是大数据、机器学习领域（PyTorch，Tensorflow，Spark），Occlum都有实例证明了其性能和兼容性的优秀。

Inclavare Containers: 面向机密计算场景的开源容器运行时技术栈

项目位置链接

https://github.com/inclavare-containers

技术自身介绍

背景

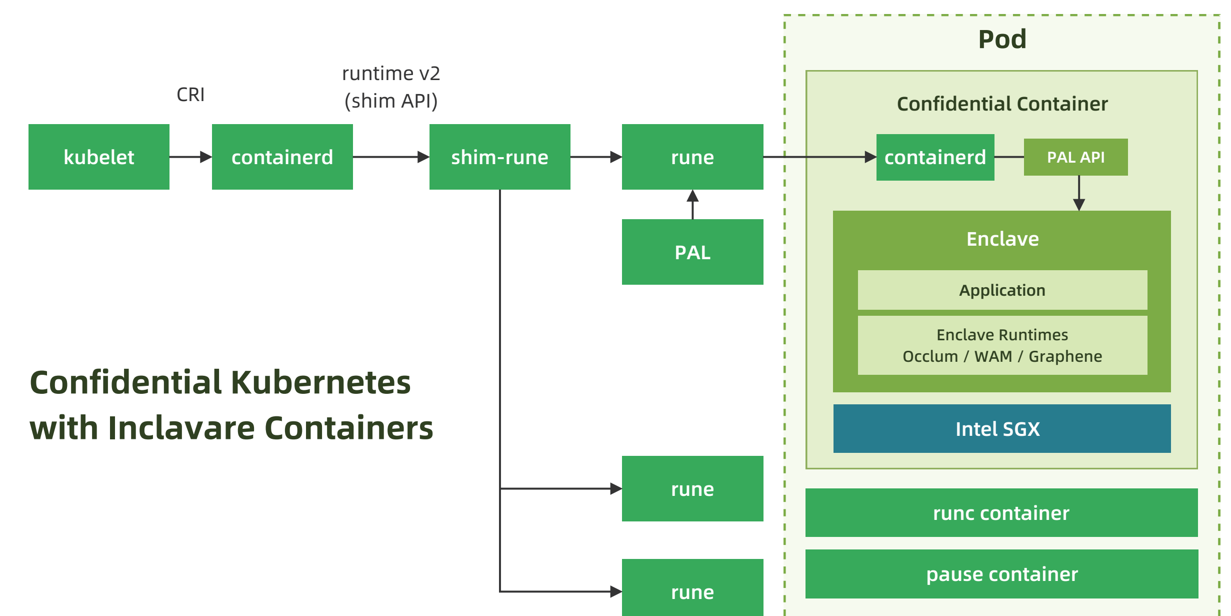
机密计算是一种能够通过软件加密算法和硬件HW-TEE保护用户数据和程序的技术。 在云原生场景中， 机密级算能够对计算中的数据提供机密性和完整性保护， 并能够有效防止云提供商和任何高权限的第三方对执行环境中的数据进行窃取和篡改。

问题&挑战

对于数据隐私和数据安全度关注高的用户， 不愿意冒风险，把自己的业务部署在共有云上， 因为云提供商具有查看用户隐私数据和业务的可能性， 这也阻止了一些企业上云的脚步。随着机密计算蓬勃发展，硬件厂商提供了HW-TEE硬件加密的方案的涌现， 可以有效解决安全敏感度较高的用户上云顾虑。 对于公有云人们开发和部署很多是以容器的方式进行， 如何让HW-TEE和容器技术相结合，成为云厂商迫切要解决的问题。

解决方案

Alibaba和Intel把Intel-SGX和容器技术相结合， 创新性的开发出了机密容器Inclavare containers, 完美兼容容器标准，并能够为客户的数据和程序进行保驾护航， 并且Inclavare containers 容器成为CNCF的第一个机密容器。



如下图所示，Inclavare Containers 中包含多个组件，这些组件可以利用基于硬件支持的 Enclave 技术使可信应用运行在容器中。包含的组件有 rune、shim-rune、Enclave Runtime等。

• rune：rune 是一个命令行工具，用于根据 OCI 规范在容器中生成和运行 Enclave。rune 是在 runc 代码基础上开发的，既可以运行普通 runc 容器也可以运行 Enclave 容器；rune已经写入 OCI 运行时实现列表：*https://github.com/opencontainers/runtimespec/blob/master/implementations.md。*

• shim-rune：为容器运行时 rune 提供的 shim，主要负责管理容器的生命周期、把普通镜像转换成 TEE 镜像；管理容器的生命周期，与 rune 配合完成容器的创建、启动、停止、删除等操作。

• Enclave Runtime：负责在 Enclave 内加载和运行受信任和受保护的应用程序。rune 和 Enclave Runtime 之间的接口是 Enclave Runtime PAL API，它允许通过定义良好的函数接口来调用 Enclave Runtime。机密计算应用通过这个接口与云原生生态系统进行交互。

一类典型的 Enclave Runtime 实现基于库操作系统。目前，推荐的与 rune 交互的 Enclave Runtime 是 Occlum，这是一种内存安全、多进程 Libos。另一类典型的 Enclave Runtime是带有 Intel® SGX WebAssembly Micro Runtime (WAMR)，这是一个占用空间很小的独立 WebAssembly (WASM) 运行时，包括一个 VM 核心、一个应用程序框架和一个 WASM 应用程序的动态管理。

此外，您可以使用您喜欢的任何编程语言和 SDK（例如英特尔 SGX SDK）编写自己的Enclave Runtime，只要它实现了 Enclave Runtime PAL API。

Inclavare Contianers主要有以下特点：

1、将 Intel® SGX 技术与成熟的容器生态结合，将用户的敏感应用以 Enclave 容器的形式部署和运行；Inclavare Contianers 的目标是希望能够无缝运行用户制作的普通容器镜像，这将允许用户在制作镜像的过程中，无需了解机密技术所带来的复杂性，并保持与普通容器相同的使用体感。

2、Intel® SGX 技术提供的保护粒度是应用而不是系统，在提供很高的安全防护手段的同时，也带来了一些编程约束，比如在 SGX enclave 中无法执行 syscall 指令；因此我们引入了 LibOS 技术，用于改善上述的软件兼容性问题，避免开发者在向 Intel® SGX Enclave 移植软件的过程中，去做复杂的软件适配工作。然后，虽然各个 LibOS 都在努力提升对系统调用的支持数量，但这终究难以企及原生 Linux 系统的兼容性，并且即使真的达成了这个目标，攻击面过大的缺点又会暴露出来。因此，Inclavare Containers 通过支持 Java 等语言 Runtime 的方式，来补全和提升 Enclave 容器的泛用性，而不是将 Enclave 容器的泛用性绑定在“提升对系统调用的支持数量”这一单一的兼容性维度上；此外，提供对语言 Runtime 的支持，也能将像 Java 这样繁荣的语言生态引入到机密计算的场景中，以丰富机密计算应用的种类和数量。

3、通过定义通用的 Enclave Runtime PAL API 来接入更多类型的 Enclave Runtime，比如 LibOS 就是一种 Enclave Runtime 形态；设计这层 API 的目标是为了繁荣 Enclave Runtime 生态，允许更多的 Enclave Runtime 通过对接 Inclavare Containers 上到云原生场景中，同时给用户提供更多的技术选择。

应用场景

作为业界首个面向机密计算场景的开源容器运行时，Inclavare Containers 为ACK-TEE（ACK-Trusted Execution Environment）提供了使用机密容器的最佳实践。ACK-TEE 依托 Inclavare Containers，能够无缝地运行用户制作的机密容器镜像，并保持与普通容器相同的使用体感。ACK-TEE 可被广泛应用于各种隐私计算的场景，包括：区块链、安全多方计算、密钥管理、基因计算、金融安全、AI、数据租赁。

Enclave-CC: 进程级机密容器

项目位置链接

https://github.com/confidential-containers/enclave-cc

技术自身介绍

背景

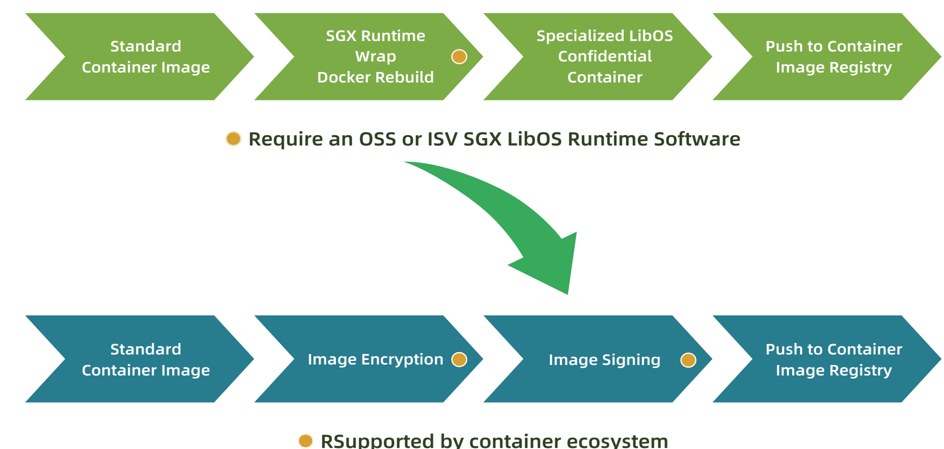
数据在整个生命周期有三种状态：At-Rest（静态）、In-Transit（传输中）和 In-Use（使用中）。在这个世界上，我们不断地存储、使用和共享各种敏感数据。保护处于所有状态中的敏感数据比以往任何时候都更为重要。如今被广泛使用的加密技术可以用来提供数据机密性（防止未经授权的访问）和数据完整性（防止或检测未经授权的修改），但目前这些技术主要被用于保护传输中和静止状态的数据，对数据的第三个状态“使用中”提供安全防护的技术仍旧属于新的前沿领域。Enclave-CC全链路安全（从部署到运行）方案能够解决用户在传输和使用中的安全， 有助于促进安全敏感度高的用户从私有部署迁移到公有云上。

问题&挑战

机密容器在部署阶段， 需要从租户侧部署到云端，在这个过程中，不能保证用户的镜像是机密的， 意味着承载用户应用的容器镜像是不安全的， 在上传到第三方的image registry有被第三方窥探的风险， 并不能保证镜像内容的安全。 而且需要针对于不同的enclave runtime制作特定的镜像。 这些操作都会对普通用户造成额外的开发负担。

解决方案

Enclave-cc方案在部署阶段对用户的镜像进行加密和签名处理， 从而保证镜像的安全性和完整性， 防止在被第三方窥探和修改镜像的内容。 在生成容器镜像的时候， 用户只需要和制作普通容器镜像一样，不需要针对特定的libos做额外定制， 对于支持POSIX API的应用， 可以完全兼容的通过enclave-cc成功运行起来。



Enclave-cc方案的运行流程如下：

1、用户使用标准工具制作一个签名和/或加密的受保护的容器镜像，并上传到容器镜像仓库中。

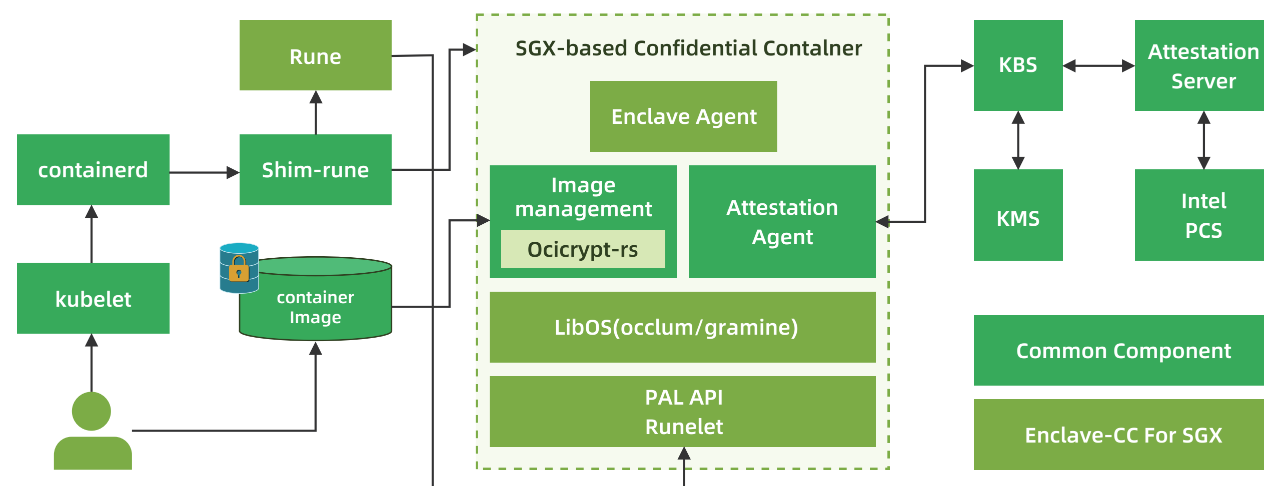
2、用户命令 Kubernetes 启动这个受保护的容器镜像。kubelet 会向 containerd 发起创建 Pod 的 CRI 请求，containerd 则把请求转发给 shim-rune，最终调用 rune 创建实际的 Pod。

3、CPU TEE 执行初始化，最终启动 enclave-agent 监听后续请求。

4、kubelet 向 containerd 发起 Image Pulling 的 CRI 请求，containerd 则把请求转发给 shim-rune，最终 enclave-agent 收到请求并通过 image-rs 子模块提供的容器镜像管理功能，在 TEE 内安全地执行拉取、验签、解密、unpack 以及挂载容器镜像的操作。

5、如果 TEE 内没有预先在 boot image 中内置验签或解密容器镜像的相关策略文件或密钥，则 image-rs 子模块会请求 attestation-agent 组件通过远程证明协议与远端可信的远程证明服务进行基于CPU TEE 硬件的身份认证与授权，通过attestation-agent 与远程证明服务建立的安全可信信道返回 image-rs 子模块需要的敏感数据。

6、远程证明服务验证 CPU TEE 硬件认证证据的完整性和真实性。起到比较验证作用的可信参考值由机密计算软件供应链安全基础设施来下发。如果 CPU TEE 通过了身份认证，远程证明服务将授权密钥管理服务（KMS ）返回 attestation-agent 请求的敏感数据，比如容器镜像的解密密钥和加密引导过程中用到的磁盘解密密钥。



Enclave-cc为云原生用户提供了一种机密容器的部署和运行方案。 能够让用户在使用机密容器的时候具有和使用普通容器一样的体感， 用户不需要针对于机密容器的场景修改应用, 在部署的时候不需要额外的操作步骤。

解决方案

Solution

Intel Confidential Computing Zoo: Intel机密计算开源解决方案

项目位置链接

https://github.com/intel/confidential-computing-zoo

技术自身介绍

问题&挑战

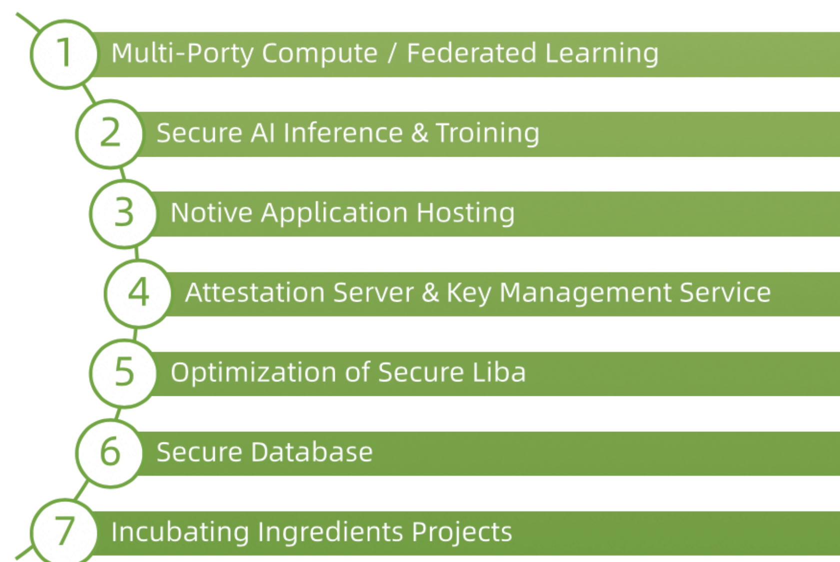
目前，机密计算还处于早期阶段，用户对SGX和TDX技术的了解和使用还需要进一步的加深和推广，对特定应用场景下如何结合Intel TEE技术以及其他安全技术打造全链路的机密计算方案缺少相应的设计参考

解决方案

为了帮助用户快速了解和使用Intel TEE技术，便捷快速地设计和开发出相关机密计算安全解决方案，Intel发起并开源了Confidential Computing Zoo(CCZoo)， CCZoo集合了不同的安全技术栈，提供了基于不同应用场景下的各种典型端到端的安全解决方案的参考案例，这些参考案例贴近实际的商业案例，让用户增加在机密计算方案设计和实现的体验，同时，引导用户结合相应的参考案例，快速设计和实现出自己特定的机密计算方案。此外，CCZoo还会选择在不同公有云，例如阿里云ECS，部署和验证已发布的端到端的机密计算方案，为用户提供相关的云上部署特有的配置参考信息，帮助用户方案快速上云。

技术介绍

CCZoo当前提供了6个类别的机密计算场景和1个孵化期项目集，具体分类如下图所示。为了方便部署，大部分的方案采用了容器化的集成。在不同的机密计算场景下，CCZoo选择使用了不同的安全技术组件模块，主要包括：Runtime Security、LibOS、Remote Attestation、KMS、TLS。用户可以通过对不同参考案例的部署增加对不同安全组件的使用和选择。



• Runtime Security: 当前主要使用Intel SGX，基于应用程序级别的隔离，对用户的应用和数据进行保护。

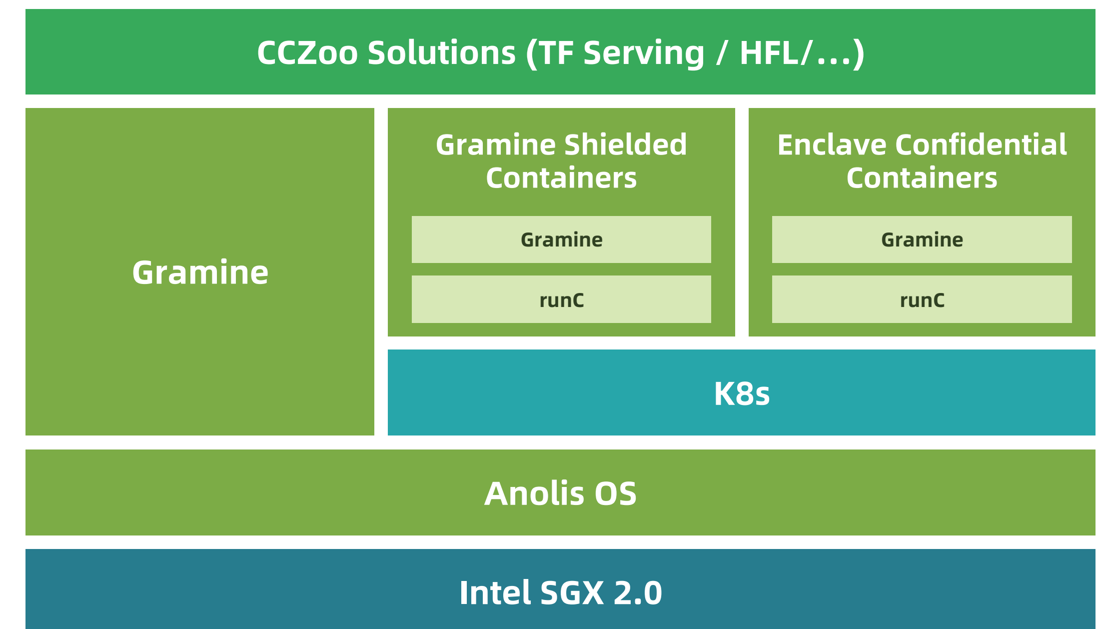
• LibOS: Gramine和Occlum。通过LibOS运行现有应用程序，只需进行细微修改或无需修改，即可在Intel SGX Enclave中运行。

• Remote Attestation: 提供了具体RA-TLS能力集成的gRPC框架，方便用户将远程认证的功能需要集成到自有框架中。

• KMS: 提供了集成远程认证加密钥管理的功能模块。

• TLS: 集成了RA-TLS功能，并启用了证书验证机制。

CCZoo将会跟Anolis Cloud Native Confidential Computing(CNCC) SIG展开合作，将CCZoo中不同机密计算场景下的方案，结合Anolis的技术框架进行实现和部署，如下图所示。



应用场景

CCZoo中多场景下的机密计算方案可以帮助CNCC SIG 进一步丰富用户基于Anolis的安全方案使用场景，为用户提供了最佳实践。同时，CCZoo也会基于方案级别，对Anolis进行全栈式的测试和验证，从用户实际使用的角度，来验证Anolis的可靠性和功能性。

部署TensorFlow Serving在线推理服务

概述

本文介绍在Intel® SGX使能的平台，基于Anolis OS部署TensorFlow Serving在线推理服务的技术架构和使用流程。

背景信息

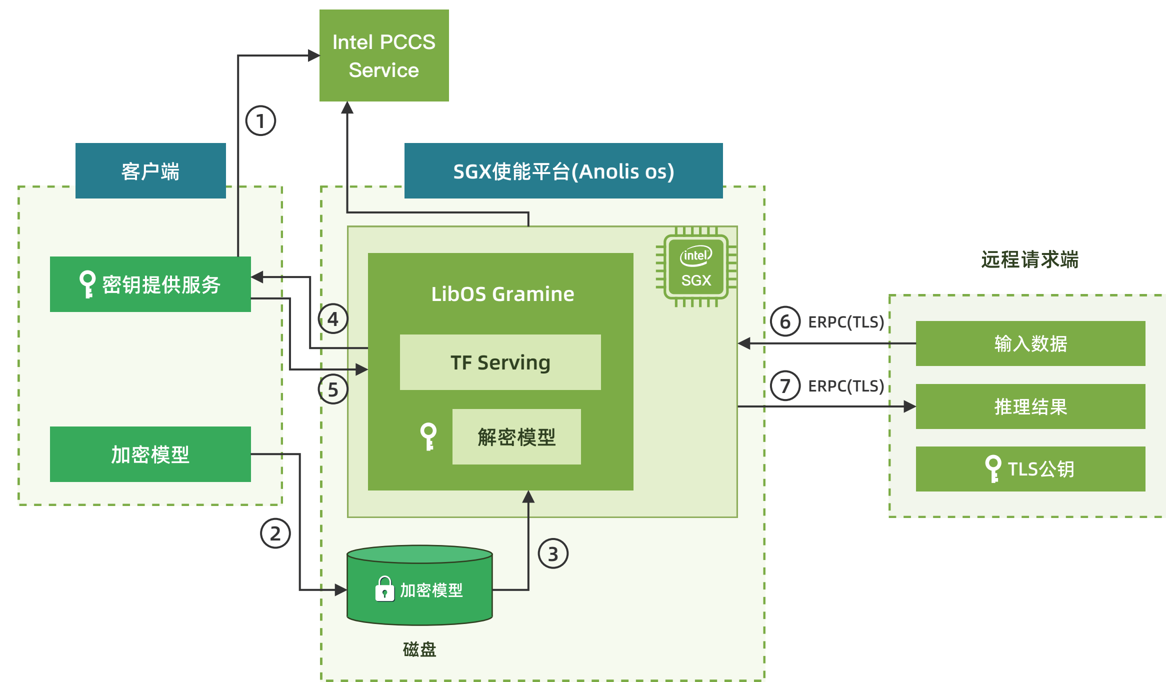
TensorFlow Serving是Google开源的机器学习平台TensorFlow生态的一部分，它的功能是将训练好的模型运行起来，提供接口给其他服务调用，以便使用模型进行推理预测。 阿里云ECS部分安全增强型实例基于Intel® SGX（Software Guard Extension）技术，提供了SGX加密计算能力，打造了基于硬件级别的更高安全等级的可信机密环境，保障关键代码和数据的机密性与完整性不受恶意软件的破坏。

将TensorFlow Serving在线推理场景部署在阿里云安全增强型实例可信机密环境中，可以保证数据传输的安全性、数据盘的安全性、数据使用的安全性、以及AI在线推理应用程序的完整性。 本实践为开发者使用Anolis OS提供了参考实现，您可以了解以下内容：

- 对基于SGX加密技术实现的端到端的全数据生命周期安全方案有整体了解。
- 对于使用TensorFlow Serving的开发者，可直接参照本实践部署和开发脚本。
- 为使用安全增强型实例的SGX开发者提供可行性参考框架和脚本，您可根据本实践步骤快速了解安全增强型实例以及环境搭建部署流程，轻松上手使用。

技术架构

本实践技术架构如下所示。



本实践涉及三个角色：客户端、Anolis OS SGX端、远程请求端。

- 客户端：客户端机器会将训练完的模型以及用来建立安全连接的TLS证书进行加密，并将这些加密文件上传到Anolis OS SGX端环境中。同时，客户端机器还将部署密钥提供服务，主要用来对 SGX端进行认证，确保运行在云上的TensorFlow Serving推理服务应用的完整性及SGX环境的可行性。验证通过后，会将密钥发送给在OS SGX中运行的TensorFlow Serving推理服务。
- Anolis OS SGX端：Anolis OS SGX端提供SGX机密计算环境，TensorFlow Serving推理服务运行在SGX环境中。当推理服务启动时，会向客户端发送远程认证请求，证明当前SGX环境的可行性和AI推理服务的完整性。验证成功后，推理服务会拿到客户端发送的密钥，并对加密的模型和TLS证书进行解密，至此，运行在SGX环境中的推理服务成功运行，并等待远程访问请求。
- 远程请求端：第三方使用者通过网络安全传输，将数据发送到运行在SGX机密计算环境中的推理服务。推理完成后，得到返回结果。

说明：本实践将客户端和远程请求端部署在同一台机器，Anolis OS SGX端部署在另外一台机器。

本实践使用到的其他组件如下：

- LibOS：Gramine是一款轻量级LibOS，结合Intel® SGX加密保护技术，提供了内核能力定制，运行资源消耗少，具备非常好的ABI兼容性，极大降低了原生应用移植到SGX环境的成本，做到了应用程序不修改或者极少的修改便能运行在SGX环境中。本实践使用Gramine封装TensorFlow Serving推理服务，将推理服务简单便捷地运行在SGX实例中。更多信息，请参见Gramine。
- AI推理服务：TensorFlow Serving是Google开源的机器学习平台TensorFlow生态的一部分，它的功能是将训练好的模型运行起来，提供接口给其他服务调用，以便使用模型进行推理预测。更多信息，请参见Tensor-Flow。
- Docker容器引擎：为了方便部署推理服务，本实践采用将推理服务运行在Container中的方式，利用Docker的命令方式运行推理服务。

本实践技术架构说明如下：

- Intel证书缓存服务PCCS（ProvisioningCertificate Caching Service）。如技术架构中①所示，Anolis OS SGX端需要向Intel PCCS获取PCK（Provisioning Certification Key）证书，Intel SGX会有一个密钥用于Enclave的签名，该密钥对于处理器或者平台是唯一的，密钥的公开部分就是PCK公钥。另外客户端也会向PCCS获取一些信息，比如TCB信息、Quote Enclave认证的信息、CRL信息等用于对SGX Enclave的认证。
- 搭建好Anolis OS SGX端后，可以将本地加密的模型文件以及TLS证书通过网络传输放到云盘中备用，如技术架构中②所示。
- 通过LibOS Gramine启动TensorFlow Serving推理服务时，会加载加密的模型文件，如技术架构中③所示。
- Gramine本身集成了远程认证的功能，在识别到有加密文件加载时，会转向配置好的远程IP服务发送认证请求，如技术架构中④所示。本实践在实现时，以一台阿里云实例模拟客户端，同时也作为远程访问端，另一台实例作为SGX环境。
- 在客户端的密钥提供服务对Anolis OS SGX端中的Enclave Quote认证成功后，会将模型加密的密钥发送给Gramine，如技术架构中⑤所示。由于此时Gramine是运行在Enclave中，因此Gramine拿到密钥对模型解密的过程是安全的。• 通过LibOS Gramine启动TensorFlow Serving推理服务时，会加载加密的模型文件，如技术架构中③所示。
- 在模型解密后，TensorFlow Serving便可以正常运行，并等待远端的访问请求。为了建立通信安全连接通道，远程访问端有TLS的公钥，在建立连接后，会对TensorFlow Serving中的TLS证书进行校验。如技术架构中⑥所示。
- 当TensorFlow Serving对远程请求端的数据推理完成后，便会通过建立的安全通道将推理结果返回给请求端，如技术架构中⑦所示。

步骤一：部署客户端

本实践运行的环境信息参考：

- 规格：加密内存≥8G
- 镜像： Ubuntu20.04
- 公网IP

• 安装SGX软件栈

1、环境配置

安装所需的mesa-libGL软件包。

```
sudo pip3 install --upgrade pip
sudo pip install multidict
sudo yum install mesa-libGL
```

2、下载软件包

下载本实践所用的TensorFlow Serving脚本代码并安装所需的argparse、aiohttp、tensorflow等软件包。

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/cczoo/tensorflow-serving-cluster/ tensorflow-serving/docker
/client/
pip3 install -r ./requirements.txt
```

3、下载模型

```
./download_model.sh
```

下载训练好的模型文件将会存放在创建的 models/resnet50-v15-fp32 目录下。

4、模型格式转换

为了兼容TensorFlow Serving，需要对训练好的模型文件进行格式转换。

```
python3 ./model_graph_to_saved_model.py --import_path `pwd -P`/models/res
net50-v15-fp32/resnet50-v15-fp32.pb --export_dir `pwd -P`/models/resnet50-v15-fp32
--model_version 1 --inputs input --outputs predict
```

转换好的模型文件将会存放在models/resnet50-v15-fp32/1/saved_model.pb。

5、创建gRPC TLS证书

本实践选择 gRPC TLS 建立客户端和TensorFlow Serving之间的通信连接，并设置 TensorFlow Serving 域名来创建单向 TLS Keys 和证书，用来建立安全通信通道。该脚本将会创建 ssl_configure 文件夹，里面包含server和client相应的证书。

```
service_domain_name=grpc.tf-serving.service.com
client_domain_name=client.tf-serving.service.com
./generate_twoway_ssl_config.sh ${service_domain_name} ${client_domain_name}
```

6、创建加密模型

```
mkdir plaintext/
mv models/resnet50-v15-fp32/1/saved_model.pb plaintext/
LD_LIBRARY_PATH=./libs ./gramine-sgx-pf-crypt encrypt -w files/wrap-key -i  plaintext/saved_
model.pb -o  models/resnet50-v15-fp32/1/saved_model.pb
```

7、启动密钥验证服务。

本实践使用Gramine提供的secret_prov_server_dcap作为远端SGX Enclave Quote认证服务，底层依赖调用SGX DCAP提供的Quote相关的认证库，该认证服务会向阿里云PCCS获取Quote认证相关的数据，比如TCB相关信息以及CRL信息等。

SGX Enclave Quote验证成功后，会将当前目录下存放的密钥files/wrap-key发送到远端应用。这里远端应用为vSGX环境中的Gramine，Gramine拿到wrap-key中的密钥后，便会对加密的模型和TLS配置文件进行解密。

• a. 切换到secrec_prov_server目录

```
./download_model.sh
```

• b. 使用密钥验证服务镜像

I) 下载密钥验证服务镜像

```
sudo docker pull intelcczoo/tensorflow_serving:anolis_secret_prov_server_latest
```

II) 根据脚本编译镜像

```
sudo ./build_secret_prov_image.sh
```

• c. 获取secret_prov_server镜像ID

```
sudo docker images
```

• d. 启动密钥验证服务

```
sudo ./run_secret_prov.sh -i secret_prov_image_id -a pccs.service.com:ip_addr
```

服务启动后便会在后台运行等待远程认证访问。当接收到远端认证后，认证通过会将密钥发送回远端。

• e. 查看secret_prov_server容器IP地址

```
sudo docker ps -a   #查看secret_prov_server镜像ID
sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <secret_prov_
server_container_id>
#<secret_prov_server_container_id>需修改为实际的secret_prov_container_id
```

步骤二：部署Anolis OS SGX端

1、下载本实践所用的TensorFlow Serving脚本代码

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/cczoo/tensorflow-serving-cluster/tensorflow-serving/docker/
tf_serving/
```

2、复制客户端的 ssl_configure 和 models 目录到Anolis OS SGX中的 tf_serving 目录中

```
scp -r tf@192.168.XX.XX:/<Tensorflow_Serving>/client/models  <Tensorflow_Serving>/dock
er/tf_serving
scp -r tf@192.168.XX.XX<Tensorflow_Serving>/client/ssl_configure  <Tensorflow_Serving>/
docker/tf_serving
```

3、创建TensorFlow Serving镜像

用户可以通过下面任意方式获取TensorFlow Serving镜像：

• a. 切换到secrec_prov_server目录

```
sudo docker pull intelcczoo/tensorflow_serving:anolis_tensorflow_serving_latest
```

• b. 自行编译TensorFlow Serving镜像

```
sudo ./build_gramine_tf_serving_image.sh image_tag
```

4、配置域名访问

```
sudo sh -c 'echo "remote_ip attestation.service.com" >> /etc/hosts'  #remote_ip请修改为客户端IP
```

说明:当客户端与vSGX端部署在同一台ECS实例上， remote_ip 为容器IP

5、运行TensorFlow Serving

```
cp ssl_configure/ssl.cfg .
sudo ./run_gramine_tf_serving.sh -i ${image_id} -p 8500-8501 -m resnet50-v15-fp32 -s ssl.cfg
-a attestation.service.com:remote_ip
```

说明： ${image_id} 需修改为TensorFlow Serving的 image id 。-p 8500-8501为TensorFlow Serving对应主机的端口。 remote_ip 需修改为 secret prov server 所在机器的IP或者容器IP（TF Serving与secretprov sever位于同一台机器）

步骤三：远端访问

1、切换到客户端

```
cd confidential-computing-zoo/cczoo/tensorflow-serving-cluster/tensorflow-serving/docker
/client/
```

2、获取client容器镜像

用户可以通过下面任意方式获取client镜像：

• a. 下载TensorFlow Serving镜像

```
sudo docker pull intelcczoo/tensorflow_serving:anolis_client_latest
```

• b. 自行编译client镜像

```
sudo docker build -f client.dockerfile . -t intelcczoo/tensorflow_serving:anolis_client_latest
```

3、运行并进入client容器

```
sudo docker run -it --add-host="grpc.tf-serving.service.com:<tf_serving_service_ip_addr>"
intelcczoo/tensorflow_serving:anolis_client_latest bash
```

说明：如果client和TensorFlow Serving 部署在同一台机器，tf_serving_service_ip_addr 为TensorFlow Serving容器IP地址。

4、向TensorFlow Serving 发送远程访问请求

```
cd /client
python3 ./resnet_client_grpc.py -batch 1 -cnum 1 -loop 50 -url grpc.tf-serving.service.com:
8500-ca `pwd -P`/ssl_configure/ca_cert.pem -crt `pwd -P`/ssl_configure/client/cert.pem  -key
 `pwd -P`/ssl_configure/client/key.pem
```

请求成功后会打印传输数据，并输出性能数据。

部署TensorFlow横向联邦学习

1概述

本文介绍在Intel® SGX使能的平台，基于Anolis OS部署TensorFlow横向联邦学习。

2背景信息

横向联邦学习是一种分布式的机器学习模型训练方案，该方案联合多个参与方在数据不出域的前提下完成模型的训练，保证了数据安全。

Intel® SGX (Software Guard Extension) 技术，提供了SGX加密计算能力，打造了基于硬件级别的更高安全等级的可信机密环境，保障关键代码和数据的机密性与完整性不受恶意软件的破坏。

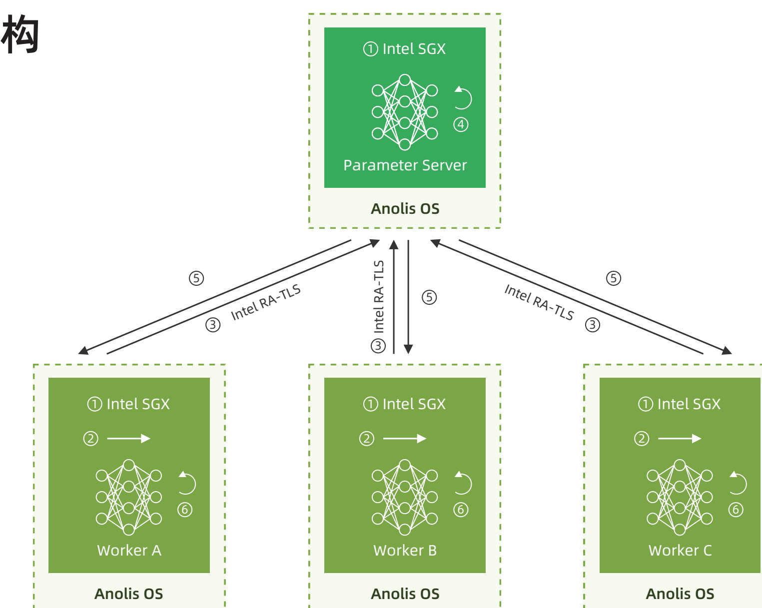
本实践为开发者使用Anolis OS提供了参考实现，您可以通过本文获取以下信息：

• 对基于SGX加密技术实现的端到端的全数据生命周期安全方案有整体了解。

• 对于使用Anolis OS的开发者，可直接参照本实践部署和开发脚本。

• 为使用安全增强型云服务器SGX开发者提供可行性参考框架和脚本，开发者可根据本实践步骤快速了解安全增强型裸金属云服务器及环境搭建部署流程，轻松上手使用。

3实践架构



本实践涉及了两种参与方：参数服务器端（parameter server）和客户端（worker）。

• 参数服务器端：存放模型参数，并利用客户端发来的梯度信息更新本地参数。

• 客户端：利用本地数据集，完成神经网络的前向传播和反向传播，并通过服务器端获取最新的模型参数。

说明：为了方便开发者部署，本实践将上述两种参与方部署在同一台云服务器中。

本实践使用到的主要组件：

• LibOS：Gramine是一款轻量级LibOS，结合Intel SGX加密保护技术，提供了内核能力定制，运行资源消耗少，具备非常好的ABI兼容性，极大降低了原生应用移植到SGX环境的成本，做到了应用程序不修改或者极少的修改便能运行在SGX环境中。更多信息，请参考Gramine。

• TensorFlow训练框架：TensorFlow是Google开源的机器学习平台，本实践采用TensorFlow的分布式训练框架作为横向联邦学习的训练框架。更多信息，请参考TensorFlow官网。

• Docker容器引擎：为了方便部署推理，本实践将三个参与方运行在Docker容器中，通过Docker的命令方式运行推理服务。

本实践架构说明： Anolis OS SGX端需要向Intel证书缓存服务（Provisioning Certificate Caching Service，PCCS）获取PCK（Provisioning Certification Key）证书。Intel SGX拥有一个密钥用于Enclave签名，该密钥对于处理器或者平台是唯一的，密钥的公开部分就是PCK公钥。另外客户端也会向PCCS获取一些信息，比如TCB信息、Quote Enclave认证的信息、CRL信息等用于对SGX Enclave的认证。

训练阶段可以分为以下几个步骤：

① 利用SGX平台，参与方运行在不同的Enclave中。

② 客户端根据其Enclave环境中的本地数据计算梯度信息。

③ 客户端通过RA-TLS向参数服务器发送梯度。

④ 参数服务器进行梯度聚合，计算并更新全局模型参数。

⑤ 参数服务器将模型参数发送给客户端。

⑥ 客户端更新局部模型参数。

训练过程中会不断重复步骤②-⑥。由于客户端和参数服务器运行在内存加密的Enclave环境中，同时RA-TLS通信方案保证了传输过程中的数据安全，因此该方案可以保证在完整的训练周期中的数据安全。

4 实践任务和配置

本实践提供图像分类和推荐系统两种训练任务类型。图像分类任务采用cifar-10数据集训练ResNet网络。推荐系统任务采用开源广告点击率数据集训练DLRM网络。

本实践的环境配置如下：

• 服务器配置：

I) 图像分类任务：单个节点加密内存：8G

II) 推荐系统任务：单个节点加密内存：32G

• 操作系统： anolisos:8.4-x86_64

• SGX软件栈

• Docker

5 实践部署

本实践提供两种部署方式：下载镜像和通过Dockerfile编译镜像。

5.1 下载镜像方式

5.1.1 下载Docker镜像

```
docker pull intelcczoo/horizontal_fl:anolis_sgx_latest
docker tag intelcczoo/horizontal_fl:anolis_sgx_latest anolisos_horizontal_fl:latest
```

5.1.2 启动Docker容器

下载实践代码

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/cczoo/horizontal_fl/
```

图像分类：

启动三个Docker容器（ps0、worker0、worker1）。如果在本地运行，请在 <PCCS ip addr> 中填写本地PCCS服务器地址。如果在云端运行，请在进入Docker容器后修改 /etc/sgx_default_qcnl.conf 文件中的PCCS服务器地址，填写云端的PCCS地址，忽略启动脚本中的 <PCCS ip addr> 参数。

```
./start_container.sh <ps0/worker0/worker1 > <PCCS ip addr> anolisos
```

推荐系统：

启动五个Docker容器（ps0、worker0、worker1、worker2、worker3）。如果在本地运行，请在 <PCCS ip addr> 中填写本地PCCS服务器地址。如果在云端运行，请在进入Docker容器后修改 /etc/sgx_default_qcnl.conf 文件中的PCCS服务器地址，填写云端的PCCS地址，忽略启动脚本中的 <PCCS ip addr> 参数。

```
./start_container.sh <ps0/worker0/worker1 > <PCCS ip addr> latest anolisos
```

5.1.3 编译应用

图像分类：

```
cd /image_classification
./test-sgx.sh make
```

推荐系统：

```
cd /recommendation_system
./test-sgx.sh make
```

编译过程中会生成MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN。

5.2 编译镜像方式

5.2.1 下载实践源码

在已创建好的SGX实例中，下载本实践所使用到的代码。

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/cczoo/horizontal_fl/
```

针对推荐系统任务，需要下载数据集，数据集保存在Google Drive中，您可以通过以下方式下载：

```
wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&con
firm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cookies--no-check
 -certificate 'https://docs.google.com/uc?export=download&id=
1xkmlOTtgqSQEWEi7ieHWYvlAl5bSthSr' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*/\1\n/p')
&id=1xkmlOTtgqSQEWEi7ieHWYvlAl5bSthSr" -O train.tar && rm -rf /tmp/cookies.txt
```

或者通过百度网盘下载。train.tar数据集文件需保存在recommendation_system/dataset目录下。

5.2.2 编译Docker镜像

可以通过参数 <image_classification/recommendation_system> 来指定编译图像分类任务或者推荐系统任务的应用程序。

```
./build_docker_image.sh <image_classification/recommendation_system> latest anolisos
```

5.2.3 启动Docker容器

图像分类：

启动三个Docker容器（ps0、worker0、worker1）。如果在本地运行，请在 <PCCS ip addr> 中填写本地PCCS服务器地址。如果在云端运行请在进入Docker容器后修改 /etc/sgx_default_qcnl.conf 文件中的PCCS服务器地址，填写云端的PCCS地址，忽略启动脚本中的 <PCCS ip addr> 参数。

```
./start_container.sh <ps0/worker0/worker1> <PCCS ip addr> latest anolisos
cd /image_classification
```

推荐系统：

启动五个Docker容器（ps0、worker0、worker1、worker2、worker3）。如果在本地运行，请在<PCCS ip addr> 中填写本地PCCS服务器地址。如果在云端运行请在进入Docker容器后修改 /etc/sgx_default_qcnl.conf 文件中的PCCS服务器地址，填写云端的PCCS地址，忽略启动脚本中的 <PCCS ip addr> 参数。

```
./start_container.sh <ps0/worker0/worker1/worker2/worker3> <PCCS ip addr> latest anolisos
d /recommendation_system
```

6 实践运行

6.1 图像分类

在多台服务器上部署不同分布式节点的情况下，可以通过修改Docker容器中/image_classification/目录下的train.py训练脚本来配置分布式节点IP地址：

```
tf.app.flags.DEFINE_string("ps_hosts", "['localhost:60002']", "ps hosts")
tf.app.flags.DEFINE_string("worker_hosts", "['localhost:61002','localhost:61003']", "worker hosts")
```

并在修改后重新编译应用：

```
cd /image_classification
./test-sgx.sh make
```

编译过程中会生成MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN。

配置Docker容器中/image_classification/下的dynamic_config.json文件，填入待通信方节点在编译应用阶段生成的MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN的值，如：

```
{
 "verify_mr_enclave" : "on",
   "verify_mr_signer" : "on",
   "verify_isv_prod_id" : "on",
   "verify_isv_svn" : "on",
   "sgx_mrs": [
      {
         "mr_enclave" : "1e4f3efafac6038dadaa94fdd248b93c82ae9f0a16642ff4bb07afe442aac
         56e",
         "mr_signer" : "5add213ac35413033647621e2fab91edcc8b82f840426803feb8a603be2ce
         8d4",
         "isv_prod_id" : "0",
         "isv_svn" : "0"
      }
   ]
}
```

修改完成后，在每个容器中运行相应的作业脚本。

```
./test-sgx.sh <ps0/worker0/worker1>
```

您可以从终端查看训练过程中的日志信息，以确认训练在正常进行。训练过程中生成的模型文件将保存在model文件夹中，其中变量值的相关信息存放在参与方 ps0 的 model/model.ckpt-data 中，计算图结构的相关信息存放在参与方 worker0 的 model/model.ckpt-meta 中。

6.2 推荐系统

在多台服务器上部署不同分布式节点的情况下，可以通过修改Docker容器中 /ecommendation_system/目录下的 ps0.py、worker0.py、worker1.py、worker2.py、worker3.py 训练脚本来配置分布式节点IP地址：

```
tf.app.flags.DEFINE_string("ps_hosts", "['localhost:70002']", "ps hosts")
tf.app.flags.DEFINE_string("worker_hosts", "['localhost:71002','localhost:71003','localhost:
71004','locaxlhost:71005']", "worker hosts")
```

并在修改后重新编译应用

```
cd /recommendation_system
./test-sgx.sh make
```

编译过程中会生成MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN。

配置Docker容器中 /recommendation_system/ 目录下的 dynamic_config.json 文件，填入待通信方节点在编译应用阶段生成的MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN的值，如：

```
{
  "verify_mr_enclave" : "on",
  "verify_mr_signer" : "on",
  "verify_isv_prod_id" : "on",
  "verify_isv_svn" : "on",
  "sgx_mrs": [
    {
      "mr_enclave" : "8b302bbf37ce27f82a3aa95b7daffe4b104e1faeb05e566dc8ded6ab0435
      9684",
      "mr_signer" : "5add213ac35413033647621e2fab91edcc8b82f840426803feb8a603be2ce
      8d4",
      "isv_prod_id" : "0",
      "isv_svn" : "0"
    }
  ]
}
```

worker节点仅需要与ps节点通信，因此只需要配置一组校验值。ps节点需要与所有worker节点通信，因此可能需要配置多组校验值。

修改完成后，在每个容器中运行相应的作业脚本。

```
./test-sgx.sh <ps0/worker0/worker1/worker2/worker3>
```

您可以从终端查看训练过程中的日志信息，以确认训练在正常进行。训练过程中生成的模型文件将保存在model文件夹中，其中变量值的相关信息存放在参与方ps0的model/model.ckpt-data中，计算图结构的相关信息存放在参与方worker0的model/model.ckpt-meta中。

部署隐私集合求交方案

1概述

本文介绍在Intel® SGX使能的平台，基于Anolis OS部署隐私集合求交方案。

2背景信息

隐私集合求交（Private Set Intersection，PSI）是多方安全计算的应用热点，其目的是通过安全方案计算两方之间的交集，而不暴露交集之外的其他信息。我们采用了基于Intel SGX技术的隐私保护计算解决方案。

Intel® SGX (Software Guard Extension) 技术，提供了SGX加密计算能力，打造了基于硬件级别的更高安全等级的可信机密环境，保障关键代码和数据的机密性与完整性不受恶意软件的破坏。

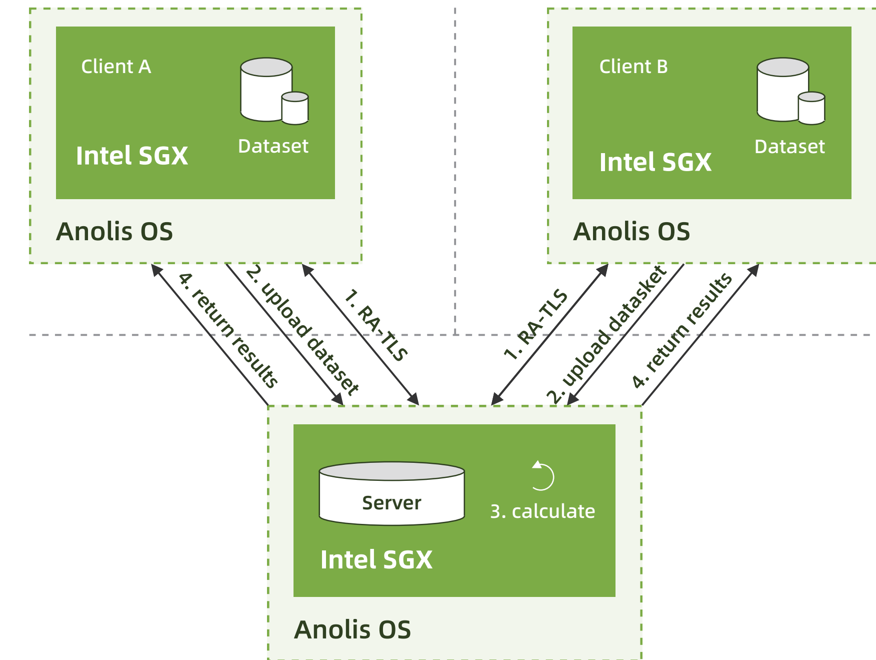
本实践为开发者使用Anolis OS提供了参考实现，您可以通过本文获取以下信息：

• 对基于SGX加密技术实现的端到端的全数据生命周期安全方案有整体了解。

• 对于使用Anolis OS的开发者，可直接参照本实践部署和开发脚本。

• 为使用安全增强型云服务器SGX开发者提供可行性参考框架和脚本，开发者可根据本实践步骤快速了解安全增强型裸金属云服务器及环境搭建部署流程，轻松上手使用。

3 实践架构



本实践涉及了两种角色：客户端（Client）和服务端（Server）。

• 服务端：部署在Anolis OS安全增强型云服务器中，提供SGX机密计算环境，隐私求交服务运行在此环境中。当隐私求交启动后，服务端会等待客户端发送远程认证请求，证明当前SGX环境的可信性。所有客户端验证成功后，服务端会等待客户端上传待求交数据，待所有数据上传成功后，服务端计算交集并将结果发送回各个客户端。

• 客户端：客户端向服务端请求远程验证服务，远程验证通过后，客户端将本地待求交数据上传到服务端，并等待服务端完成求交并将结果发送回客户端。

说明：为了方便开发者部署，本实践将隐私集合求交方案参与方部署在同一台云服务器中。

本实践使用到的主要组件：

• LibOS：Gramine是一款轻量级LibOS，结合Intel SGX加密保护技术，提供了内核能力定制，运行资源消耗少，具备非常好的ABI兼容性，极大降低了原生应用移植到SGX环境的成本，做到了应用程序不修改或者极少的修改便能运行在SGX环境中。更多信息，请参考Gramine。

• Docker容器引擎：为了方便部署推理，本实践将参与方运行在Docker容器中，通过Docker的命令方式运行推理服务。

本实践架构说明： Anolis OS SGX端需要向Intel证书缓存服务（Provisioning Certificate Caching Service，PCCS）获取PCK（Provisioning Certification Key）证书。Intel SGX拥有一个密钥用于Enclave签名，该密钥对于处理器或者平台是唯一的，密钥的公开部分就是PCK公钥。另外客户端也会向PCCS获取一些信息，比如TCB信息、Quote Enclave认证的信息、CRL信息等用于对SGX Enclave的认证。

训练阶段可以分为以下几个步骤：

① 所有参与者都在SGX环境中运行。每个客户端通过RA-TLS与服务端完成双向认证，以互相确认身份。

② 客户端通过RA-TLS增强型gRPC将数据安全传输到服务器。

③ 服务端等待所有客户端的数据上传完成后，计算上传数据的交集。

④ 服务端通过RA-TLS增强型gRPC将计算结果回传给每个参与的客户端。

由于客户端和参数服务器运行在内存加密的Enclave环境中，同时RA-TLS通信方案保证了传输过程中的数据安全，因此该方案可以保证在完整的训练周期中的数据安全。

4 实践任务和配置

本实践的环境配置如下:

- 服务器配置:
  - I) Anolis OS安全增强型云服务器
  - II) 单个节点加密内存：2G
- SGX软件栈
- Docker

5 实践部署

本实践提供两种部署方式：下载镜像和通过Dockerfile编译镜像。

5.1 下载镜像方式

5.1.1 下载Docker镜像

```
docker pull intelcczoo/horizontal_fl:anolis_sgx_latest
docker tag intelcczoo/horizontal_fl:anolis_sgx_latest anolisos_horizontal_fl:latest
```

5.1.2 启动Docker容器

下载实践代码

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/cczoo/psi/
```

如运行两方求交，需启动三个Docker容器（server，client1，client2）;如运行三方求交，需启动四个Docker容器（server，client1，client2，client3）。在每个终端运行如下命令以启动Docker容器:

```
./start_container.sh <server/client1/client2/client3> <PCCS ip> anolisos
```

其中 <server/client1/client2/client3> 字段表示在不同的节点分别启动server和client， <PCCS ip> 字段表示PCCS的IP地址。

5.1.3 配置PCCS信息

在每个Docker容器的/etc/sgx_default_qcnl.conf文件中配置PCCS信息，如:

```
PCCS_URL=https://sgx-dcap-server.cn-beijing.aliyuncs.com/sgx/certification/v3/
USE_SECURE_CERT=TRUE
```


5.2 编译镜像方式

5.2.1 下载实践源码

在已创建好的SGX实例中，下载本实践所使用到的代码。

```
git clone https://github.com/intel/confidential-computing-zoo.git
cd confidential-computing-zoo/
```

5.2.2 编译Docker镜像

```
cd cczoo/common/docker/gramine
./build_docker_image.sh anolisos anolisos
cd -
cd cczoo/psi/gramine
./build_docker_image.sh anolisos
```

5.2.3 启动Docker容器

如运行两方求交，需启动三个Docker容器（server，client1，client2）;如运行三方求交，需启动四个Docker容器（server，client1，client2，client3）。在每个终端运行如下命令以启动Docker容器

```
./start_container.sh <server/client1/client2/client3> <PCCS ip> anolisos
```

其中 <server/client1/client2/client3> 字段表示在不同的节点分别启动server和client，<PCCS ip> 字段表示PCCS的IP地址。

6 实践运行

本实践方案提供Python和C++两种版本。

6.1 Python版本

6.1.1 编译Python程序

在每个启动的Docker容器中编译程序:

```
cd /gramine/CI-Examples/psi/python
./build.sh
```

在多台服务器上部署不同分布式节点的情况下，需要配置 dynamic_config.json 文件，填入待通信方节点在编译应用阶段生成的 MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN 的值，如:

```
{
 "verify_mr_enclave" : "on" ,
   "verify_mr_signer" : "on" ,
   "verify_isv_prod_id" : "on" ,
   "verify_isv_svn" : "on" ,
   "sgx_mrs": [
     {
        "mr_enclave" : "1e4f3efafac6038dadaa94fdd248b93c82ae9f0a16642ff4bb07afe442aac
```

```
            56e" ,
            "mr_signer" : "5add213ac35413033647621e2fab91edcc8b82f840426803feb8a603be2ce
            8d4" ,
            "isv_prod_id" : "0" ,
            "isv_svn" : "0"
        }
    ]
}
```

6.1.2 运行两方求交程序

在每个Docker的容器中的 /gramine/CI-Examples/psi/python 路径下分别执行对应的参与方命令：

```
# Run the server
gramine-sgx python -u server.py -host localhost:50051 -config dynamic_config.json

# Run the client1
gramine-sgx python -u data_provider1.py -host localhost:50051 -config dynamic_config.json
-is_chief True -data_dir "data1.txt" -client_num 2

# Run the client2
gramine-sgx python -u data_provider2.py -host localhost:50051 -config dynamic_config.json
-is_chief False -data_dir "data2.txt" -client_num 2
```

每个客户端都会得到交集结果：

```
['car', 'cat', 'train']
```

6.1.3 运行三方求交程序

在每个Docker的容器中的 /gramine/CI-Examples/psi/python 路径下分别执行对应的参与方命令：

```
# Kill the previous server process
pkill -f gramine
# Run the server
gramine-sgx python -u server.py -host localhost:50051 -config dynamic_config.json

# Run the client1
gramine-sgx python -u data_provider1.py -host localhost:50051 -config dynamic_config.json
-is_chief True -data_dir "data1.txt" -client_num 3

# Run the client2
gramine-sgx python -u data_provider2.py -host localhost:50051 -config dynamic_config.json
-is_chief False -data_dir "data2.txt" -client_num 3

# Run the client3
gramine-sgx python -u data_provider3.py -host localhost:50051 -config dynamic_config.json
-is_chief False -data_dir "data3.txt" -client_num 3
```

每个客户端都会得到交集结果：

```
['train', 'car', 'cat']
```

6.2 C++版本

6.2.1 编译C++程序

在每个启动的Docker容器中编译程序：

```
cd /gramine/CI-Examples/psi/cpp
./build.sh
```

在多台服务器上部署不同分布式节点的情况下，需要配置 dynamic_config.json 文件，填入待通信方节点在编译应用阶段生成的MR_ENCLAVE，MR_SIGNER，ISV_PROD_ID，ISV_SVN的值，如：

```
{
 "verify_mr_enclave" : "on" ,
   "verify_mr_signer" : "on" ,
   "verify_isv_prod_id" : "on" ,
   "verify_isv_svn" : "on" ,
   "sgx_mrs": [
      {
         "mr_enclave" : "1e4f3efafac6038dadaa94fdd248b93c82ae9f0a16642ff4bb07afe442aac
         56e" ,
         "mr_signer" : "5add213ac35413033647621e2fab91edcc8b82f840426803feb8a603be2ce
         8d4" ,
         "isv_prod_id" : "0" ,
         "isv_svn" : "0"
      }
   ]
}
```

6.2.2 运行两方求交程序

在每个Docker的容器中的 /gramine/CI-Examples/psi/cpp 路径下分别执行对应的参与方命令：

```
# Kill the previous server process
pkill -f gramine
# Run the server
cd runtime/server
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json

# Run the client1
cd runtime/data_provider1
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json -is_chief=true -client_
num=2 data_dir="data1.txt" client_name="data_provider1"

# Run the client2
cd runtime/data_provider2
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json -is_chief=false -client_
num=2 data_dir="data2.txt" client_name="data_provider2"
```


OpenAnolis
龙 蜥 社 区

每个客户端都会得到交集结果：

```
car
cat
train
```

6.2.3 运行三方求交程序

在每个Docker的容器中的 /gramine/CI-Examples/psi/cpp 路径下分别执行对应的参与方命令：

```
# Kill the previous server process
pkill -f gramine
# Run the server
cd runtime/server
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json

# Run the client1
cd runtime/data_provider1
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json -is_chief=true-client_
num=3 data_dir="data1.txt" client_name="data_provider1"

# Run the client2
cd -
cd runtime/data_provider2
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json -is_chief=false-client_
num=3 data_dir="data2.txt" client_name="data_provider2"

# Run the client3
cd -
cd runtime/data_provider3
gramine-sgx grpc -host=localhost:50051 -config=dynamic_config.json -is_chief=false-client_
num=3 data_dir="data3.txt" client_name="data_provider3"
```

每个客户端都会得到交集结果：

```
car
cat
train
```


PPML：端到端隐私保护机器学习解决方案

项目位置链接

https://github.com/intel-analytics/BigDL

https://github.com/intel-analytics/BigDL/tree/main/ppml

技术自身介绍

领域的问题和挑战

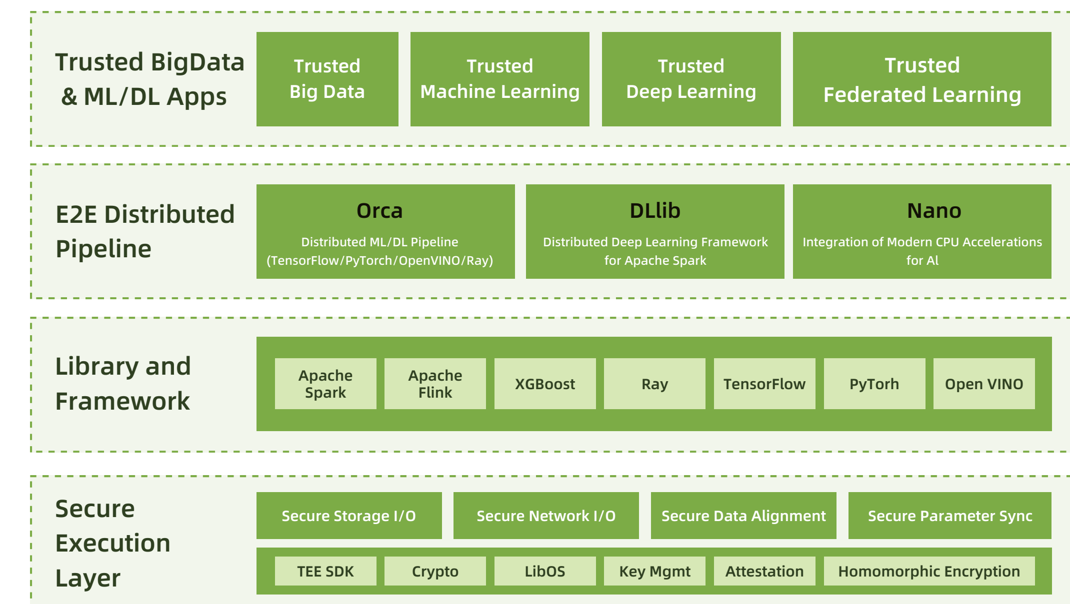
在众多计算应用场景中，大数据分析和人工智能已经成为不可或缺的关键环节。总体而言，数据越多，数据分析的价值越大，ML/DL的模型也会越完善。但囤积和处理海量数据也带来了隐私、安全和监管等风险。隐私保护机器学习（PPML，包含大数据分析和人工智能）有助于化解这些风险，能够在不透露原始数据的前提下，实现数据的有效流动，让使用方利用数据的价值。

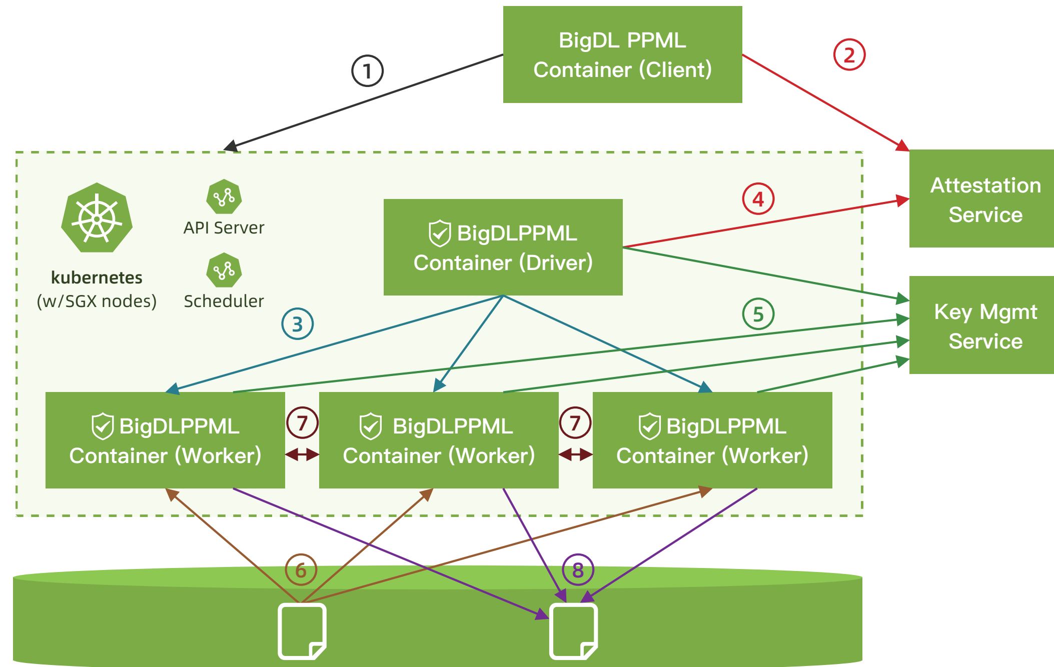
对挑战的解决方案BigDL PPML

英特尔至强可扩展处理器为隐私保护机器学习提供了可信硬件执行环境 - 英特尔® SGX和英特尔®TDX。英特尔基于这些技术打造了端到端的大数据和人工智能隐私计算解决方案BigDL PPML。

BigDL是英特尔开源的统一的人工智能解决方案平台，数据科学家，数据工程师等开发者可以使用BigDL轻松创建端到端的分布式人工智能应用。BigDL应用英特尔SGX/TDX可信硬件执行环境（Trusted Execution Environ[1]ment, TEE），并集成了其他软硬件安全措施，构建了一个分布式的隐私保护机器学习（Privacy Preserving Machine Learning, PPML）平台，能够保护端到端（包括数据输入，数据分析，机器学习，深度学习等各个阶段）的分布式人工智能应用。

技术介绍





上图是BigDL PPML预制工作流：

1、用户通过BigDL PPML命令行向Kubernetes提交任务，此操作会创建一个驱动节点

2、BigDL PPML客户端认证驱动节点

3、驱动节点创建更多的工作节点

4、驱动节点认证工作节点

5、驱动节点和工作节点从密钥管理服务获取密钥

6、工作节点读取输入数据并解密

7、工作节点分布式运行大数据，机器学习或者深度学习任务

应用场景

场景描述

1、在加密数据上开发并运行标准的分布式人工智能应用（如大数据分析、机器学习、深度学习等）。

2、利用基于硬件的安全技术技术（如英特尔®SGX/TDX）保护计算过程以及相应的内存数据。

3、为人工智能应用提供端到端的安全和隐私保护，例如：

- 在具备英特尔®SGX/TDX硬件能力的K8s环境中创建并认证可信的集群环境
- 通过密钥管理系统（Key Management System, KMS）为分布式数据提供加密和解密能力
- 通过英特尔®SGX/TDX，加解密技术，TLS，安全认证等技术实现安全的分布式计算和数据通信

应用效果

1、BigDL PPML 端到端隐私保护机器学习解决方案继承了可信执行环境 (TEE) 的优点。和传统数据安全解决方案相比，它的安全性和数据效用性十分突出，同时性能损耗较低。

2、相比普通大数据分析工作流，BigDL PPML预制了可信安全的工作流。BigDL PPML工作流包括了：基于SGX/TDX 的可信计算核心组件，支持 Apache Spark， Spark SQL 以及机器学习，深度学习等应用；经过抽象的认证服务客户端 API；经过抽象的密钥管理服务客户端 API；加密的数据传 输和存储；以及定制的 K8s 容器镜像。

3、使用BigDL PPML预制的工作流，开发者可以更加专注于业务逻辑的相关开发工作，利用 BigDL PPML 保障其应用的端到端安全性和隐私性。用户可以显著提高隐私计算应用的开发效率，大幅缩短实现隐私计算解决方案的开发时间 (Time to solution)。

4、通过应用该方案，企业能够构建端到端的安全保护流程，在数据输入、数据分析、机器学习、深度学习等 AI、大数据应用的多个阶段建立安全防护能力，避免安全威胁乘虚而入。同时，该方案实现了基于硬件底层的保护，具备更高的数据保护等级，能够防护传统安全方案难以抵抗的攻击形式，降低重要数据泄露的风险。

5、在该方案的支持下，企业能够提供安全的数据融通服务，联合分析、联合训练、联合预测等应用不透露原始数据以及基于数据应用逻辑层面的授权，保证场景化的数据融通安全需要；满足商业的自主性、可控性、安全性，为客户提供透明可控的安全流通环境，可随时管控和退出，永保数据控制权。

用户情况

- 蚂蚁集团安全团队适配了BigDL PPML, 为集团业务提供隐私大数据和机器学习的方案。
- 其他用户包括火山引擎，腾讯等。