The Certifier Framework for Confidential Computing

John Manferdelli, VMware

jmanferdelli@vmware.com

October 5, 2023

Barriers to Confidential Computing adoption

Motivating the certifier framework

Hardware diversity	Current software complexity

• TEE mechanisms (attestation protocols, core capabilities, interfaces) are not consistent across vendors	• Cooperating programs require flexible management infrastructure to support many program providers and different security requirements.
	  • Trust policy often embedded in program (Bad!)
	  • Policy difficult to write, understand or audit
	  • Can make deployment on different platforms difficult
	• Support code for trust management involves extensive security-sensitive logic and requires deep security expertise
	• No "hello world" exemplars for end-to-end secure system design that work in an hour.
	• Most bespoke code is not scalable and requires rewriting between platforms

Gap filled by Certifier: a standard, vendor-independent, easy-to-use framework for CC trust management for developers and deployers

CONFIDENTIAL COMPUTING CONSORTIUM

Confidential Computing software tedium

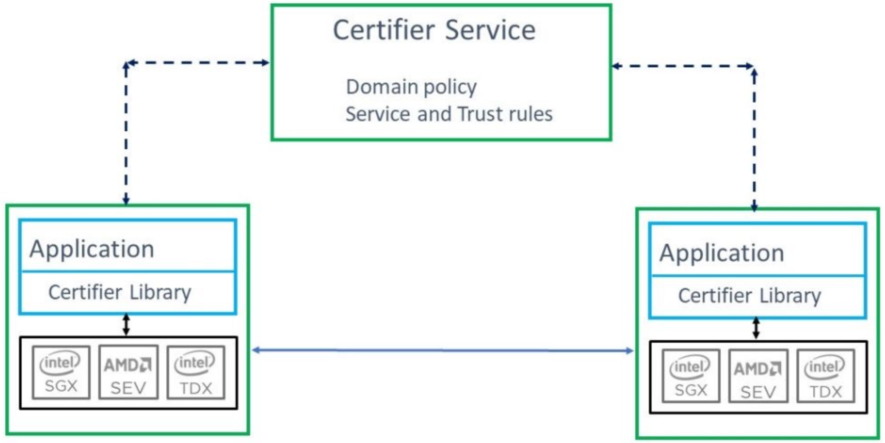
All that work…

- Generating, rotating and managing lots of keys
- Authoring, managing and enforcing program policy universally understood by all “trusted programs”
- Binding security policy to pre-authored program
- Verifying policy compliance with absolute assurance
- Securely storing and recovering secrets and data
- Securely communicating with other unforgeably identified Confidential Computing Programs
- Operating on different Confidential Computing platforms without application changes
- Rapid CC enablement of existing “well written” programs
- Preserving existing deployment models
- Providing scalable support managing related distributed components (including upgrade and new components)
- Enabling features with secure code and appropriate, agile logging, encryption and authentication primitives

Existing SDKs (Gramine, OE) help the CC developer but largely focus on orthogonal issues

The certifier framework

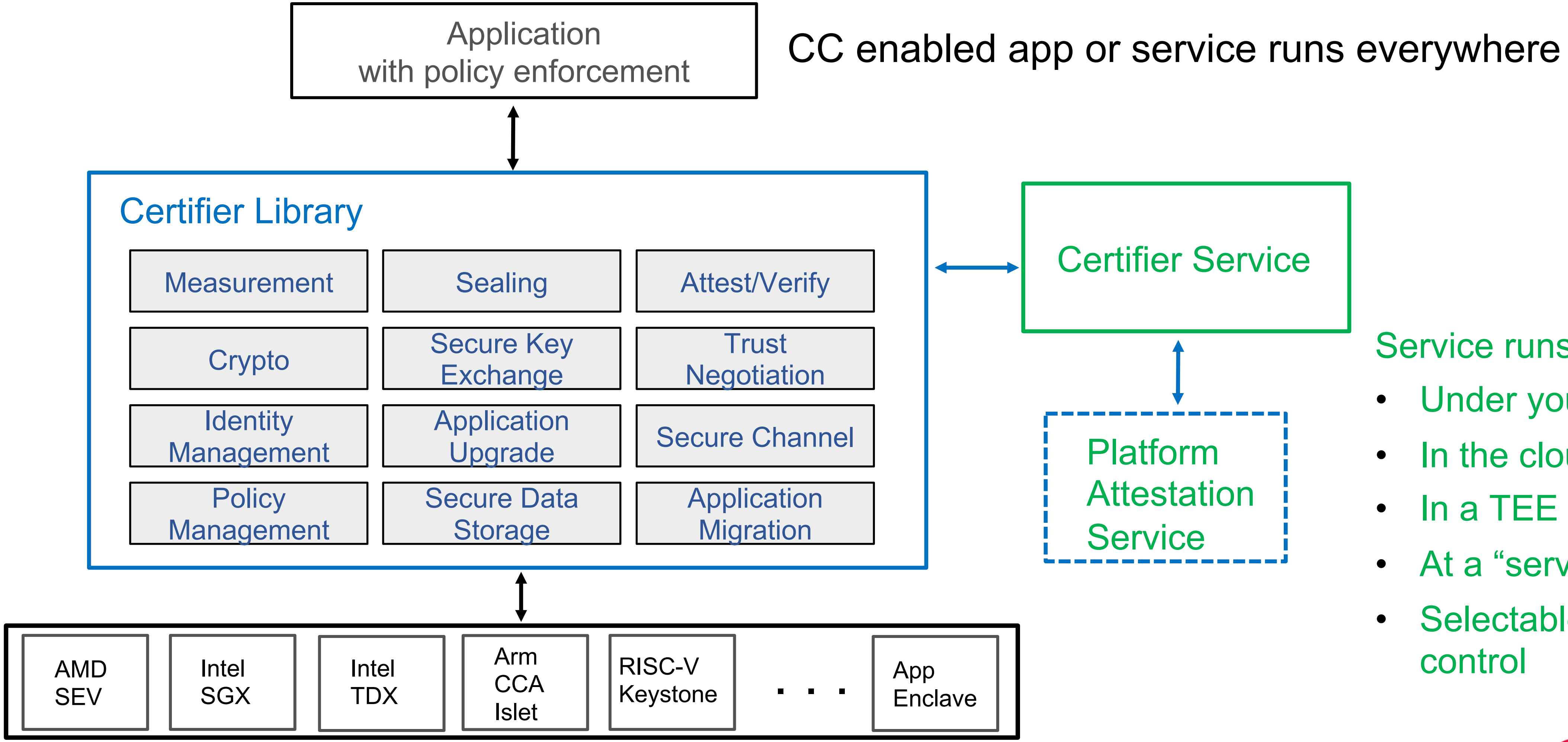
Design goals



- Open-source community project
- Vastly decreases time to build applications
- Platform independent; avoid need to port applications
  - Already supports SGX (via Open Enclaves or Gramine), AMD-SEV-SNP, Arm CCA, RISC-V Keystone, and soon TDX. Also supports recursive enclaves (applications within VM's)
  - Supports VM and application protection
- Provides simple API for most applications but complete support for "edge" cases
- Avoids dependency on a central authority or differences in deployment models
- Secure, high-availability deployment with "embarrassingly parallel" server support
- Easy to use infrastructure to support policy management
  - No deployment changes as policies evolve
  - Audited, comprehensible, declarative policy and verification
  - Efficient introduction of new components and upgrades
  - Scalable management
- Enable new services and use models for Data Economy, high security applications and regulatory regimes

Certifier Framework Architecture

Taking the devil out of the details



Service runs anywhere:

- Under your desk
- In the cloud
- In a TEE or not
- At a "service provider"
- Selectable security control

Certifier Framework Concepts and API

Key Concepts:

Security Domain	Identified by a public key associated with all application code within a trusted environment. Programs are in a primary domain but can certify to secondary domains.
Certification	Refers to the verification of all properties in the security domain (including program identity, involving attestation) resulting in an x509 certificate for trust.
Trust and Policy	Should not be hard-coded in an application which should be able to operate in compliance in different security domains. Don't complicate program development or deployment.

C++ Classes:

`cc_trust_manager`	Basic interface to establish keys, policy and manage certification with the Certifier Service.
`secure_authenticated_channel`	Establishes secure channel with an "authenticated program in security domain
`policy_store`	Stores policy, keys, communications endpoints securely. Additional helper function APIs for complicated applications.

Additional helper function APIs provided for use by more complicated applications.

Simple API: Most applications will use exactly this (and nothing else)

```
string public_key_alg("rsa-2048"); string symmetric_key_alg("aes-256-gcm");
cc_trust_manager trust_mgr(“sev-enclave”, “authentication”, “store”);

trust_mgr.initialize_enclave(NULL); // init enclave
trust_mgr.init_policy_key(initialized_cert_size, initialized_cert)
trust_mgr.cold_init(public_key_alg, symmetric_key_alg, …);
trust_mgr.certify_me(); // Get admission certificate

secure_authenticated_channel channel(“client”);
channel.init_client_ssl(host, port, policy-key, auth_key, admissions-cert);

// Your application here
```

Certifier: “Simple Example” App

```
Running App as server
    Client peer id is Measured
    8522d8e996e0089b26cb5dde06341c728e3017fa78974e3dce364bef56bdd307
    SSL server read: Hi from your secret client
```

```
Running App as client
    Server peer id : Measured
    8522d8e996e0089b26cb5dde06341c728e3017fa78974e3dce364bef56bdd307
    SSL client read: Hi from your secret server
```

You may want to add:

- API serialization
- ACLs for differentiated access
- Standard “distributed programming” primitives like Byzantine agreement

Works on all CC platforms:

- Simulated enclave
- AMD SEV-SNP
- Intel SGX (Gramine and OE)
- Arm CCA (Samsung Islet)
- RISC-V Keystone
- Intel TDX (when it’s ready)
- **Application enclave service**

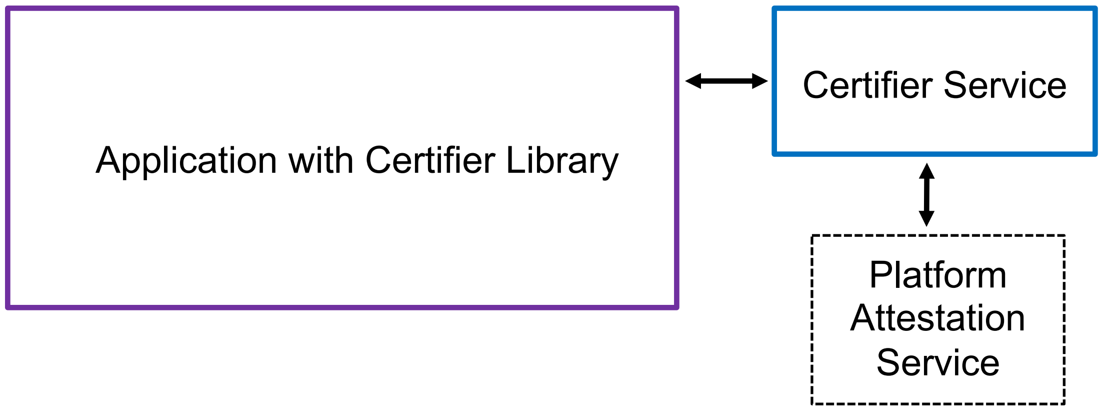
Certifier Framework API: Observations

1. **Simple** for most applications (and **adequate** for complex ones)
2. Abstracts underlying **isolate**, **measure**, **seal/unseal**, and **attest** primitives
3. Provides a **secure store** for save/recovery operations (in one statement!).
4. Includes mechanism to establish a **secure channel** within security domain
    - Encrypted, integrity protected, bi-directional, with authenticated trusted enclave named by their measurements

5. Almost as simple as **“Hello world”**

Certifier Service

“Centralized” management for a security domain



- Policy-driven (rooted in key associated with application)
- Admits new components without changing old ones
- Application upgrade without changing other applications
- Facilitates data migration and sharing in a domain
- Enforce security domain wide policy
- Machine capabilities
- Revocation

Scalable, resilient deployment supporting all environments

Policy example

1. Key[rsa, policyKey, a5fc2b7e629fbbfb04b056a993a473af3540bbfe] is-trusted

2. Key[rsa, policyKey, ...] says Measurement[a051a41593ced366462caea392830628742943c3e81892ac17b70dab6fff0e10] is-trusted

3. Key[rsa, policyKey, ...] says Key[rsa, platformKey, ...] is-trusted-for-attestation

4. Key[rsa, platformKey, ...] says Key[rsa, attestKey, ...] is-trusted-for-attestation

Proof from the Certifier Service

1. Key[rsa, policyKey, ...] is-trusted and Key[rsa, policyKey, ...] says Measurement[a051a41593ced366462caea392830628742943c3e81892ac17b70dab6fff0e10] is-trusted, imply via rule 3, Measurement[...] is-trusted
2. Key[rsa, policyKey, ...] is-trusted and Key[rsa, policyKey, ...] says Key[rsa, platformKey, ...] is-trusted-for-attestation, imply via rule 5, Key[rsa, platformKey, ...] is-trusted-for-attestation
3. Key[rsa, platformKey, ...] is-trusted-for-attestation and Key[rsa, platformKey, ...] says Key[rsa, attestKey, ...] is-trusted-for-attestation, imply via rule 5, Key[rsa, attestKey, ...] is-trusted-for-attestation
4. Key[rsa, attestKey, ...] is-trusted-for-attestation and Key[rsa, attestKey, ...] says Key[rsa, program-auth-key, ...] speaks-for Measurement[...], imply via rule 6, Key[rsa, program-auth-key, ...] speaks-for Measurement[...]
5. Measurement[...]is-trusted and Key[rsa, program-auth-key, ...] speaks-for Measurement[...], imply via rule 1, Key[rsa, program-auth-key, ...] is-trusted-for-authentication

Proved: Key[rsa, program-auth-key, ...] is-trusted-for-authentication

Platform Policy

Used to verify platform characteristics

- Key[rsa, policyKey, a5fc2b7e629fbbfb04b056a993a473af3540bbfe] says Key[rsa, ARKKey, aeb214025256a56863fe9aa9c9f1cca153af4416] is-trusted-for-attestation
- Key[rsa, policyKey, a5fc2b7e629fbbfb04b056a993a473af3540bbfe] says platform[amd-sev-snp, debug: no, migrate: no, api-major: >= 0, api-minor: >= 0, smt: no, tcb-version: = 3458764513820573973] has-trusted-platform-property

Supplants the need to use external attestation services and preserves privacy and control.

CONFIDENTIAL COMPUTING CONSORTIUM

Certifier Service: Observations

1. Service provides a **policy language**, **evidence formats**, and **policy evaluation** to determine when a Confidential Computing application should be trusted.

   Evidence submitted and evaluated includes platform attestation reports. Other formats converted to "canonical form."

2. Utilities to **generate keys** and **write policy**.
3. Checks **program** and **platform policy**
4. Issues "**Admissions Certificate for Security Domain**"

Feature roadmap

- Extended fields in tokens (Done)
- Python bindings (Almost done)
- GPU support (In progress)
- Example of running Certifier Service in TEE and provisioning it. (Done)
- Multi-security domain certification (Done)
- Encrypted clients (to protect code)
- Switch to smphost tools (https://virtee.io/)
- Nitro?
- Security review
- Rust Client
- Highly efficient differentiated access control

Community inputs and contributions welcome

Confidential Computing and the Certifier Framework

Verifiably secure operational properties, including confidentiality, integrity and policy compliance, no matter where program runs. Safe against malware and “insiders.”

Before CC: developer/deployer must:

1. Write applications correctly
2. Deploy the program safely (no changes)
3. Configure operating environment correctly
4. Ensure other programs can’t interfere with safe program execution
5. Generate and deploy keys safely
6. Protect keys during use and storage
7. Ensure data is not visible to adversaries and can’t be changed in transmission or storage
8. Ensure trust infrastructure is reliable
9. Audit to verify this all happened

- Consequence: App writer/deployer entirely reliant on provider for all security --- unverifiable

With CC: developer / deployer must:

1. Write the application correctly
   – For every backend
   – Manage migration
   – Support each providers deployment model
   – Implement all the crypto
   – Implement secure communications and storage
   – Make it scalable and upgradable
2. Implement the trust policy
   – Maintain trust policy
   – Different for every app/deployer
   – Make it scalable

- Consequence: You can have safe application but it’s platform dependent and a lot of work

With CC & Certifier Framework: developer/deployer must:

1. Write the application correctly using Certifier APIs
2. Write the trust policy
3. Use Certifier Service to manage it!

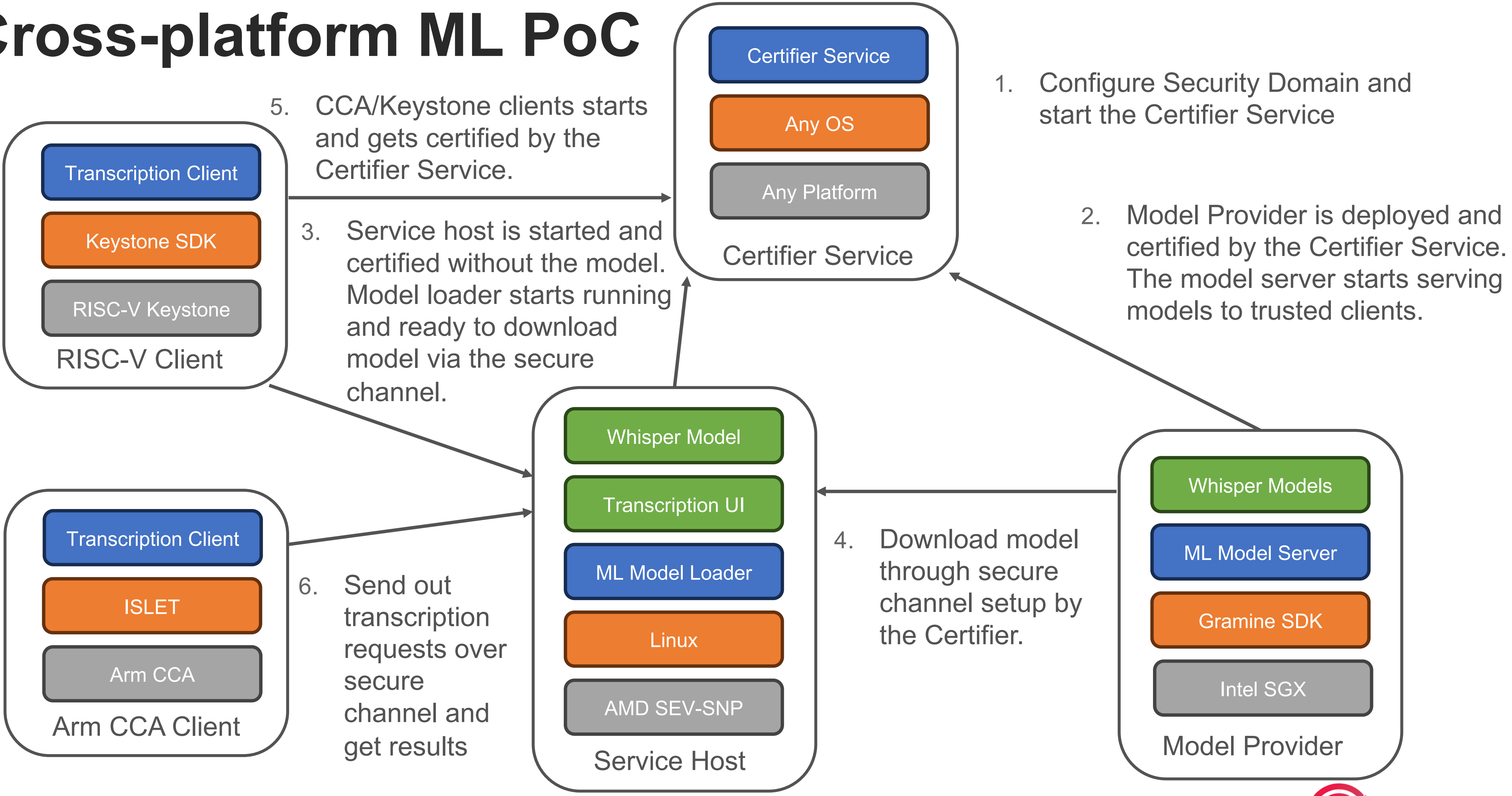
Consequence:

- You write the application once.
- Need only add a few dozen lines of code to enable CC protection.
- Trust policy is independent of application.
- Can move to another “backend” effortlessly

Thanks to David Wagner

CONFIDENTIAL COMPUTING CONSORTIUM

Cross-platform ML PoC

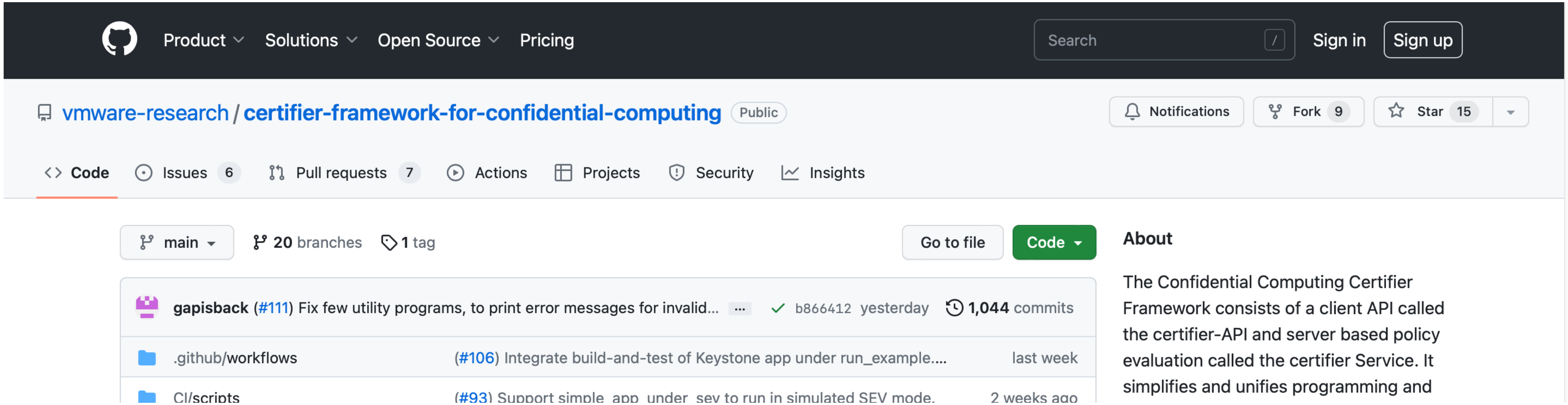


CONFIDENTIAL COMPUTING CONSORTIUM

Can we join the CCC party?

Open-Source project

- github.com/vmware-research/certifier-framework-for-confidential-computing
- Apache license
- Contributions and new contributors are welcome
- Make Confidential Computing an open Universal Platform

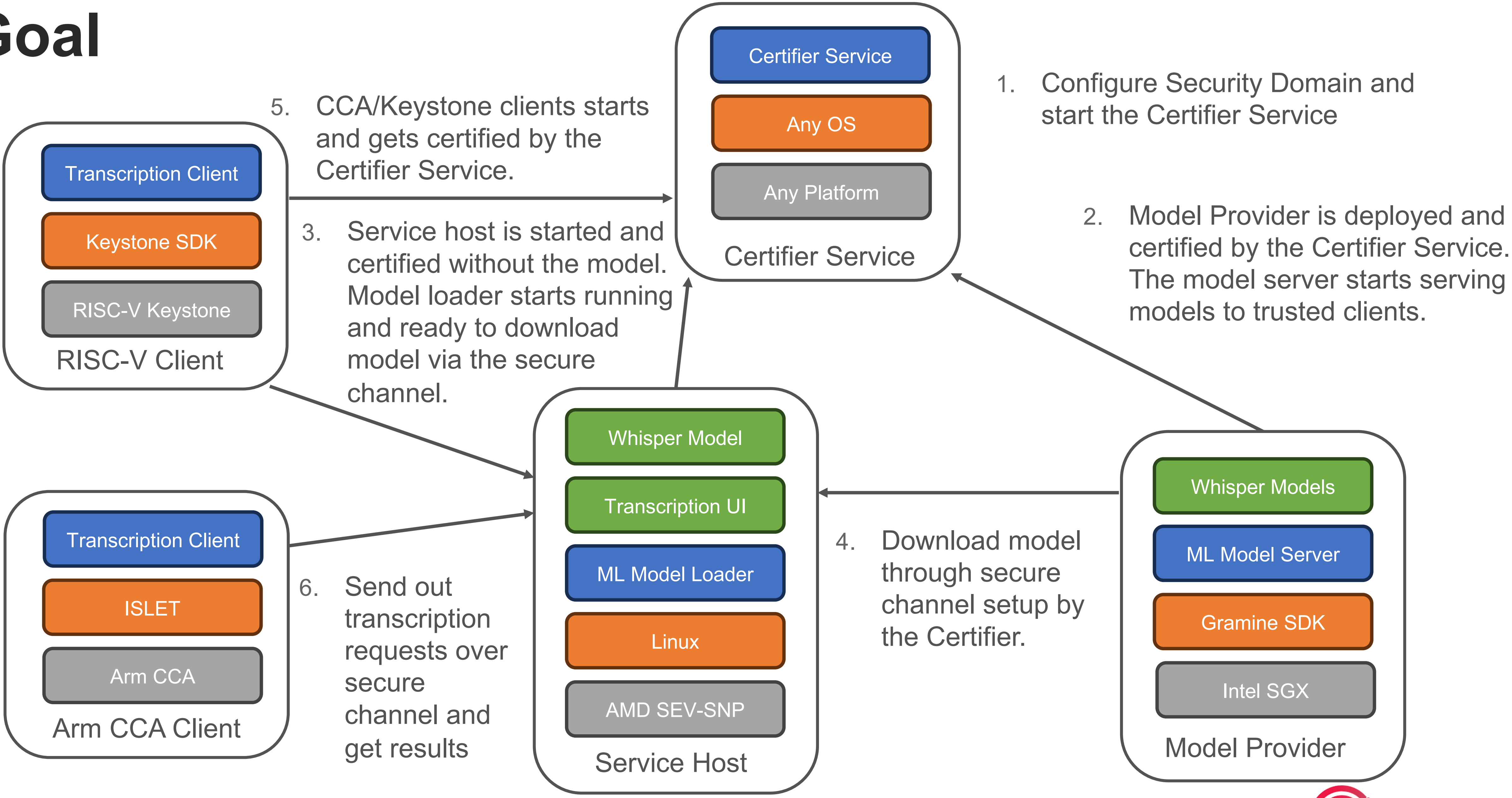


Thank you!

John Manferdelli <jmanferdelli@vmware.com>

CONFIDENTIAL COMPUTING CONSORTIUM

Goal



CONFIDENTIAL COMPUTING CONSORTIUM

Overcoming Barriers to Confidential Computing as a Universal Platform

Abstract: Confidential Computing (CC) provides simple, principled confidentiality and integrity for workloads wherever they run. Within multi-cloud infrastructures, it opens the door for a universal distributed computing solution that addresses verifiable program isolation, programs as authenticated security principals, secure key management, trust management, and the ability to prove these security properties cryptographically "over the wire" to relying parties using attestation. Yet the adoption of confidential computing has been slowed by the difficulty of writing CC-enabled programs quickly and securely, and across hardware technologies. Manferdelli will describe issues and requirements for a universal programming platform and introduce the open source "Certifier Framework for Confidential Computing" that provides a step towards overcoming development barriers.

CONFIDENTIAL COMPUTING CONSORTIUM

The Promise of Confidential Computing

Security enablement anywhere (cloud or not) →	Standard platform components (key store, storage, time, IAM) Secure shared data access (Regulators: GDPR, Health, Finance) Safe program execution ("A safe place to stand in the cloud") Zero Trust
Secure privacy preserving service enablement (Data Economy) →	Secure collaborative machine learning Secure Motion planning as a service Secure Auctions
Secure infrastructure management →	Secure Kubernetes container management Secure Document sharing
Platforms for sensitive edge services →	Edge sensor collection Caching services and the "extended internet"

Barriers to Confidential Computing Adoption

Use or delete

"In the future, all programs will be Confidential Computing Programs" -- Intel

<u>Software</u>

Writing or converting programs to CC is difficult!

- Different code for different programs
- Complicated support code raises a security problem and you can't start right away
- No "copy and paste" to get started with a secure program
- Cooperating programs require flexible management infrastructure to support many program providers and different security requirements.
  - Trust policy often embedded in program (Bad!)
  - Policy difficult to write, understand or audit

What is the "Linux" for Confidential Computing?

<u>Hardware</u>

- TEE availability
- Programs are not "portable" across platform technologies

CC Provides a Foundation for Security

Four capabilities of a Confidential Computing:

- **Isolation.** Program address space and computation.
- **Measurement.** Use cryptographic hash to create an unforgeable program identity.
- **Secrets.** Isolated storage and exclusive program access. (aka, "sealed storage").
- **Attestation.** Enable remote verification of program integrity and secure communication with other such programs.

CC provides principled security wherever your programs run and wherever your data resides *even if you don't operate the computers the programs run on.*

Isolation and measurement

- Program address space isolated
- Program hashed to give non-forgeable identity

Secrets

- Seal: protect a secret for this measurement
- Unseal: restore a secret for this measurement

Attestation

- Statement signed by a trusted party (HW) that specifies
  - Program identity (measurement) program
  - Hardware protection (isolation, integrity, confidentiality) guarantees
  - Statement attributable to isolated entity

CONFIDENTIAL COMPUTING CONSORTIUM