Confidential Computing

Fundamentals, Platforms, and Research

IBM Research

Who we are!

Julian Stephen
IBM Research
julian.stephen@ibm.com

Ray Valdez
IBM Research
rvaldez@us.ibm.com

Zhongshu Gu
IBM Research
zgu@us.ibm.com

Agenda

Session I: Fundamentals

- Why do we need CC?
- Security Principles
- Application Scenarios
- Research Challenges

Session II: Platforms

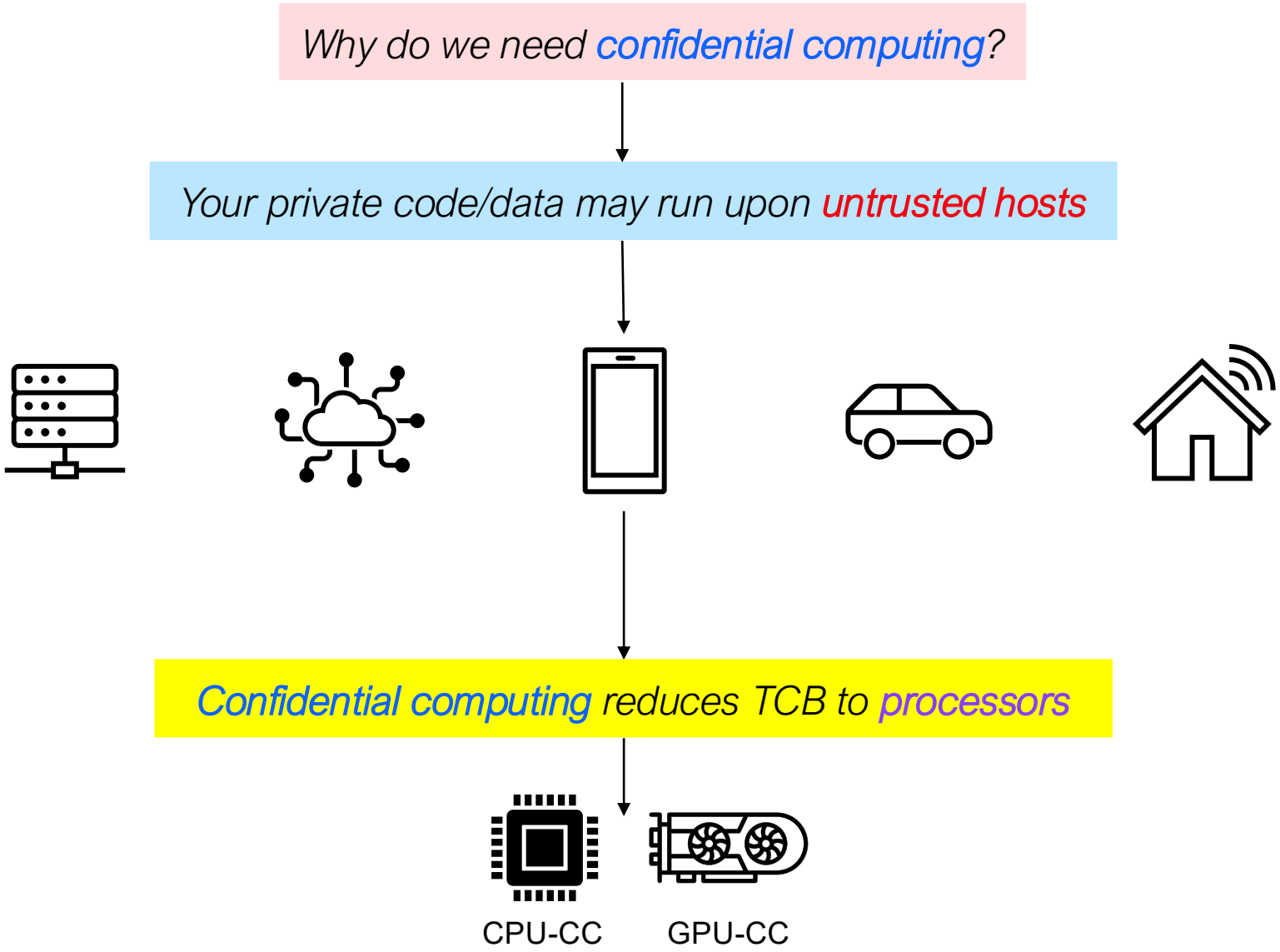
- How does CC work?
- CPU-CC (TDX)
- GPU-CC (Nvidia)

Session III: Research

- Federated Learning (DeTA)
- Security Pitfalls (SplitAPI)
- Performance Implications (GPU Traveling)

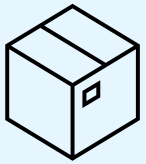
Session I: Fundamentals

Session I: Fundamentals



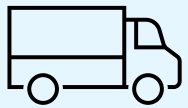
Data at work!

Data at Rest



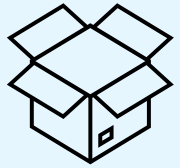
- **Attacks:** unauthorized access or physical theft of storage
- **Mitigations:** disk encryption, e.g., LUKS, BitLocker

Data in Transit



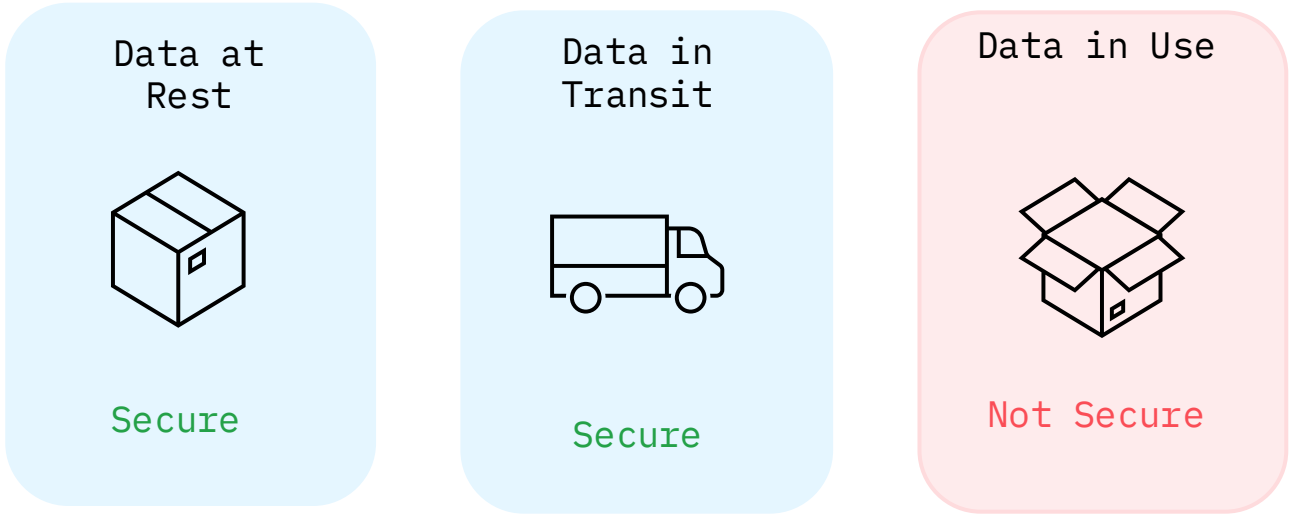
- **Attacks:** eavesdropping, modifying, or replaying network traffic
- **Mitigations:** secure communication, e.g., TLS

Data in Use



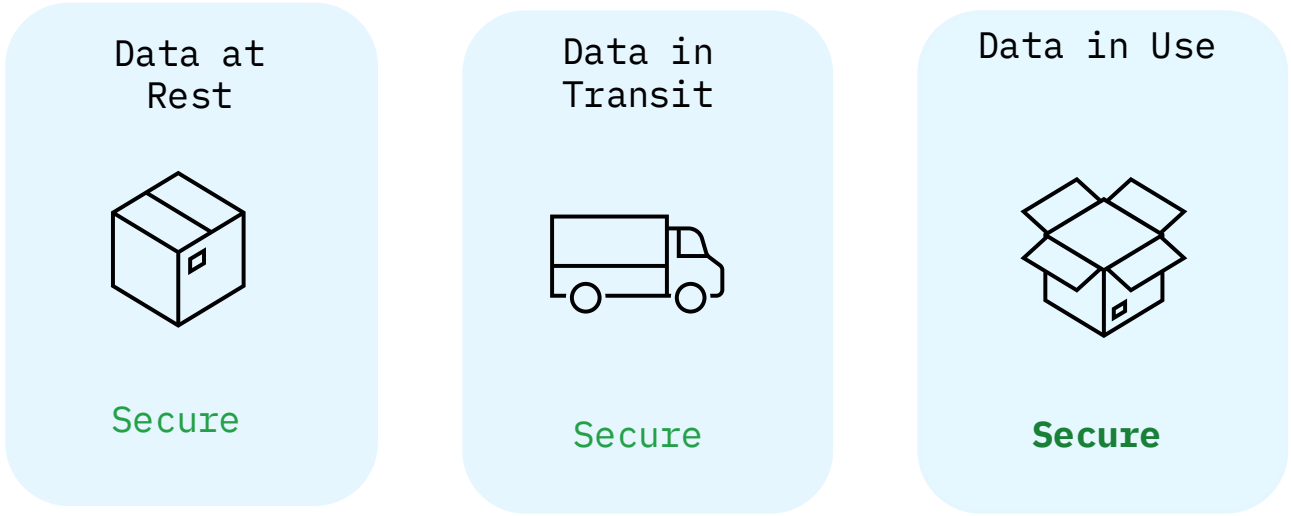
- **Attacks:** memory snooping, execution tampering, cold boot
- **Mitigations:** confidential computing

Why do we need confidential computing?



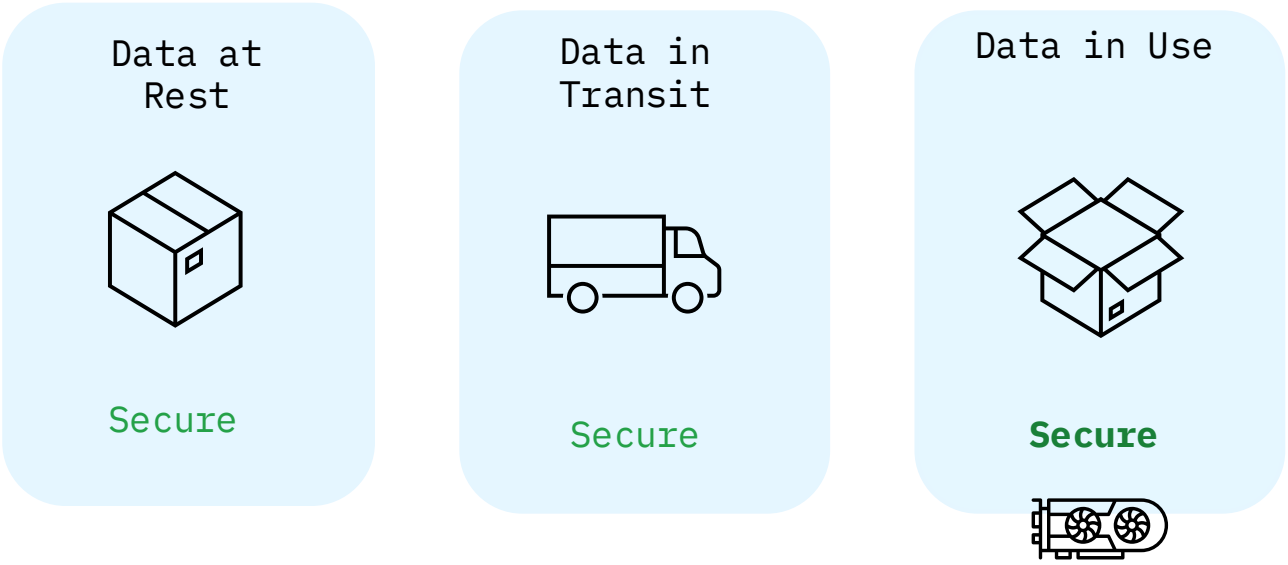
- Little/no protections against weaknesses in
  - Hypervisor
  - Host OS
  - Infrastructure owner
  - Physical access
- Cloud users need to trust
  - Host OS
  - Cloud provider
  - Hypervisor
  - Hardware vendors

Confidential computing



- Protect data in use by performing computation in a hardware-based, attested TEE
- Supported by:
  - AMD SEV, Intel TDX, IBM Z Secure Execution, ARM CCA, …
- No need to trust
  - Host OS
  - Cloud provider/admin
  - Hypervisor
- Need to trust
  - Hardware vendor

GPU confidential computing



- Protect CPU to GPU channel
  - Security Protocol and Data Model (SPDM) session to securely connect the GPU to the GPU driver in a CPU TEE
- Chain of trust established through GPU boot sequence, with secure and measured boot
- Cryptographically signed measurements
- Supported by:
  - NVIDIA Hopper, Blackwell series GPU

Confidential Computing: Classical Workloads

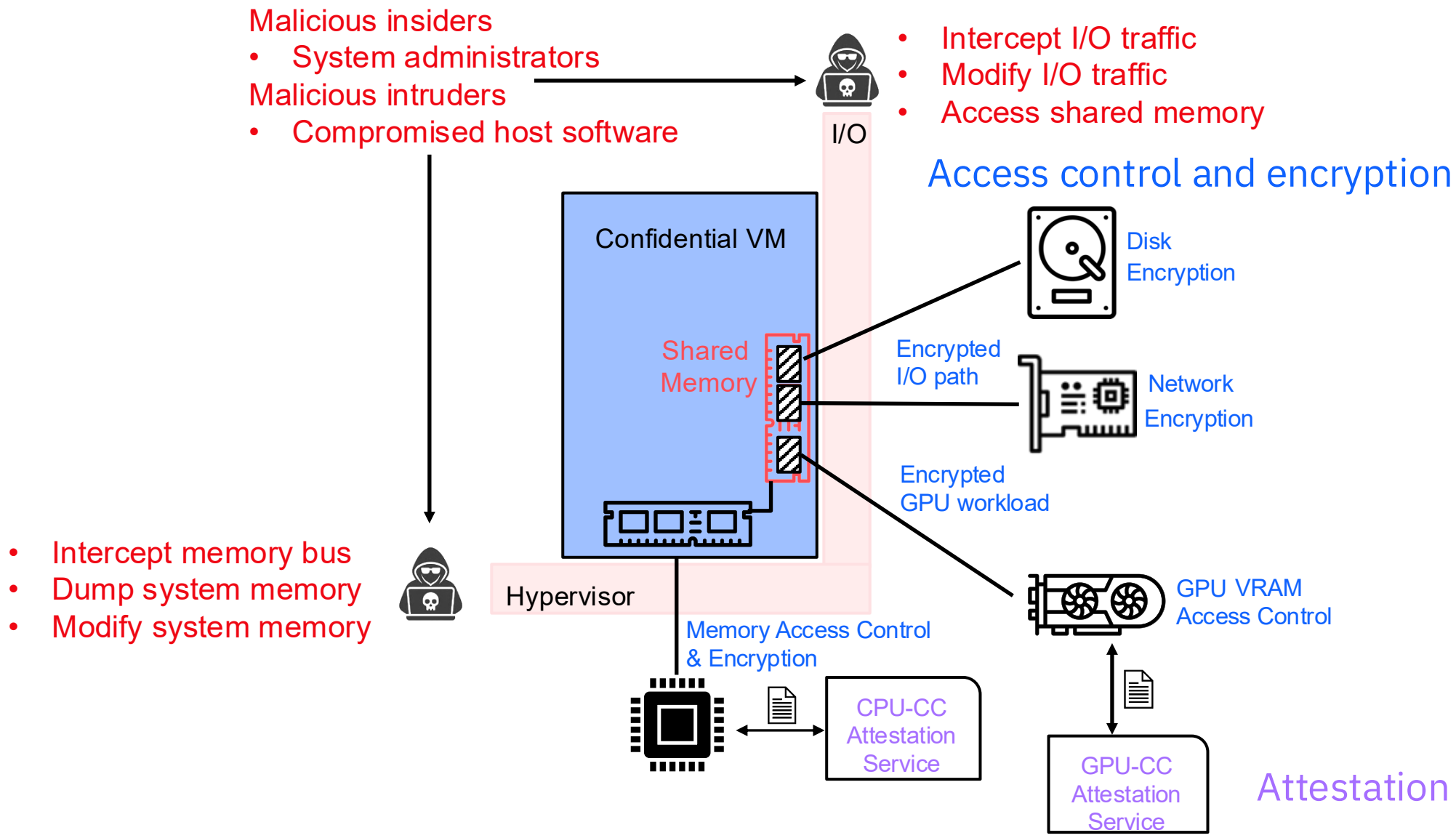
Multi party data sharing

- Cross bank transaction fraud
- Medical diagnosis
- Federated machine learning

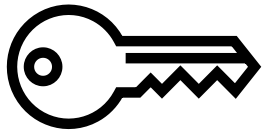
Protecting sensitive data

- Intellectual property (e.g., Electronic Design Automation)
- Operating on PII data

Security Principles

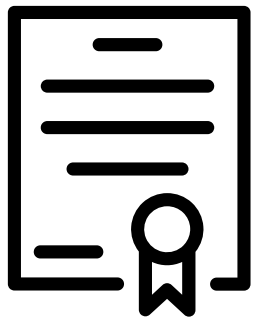


Security Principles



Access control and encryption

- Encrypt data across untrusted IO paths
- System memory encryption and restricted access
- Restricted access to GPU buffers, …



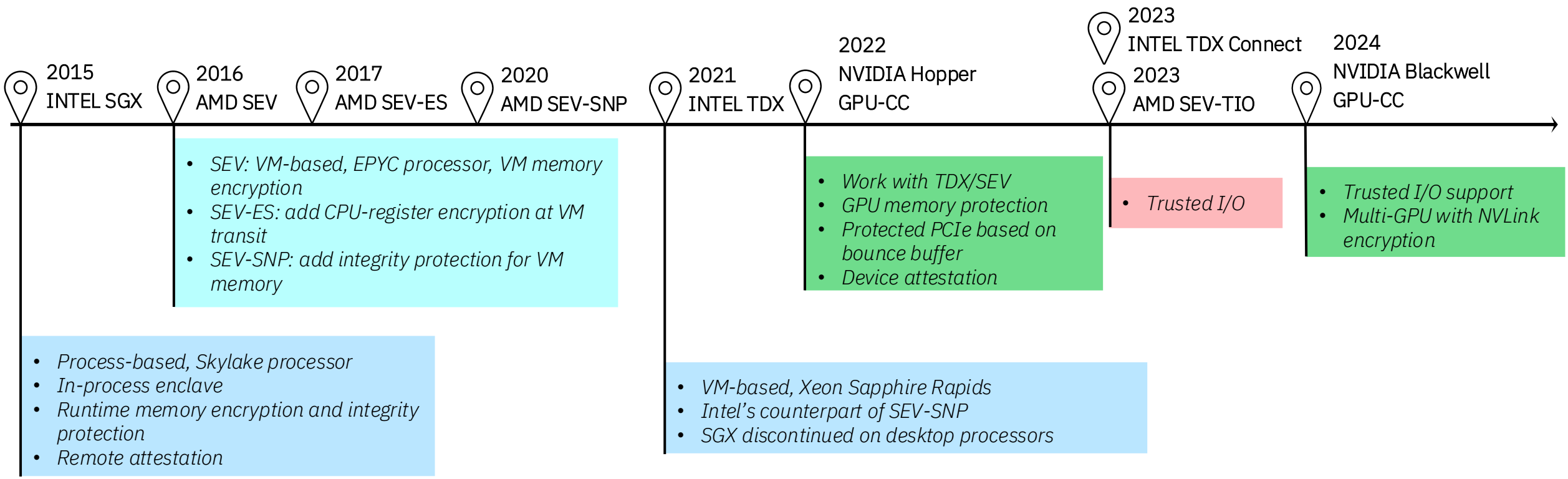
Attestation

- Cryptographically verify system’s hardware and software integrity
- Measure, verify, appraise, certify

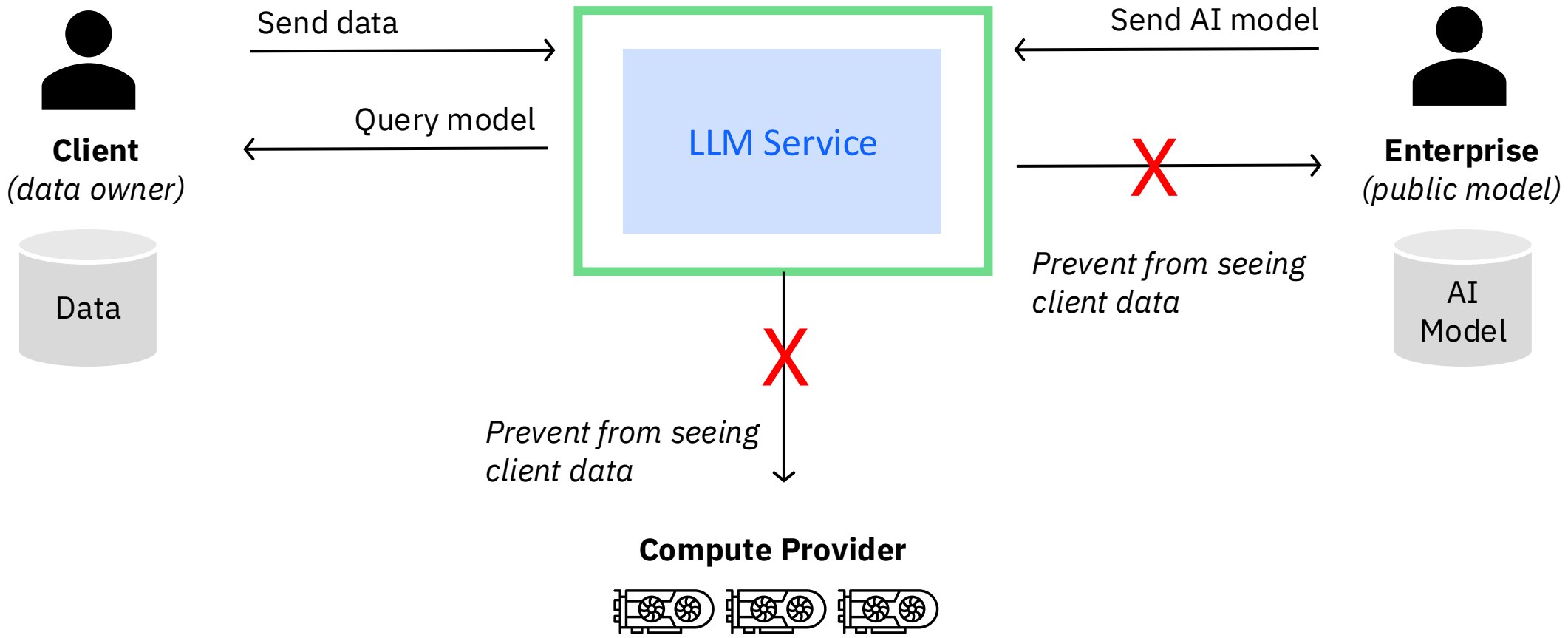
Timeline of CC Technologies

Disclaimer: limited to Intel/AMD CPUs and Nvidia GPUs due to personal research scope
CC on other platforms: IBM PEF/SE, ARM CCA, RISC-V CoVE, etc.

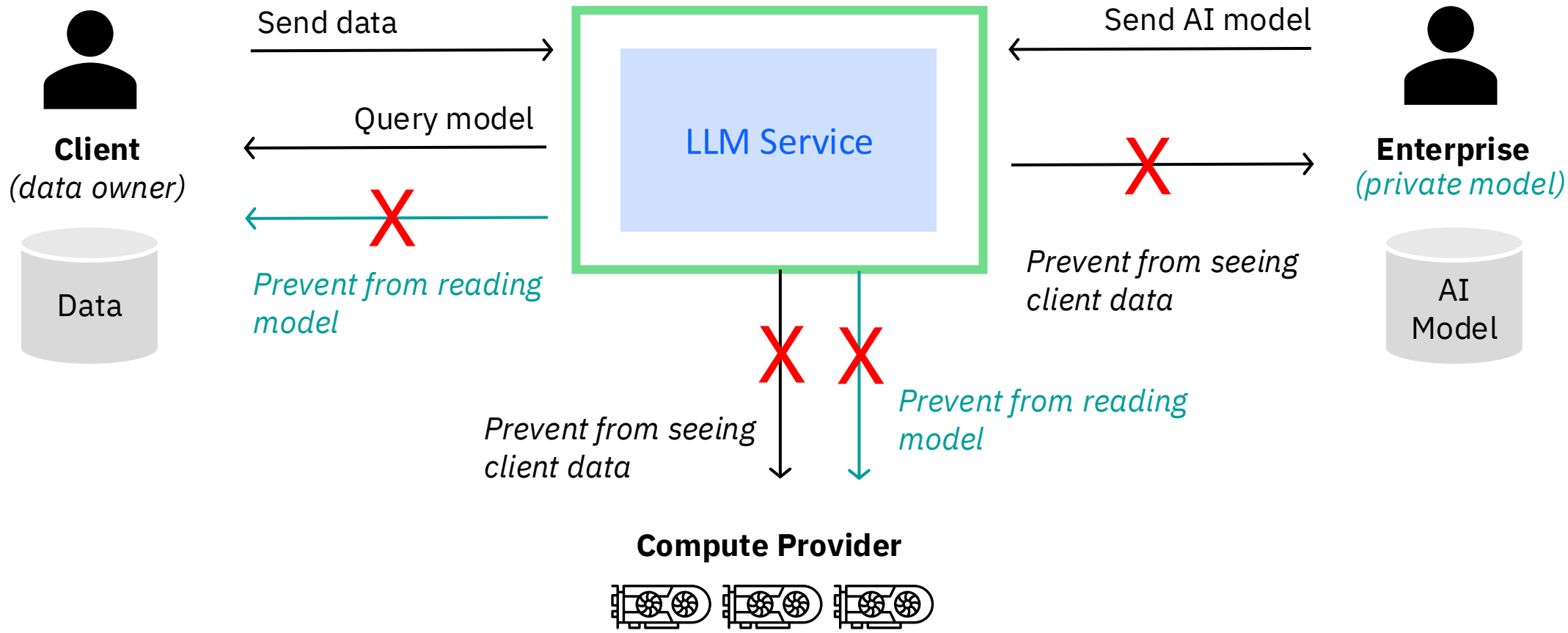
Hardware enablement: early
Software ecosystem: 1~2 years delay



Example: **Confidential AI** and growing **distrust**



Example: **Confidential AI** and growing **distrust**

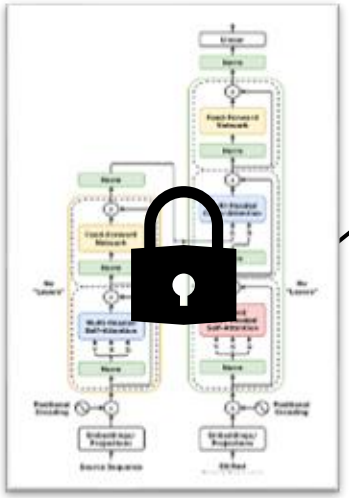


Application I: Confidential AI

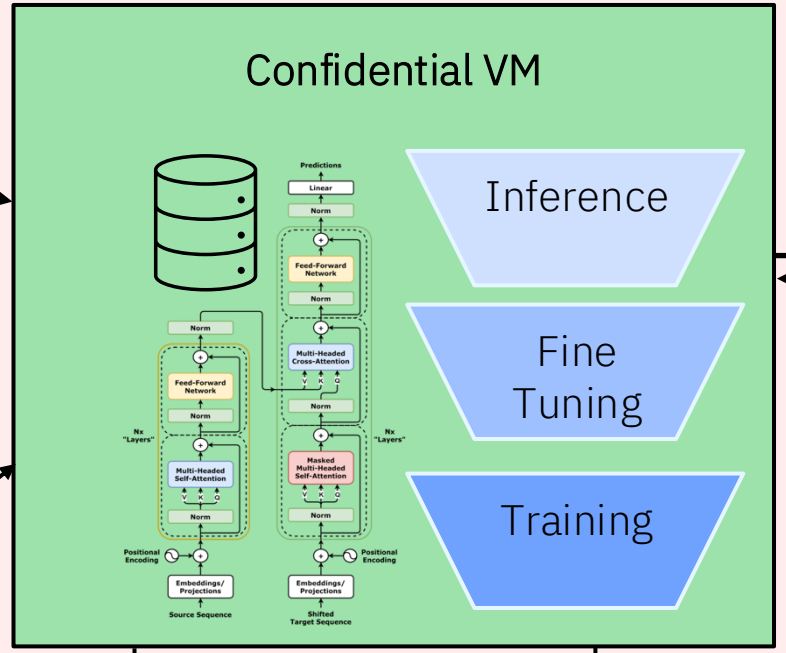
Training/Inference Data: private, sensitive, expensive

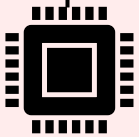
Encrypted Data

AI Model: expensive IP, risk of data reconstruction

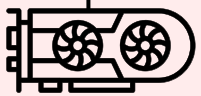


Encrypted Model





CPU-CC



GPU-CC

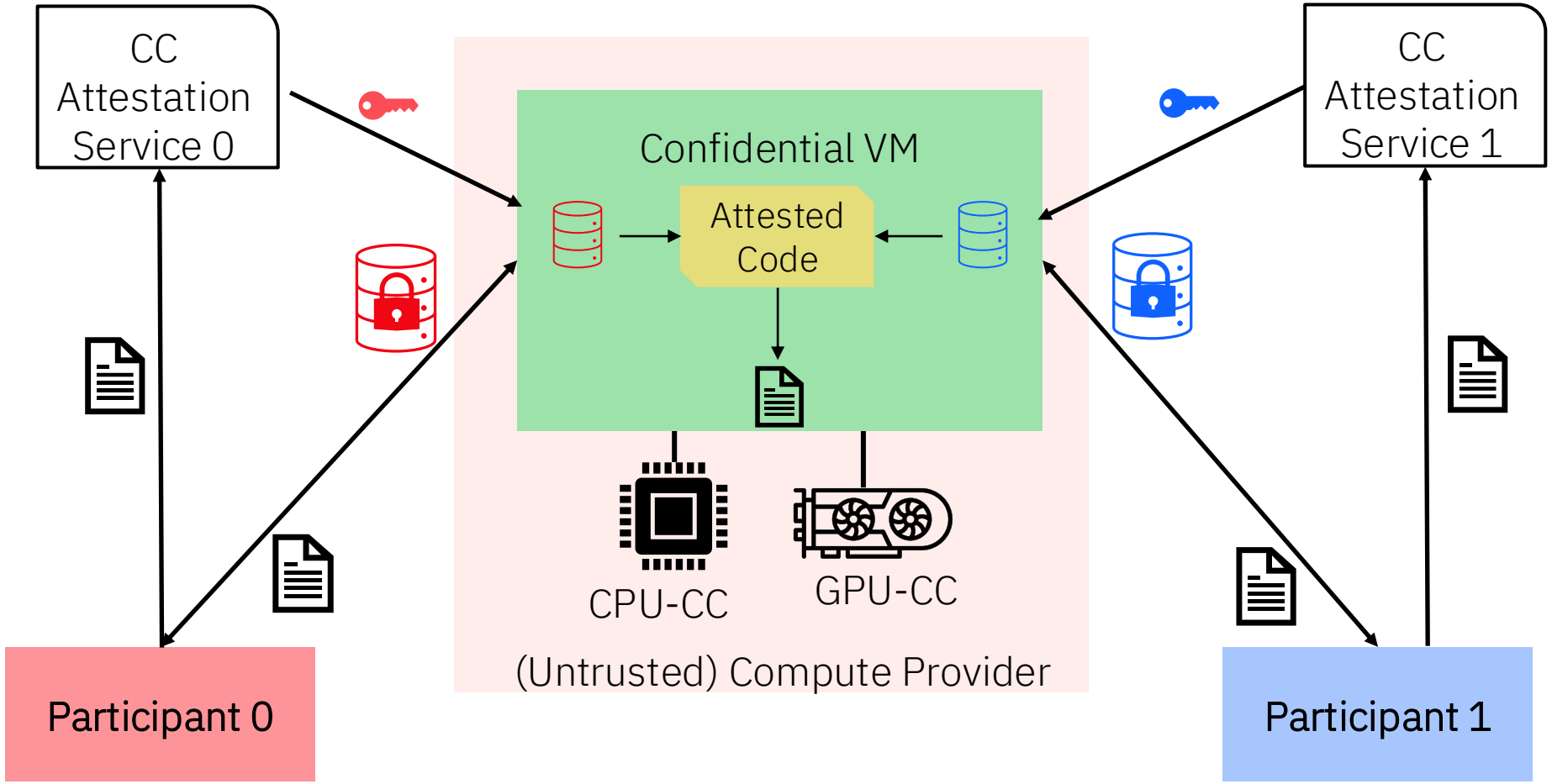
(Untrusted) Compute Provider

CC Attestation Service

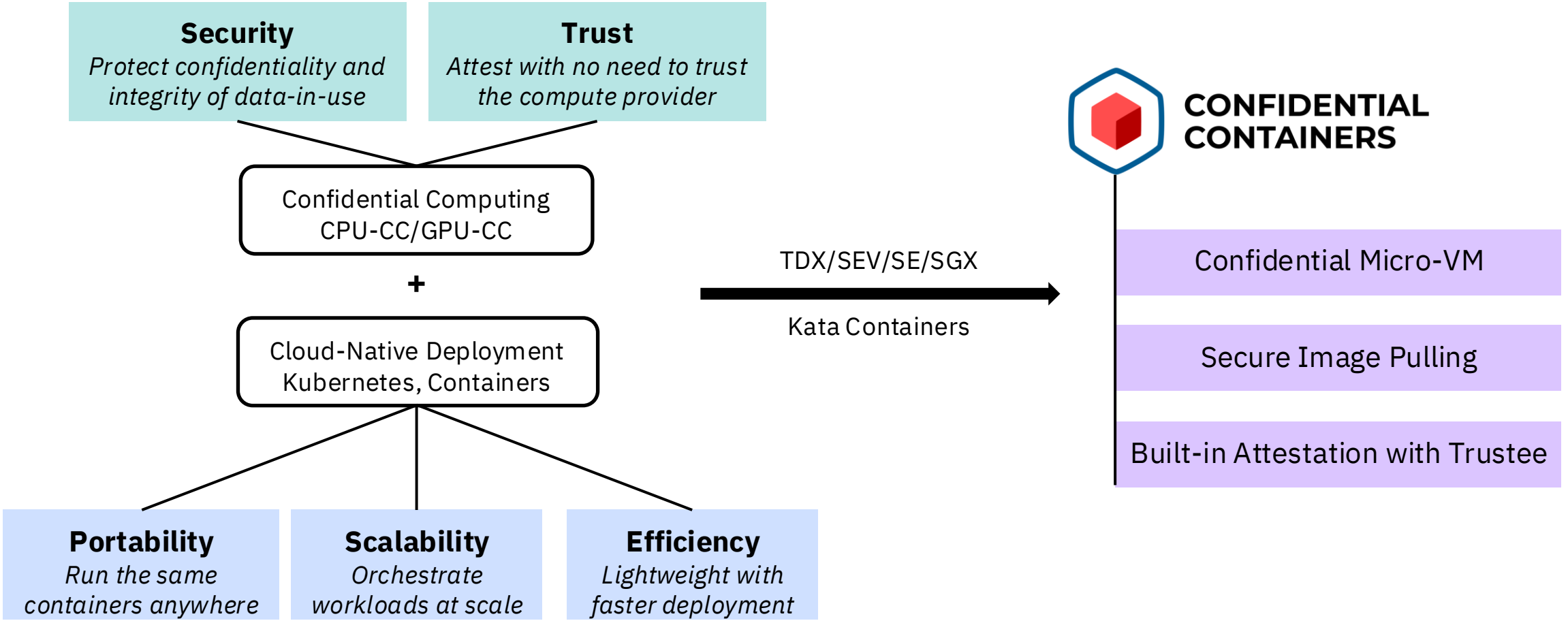
Application II: Collaborative Computing

Collaboration with Mutual Distrust

- No data sharing among participants
- No trust on the compute provider
- Conduct computation together on data

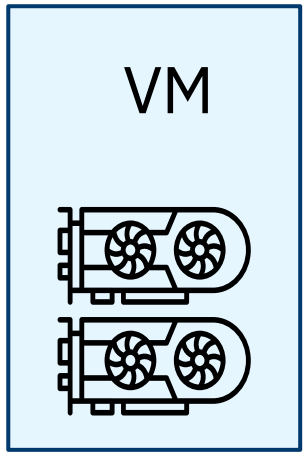


Application III: Confidential Containers



System Considerations: VM configurations

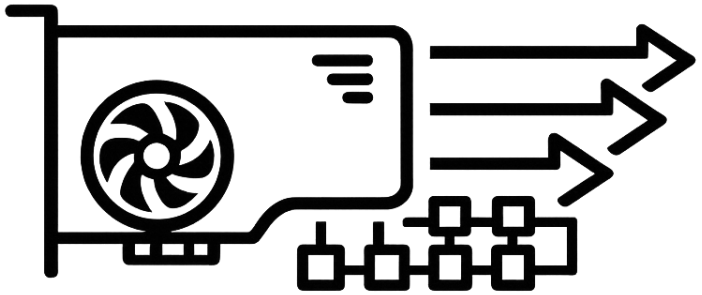
Single GPU

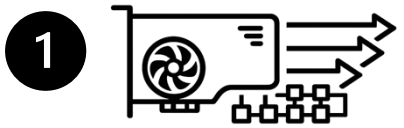


Single-Node
Multi- GPU

Multi-Node
Multi- GPU

System Challenges for **Modern** CC Workloads - 1

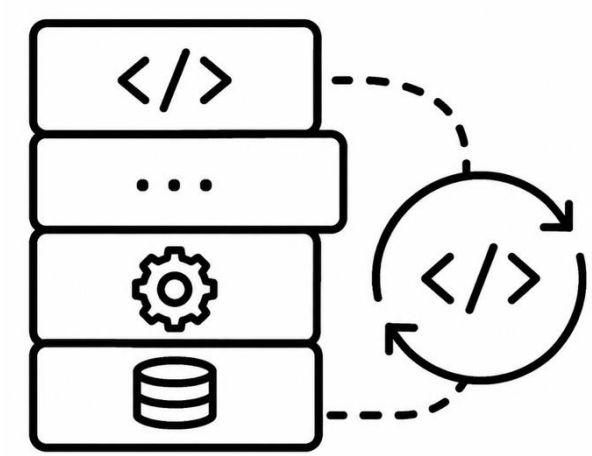


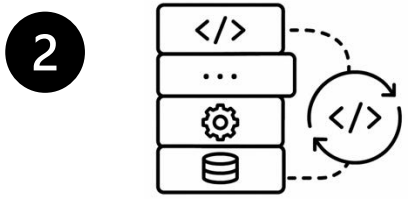


1. Accelerators to support parallel computing

- *SGX: unable to offload computation to accelerators*
- *TDX/SEV + GPU-CC: software-based encryption overheads in bounce buffer*
- *GPU-CC: incomplete support for trusted I/O*

System Challenges for **Modern** CC Workloads - 2

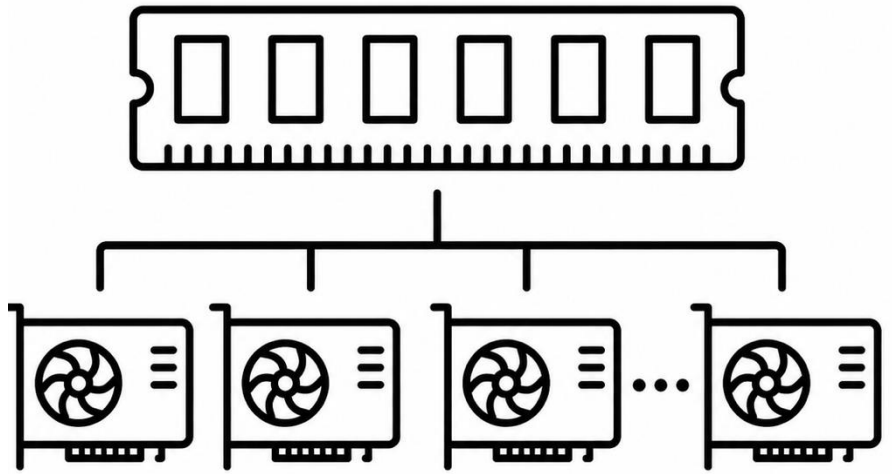




2. Software stack support

- *SGX: refactor code of applications and libraries for the constrained SGX interfaces*
- *TDX/SEV/GPU-CC: significant changes on legacy system code for CC compliance*

System Challenges for Modern CC Workloads - 3



3. Large memory capacity

- *SGX: initially support 128MB protected memory, extending to 1TB (on Xeon after 2021)*
- *GPU-CC: initially with no multi-GPU support*

Security Perspectives - **Workload Owner**

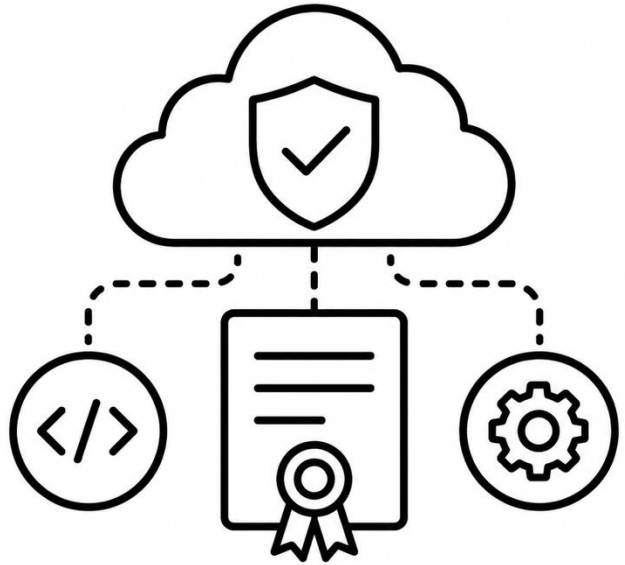
1. Confidentiality and Integrity

- *Can malicious host observe or infer data within CC?*
- *Can malicious host tamper with execution integrity within CC?*
- *Will adding the support for CC make legacy code vulnerable?*
- *Worst case scenario: how to minimize damage if CC is breached?*

Security Perspectives - Compute provider

2. Accountability

- *CC prevents host from introspecting CC workloads*
- *What if the code in CC workloads is malicious?*
- *What if some participants in collaborative computing provide poisoned data?*



Security Perspectives – Security Researcher

3. Transparency

- *The implementations of CC systems are largely proprietary and closed-source*
- *Documents are mostly for end users, not for security researchers*

Key Takeaways (Session I)

Security

- *Confidentiality and integrity of data-in-use*
- *No availability protection*
- *Techniques integration for end-to-end protection*

Mechanisms

- *Memory access control*
- *Memory encryption*
- *Execution integrity protection*
- *Remote attestation*

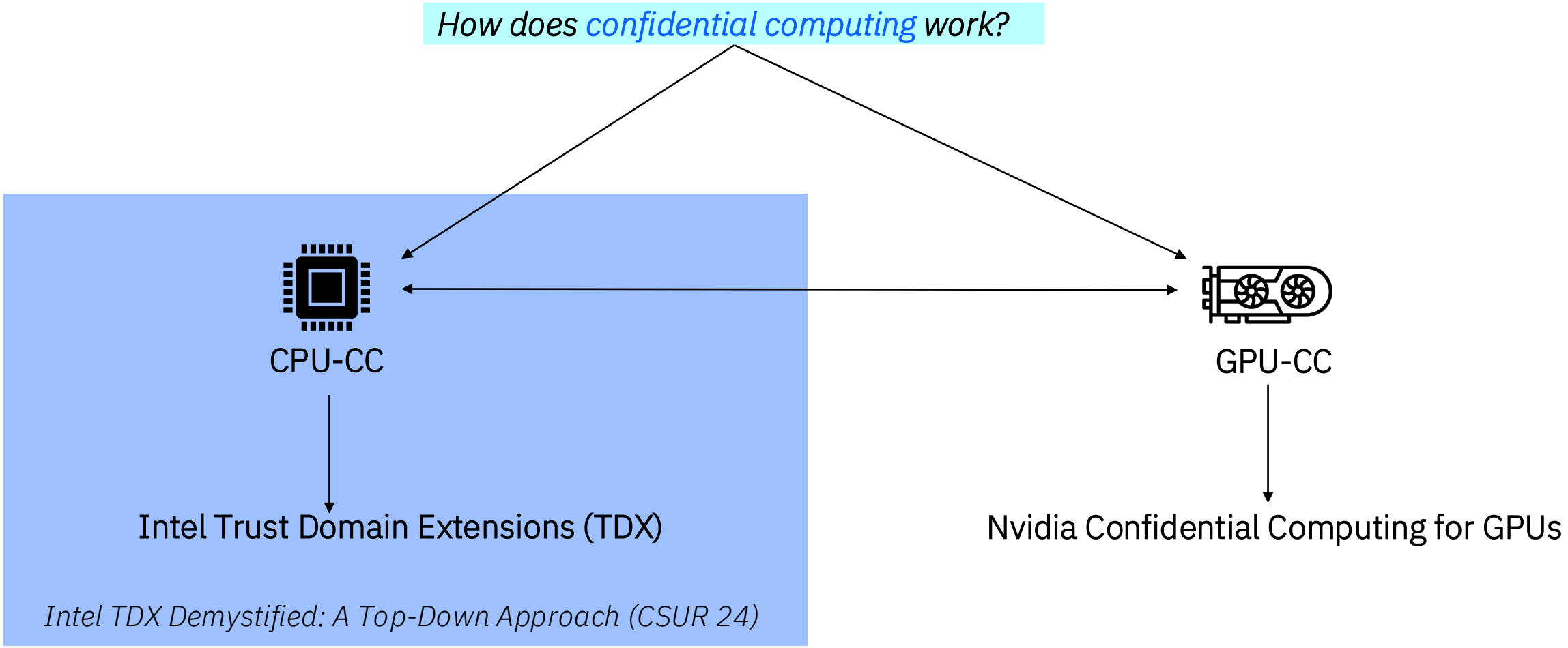
Challenges

- *System constraints*
- *Security problems*
- *Addressed gradually in research and product iterations*

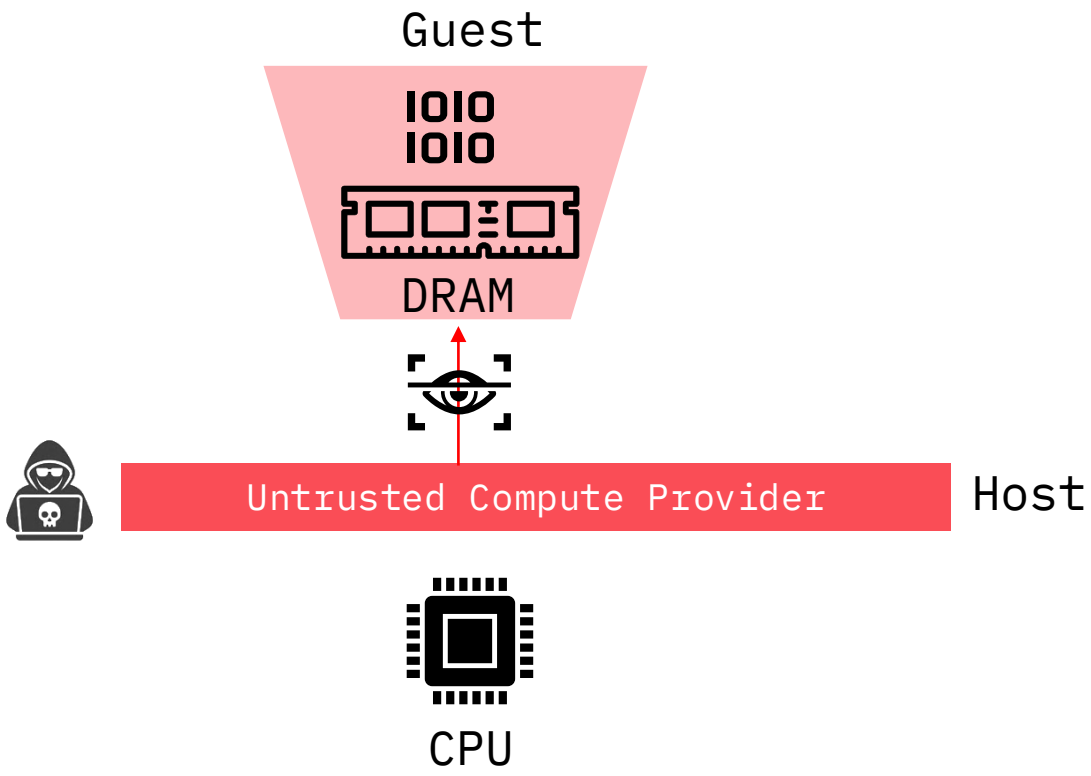
Session I: Fundamentals Q&A

Session II: Platforms

Session II: Platforms



Why We Need CPU-CC?



Scenario

- Protect sensitive computation on untrusted infrastructure

Threat Landscape

- Untrusted compute provider: cloud operator, hypervisor, host OS
- Host software has the privilege to read/write guest memory and vCPU state directly

CPU-CC

- Protect data-in-use: data in guest memory and vCPU state

Process-based CPU-CC → VM-based CPU-CC

Process-based CPU-CC (e.g., Intel SGX)		VM-based CPU-CC (e.g., Intel TDX)
	• Introduce new execution modes or privilege levels • Offload privileged operations into attested firmware/software • Secure or measured launch of trusted components • Protect memory confidentiality and integrity • Enforce cryptographic isolation	
• Isolate specific regions of a process • Code/data within the enclave is trusted • Requires code rewriting for legacy apps • Historically limited memory (128MB) • Overhead of enclave exits for system calls • Use case: highly sensitive code snippet • Deprecated on desktop-grade processors		• Isolate an entire VM • Code/data within the VM is trusted • Lift-and-shift: runs unmodified apps • Large secure memory • System calls within the VM boundary • Use case: general cloud workloads • Available on server-grade processors

Intel TDX Overview

- Intel Trust Domain Extensions (TDX): VM-based CPU-CC
  - Trust Domain (TD) = Confidential Virtual Machine (CVM)
  - Supported since 4th Gen Intel Xeon Processors
- Security Features of TDX
  - Exclude hypervisor from the TCB
  - Enforce Cryptographic isolation of security domains
  - Provide confidentiality and integrity for TD’s memory and vCPU state
  - Remote attestation and verifiability

Building Blocks of TDX

Intel VT → Isolation

Hardware-assisted Virtualization

- Hardware virtualization foundation
- Trust Domains built on VM technology
- TDX Module manages and isolates TDs

Intel MKTME → Memory Protection

Runtime Memory Encryption

- Hardware memory encryption
- Per-TD encryption keys
- Host cannot read TD private memory

Intel SGX → Attestation

Secure Enclaves

- Reuses SGX attestation infrastructure
- Quoting Enclave generates signed evidence
- Remote verification of TD state

Intel TXT → Trusted Launch

Trusted Execution Technology

- Measures and verifies software stack
- Trusted launch of TDX components
- Hardware-rooted chain of trust

New Features of TDX

Secure-Arbitration Mode (SEAM)

- *SEAM VMX root* (SEAM Modules)
- *SEAM VMX non-root* (TD)

SEAM Modules (in SEAM Range)

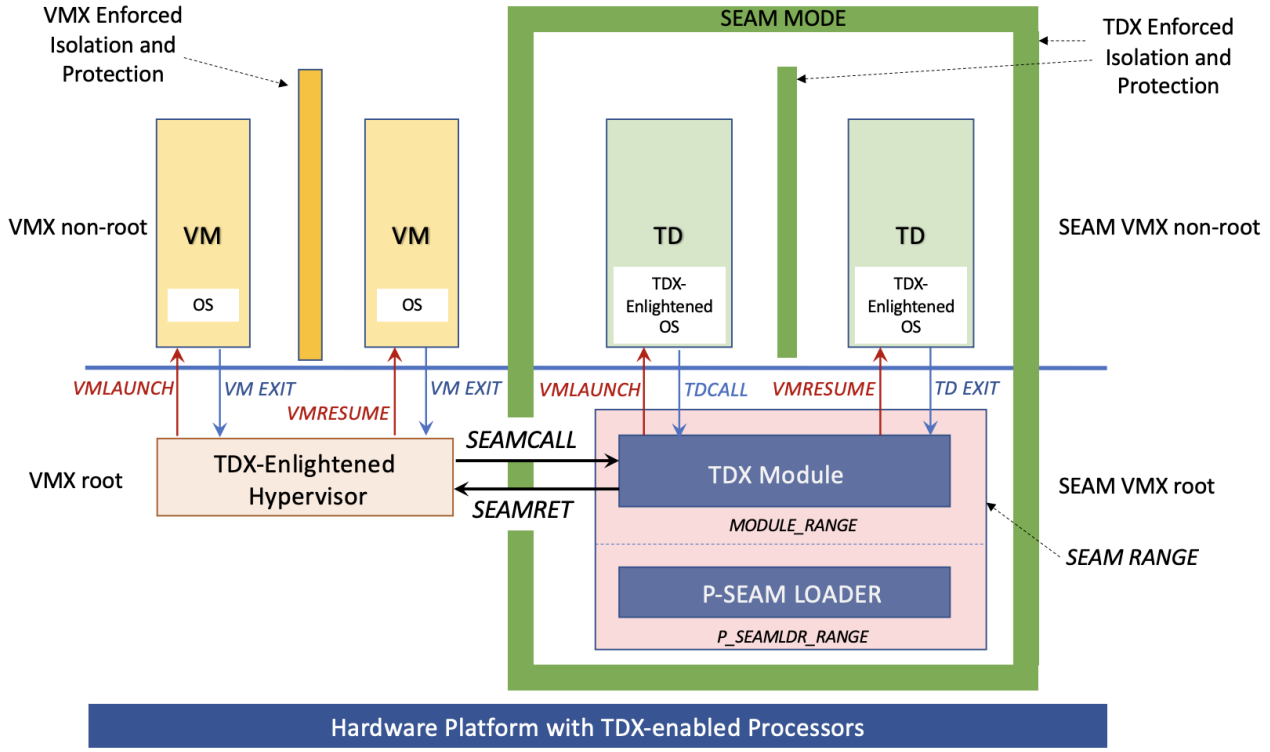
1. P-SEAM Loader
   - Install/update the *TDX Module*
2. TDX Module
   - A “trusted” management layer for TDs
   - Enforce protection for TD’s memory and CPU state

Transition Interfaces

- *SEAMCALL* and *SEAMRET*
- *TDCALL* and *TD Exit*

Enlightened Hypervisor and Guest OS

- Adapt to the new transition interfaces



Secure Loading of TDX Module

CPU

↓

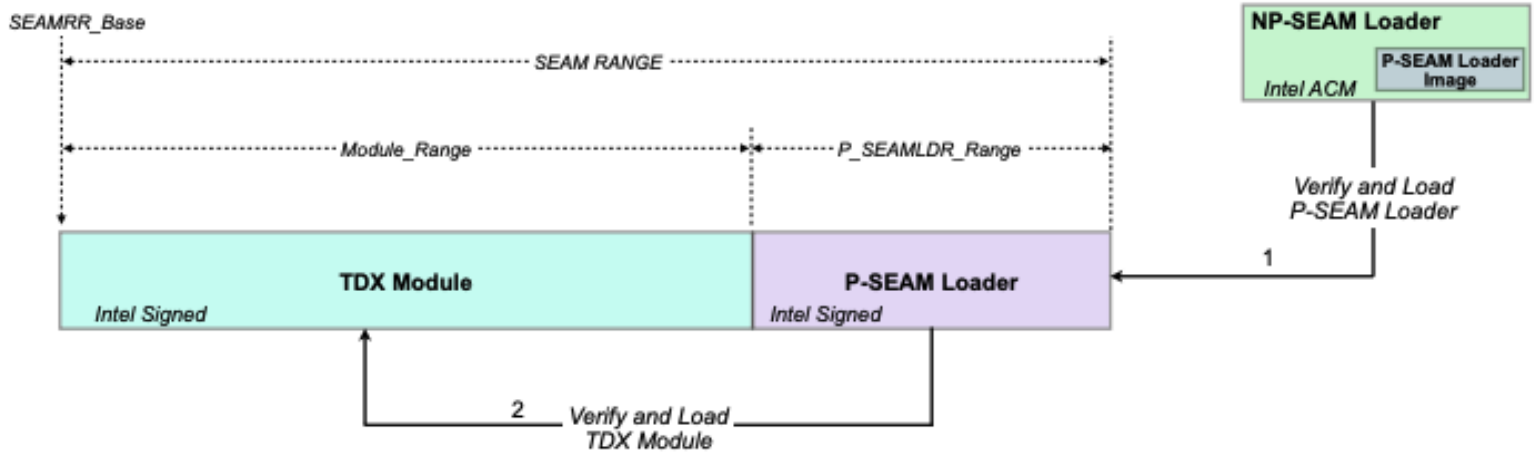
NP-SEAM Loader (ACM)

↓

P-SEAM Loader

↓

TDX Module



Memory Protection of TDX

Memory Conversion: Normal memory → Secure Memory

- Attach private *HKIDs* to physical addresses
- *TD owner bit* per cache line
- Controlled by the *TDX Module*
- Untrusted hypervisor cannot access or modify secure memory

Secure Memory: Confidentiality and Integrity

- Private memory of TDs
- Metadata of TDs
  - TD control structures
  - vCPU state
  - *Secure EPT*: Address translation for private memory

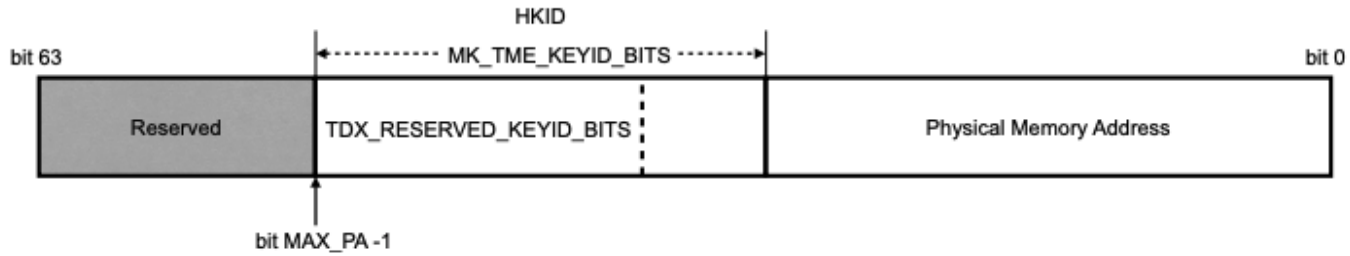
Normal Memory within TD

- Shared memory: accessible by the hypervisor and I/O devices
- Move data from secure memory to shared memory
  - Users should enforce software-based encryption: TLS, LUKS

Memory Confidentiality

HKID Partitioning

- *HKID* partitioning is determined via UEFI/BIOS
- *Private HKIDs* for TDX and *shared HKIDs* for legacy MKTME use



Use MKTME in TDX Module

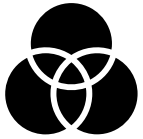
- Each TD associates with one *private HKID*
- Only the *TDX Module* (in *SEAM VMX root mode*) can program *private HKIDs* via MKTME
- The *TDX Module* attaches corresponding *HKIDs* to physical addresses of TDs
- MKTME selects keys to encrypt memory pages based on HKIDs

	HKID	Key
Shared HKIDs	0	Legacy TME key, shared
	1	Legacy MKTME key #1
	2	Legacy MKTME key #2
	…	…
	NUM_MKID_KEYS	Last legacy MKTME key
Private HKIDs	NUM_MKID_KEYS + 1	Private key of a specific TD
	NUM_MKID_KEYS + 2	Private key of a specific TD
	NUM_MKID_KEYS + 3	Private key of a specific TD
	…	…
	NUM_MKID_KEYS + NUM_TDX_PRIV_KIDS	Private key of a specific TD

Memory Integrity

Logical Integrity (simple, fast)

- TD owner bit per cache line
- Legitimate write: set TD owner bit to 1
- Illegal write: clear TD owner bit to 0
- Next read: TD owner bit 0 indicates poisoned
- Effective for software-level attack: cannot defend against rowhammer attack



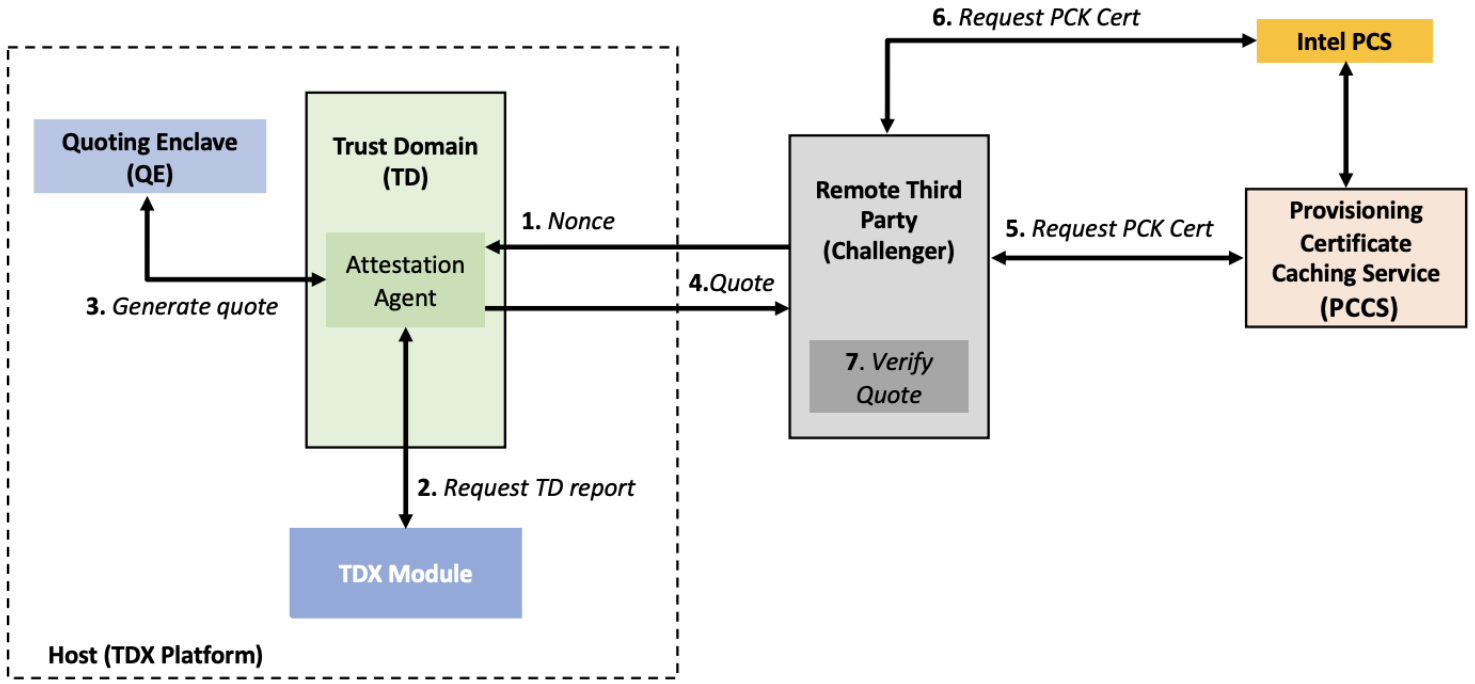
Cryptographic Integrity (slow, more secure)

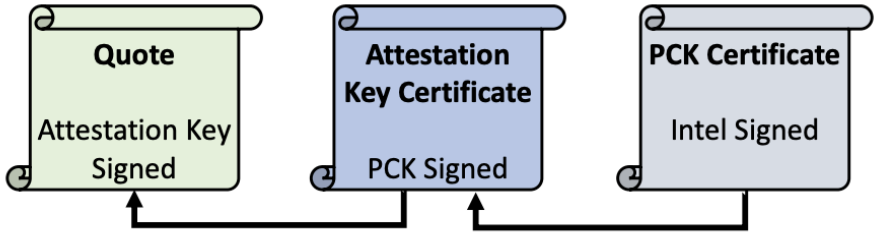
- 28-bit MAC per cache line based upon
  - Ciphertext
  - Tweak value for AES-XTS
  - TD owner bit
  - 128-bit MAC key
- Legitimate write: store MAC in ECC
- Illegal write: corrupt the memory
- Next read: mismatched MAC indicates poisoned
- Effective for software-level and some physical attacks: cannot defend against replay attack

Remote Attestation

Repurpose SGX Attestation

- The *TDX Module* retrieves TD report
- TD report: measurements of TD and TDX platform
- *Quoting Enclave* verifies TD report and signs it with the attestation key
- Quote = signed report + certificate chain
- The challenger verifies certificate chain and measurements in the Quote





Features in Progress

Live Migration

- Mutual attestation of source and destination TDX platforms
- Secure channel for transferring TD assets
- TDX Module 1.5

TD Partitioning

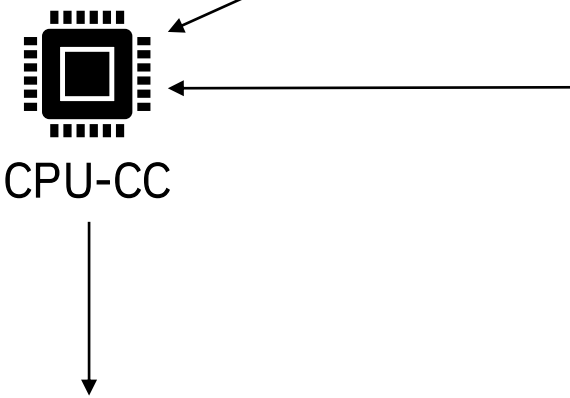
- Nested virtualization for TD
- TDX Module 1.5

TDX Connect

- Move trusted devices into trust boundary
- Device attestation
- Protected I/O data path
- TDX Module 2.0

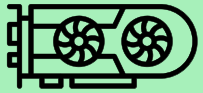
Session II: Platforms

How does confidential computing work?



CPU-CC

Intel Trust Domain Extension (TDX)

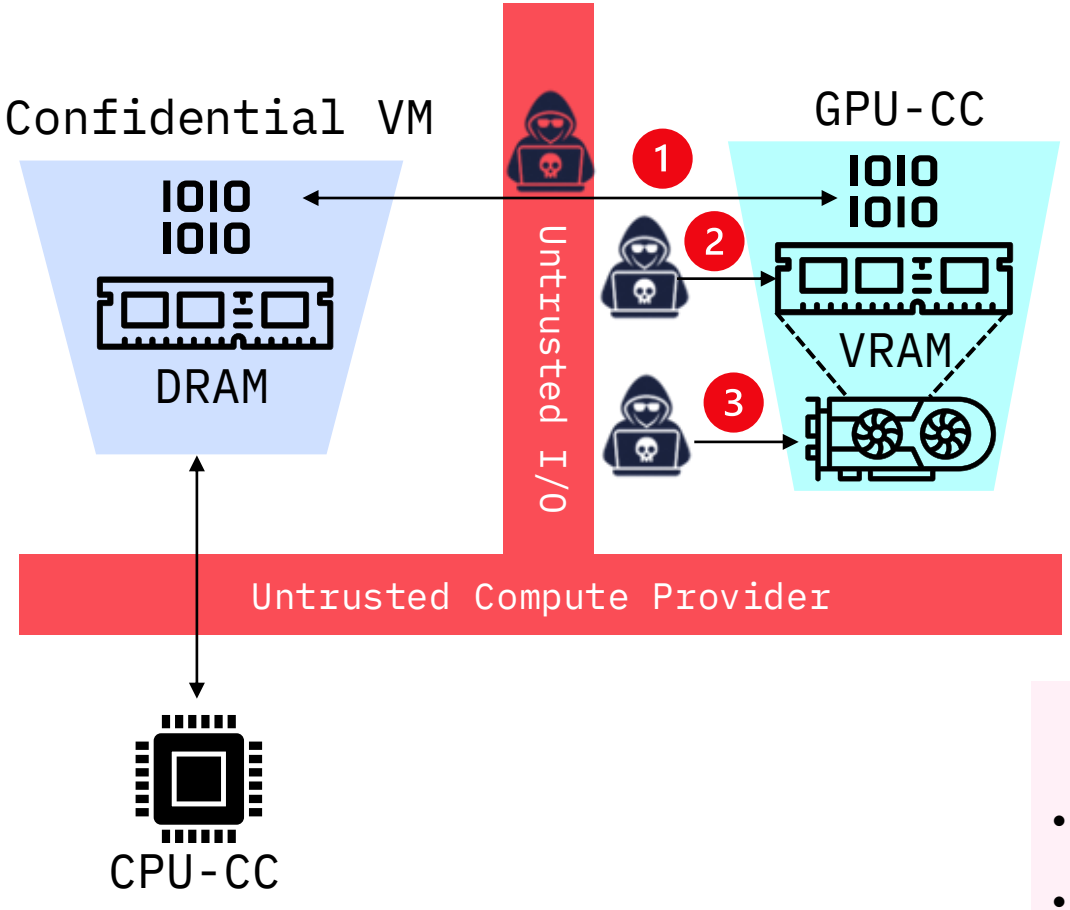


GPU-CC

Nvidia Confidential Computing for GPUs

Blueprint, Bootstrap, and Bridge:
A Security Look at NVIDIA GPU Confidential Computing (MLSys 26)

Background



Confidential Computing

- Protect sensitive workloads on untrusted 3rd party's machines, e.g., public cloud
- Threat model: attackers control the privileged host system software

CPU-CC

- Protect generic computation
- Protected domain: CVM
- Confidentiality, integrity and authenticity

GPU-CC

- Protect computation offloaded to GPU
- A unified protected domain: CVM + GPU

Attack Targets in CPU-CC

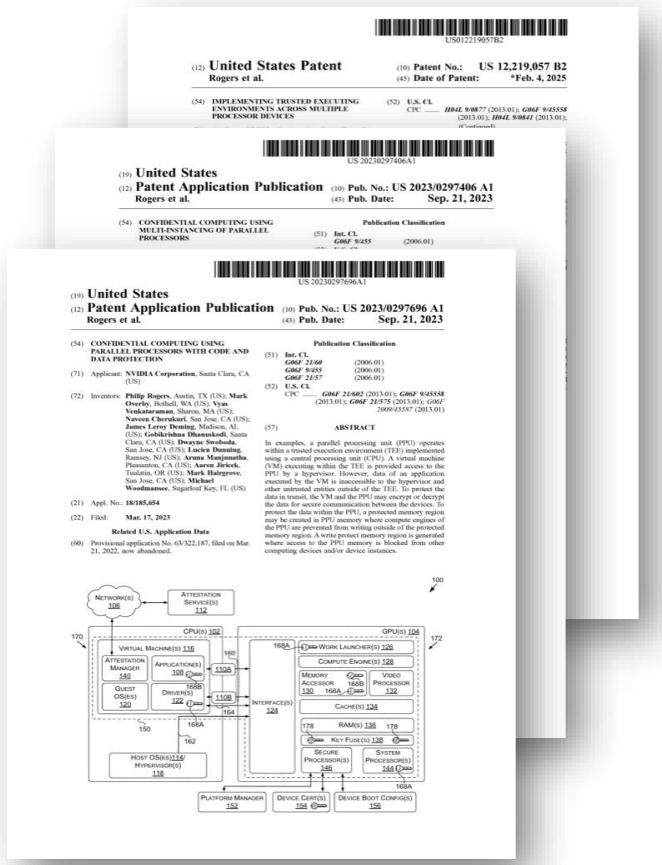
- CVM's memory
- CVM's virtual CPUs
- CVM's I/O

Attack Targets in GPU-CC

- I/O between CVM and GPU
- GPU's VRAM
- GPU's control registers

NVIDIA GPU Confidential Computing

GPU-CC is critical for protecting sensitive AI workloads in the cloud



(12) **United States Patent**
Rogers et al.
(10) Patent No.: US 12,219,057 B2
(45) Date of Patent: *Feb. 4, 2025
(54) IMPLEMENTING TRUSTED EXECUTING ENVIRONMENTS ACROSS MULTIPLE PROCESSOR DEVICES

(19) United States
(12) Patent Application Publication
Rogers et al.
(10) Pub. No.: US 2023/0297406 A1
(43) Pub. Date: Sep. 21, 2023
(54) CONFIDENTIAL COMPUTING USING MULTI-INSTANCING OF PARALLEL PROCESSORS

(19) United States
(12) Patent Application Publication
Rogers et al.
(10) Pub. No.: US 2023/0297696 A1
(43) Pub. Date: Sep. 21, 2023
(54) CONFIDENTIAL COMPUTING USING PARALLEL PROCESSORS WITH CODE AND DATA PROTECTION
(71) Applicant: NVIDIA Corporation, Santa Clara, CA (US)
(72) Inventors: Philip Rogers, Austin, TX (US); Mark Overby, Bothell, WA (US); Vyas Venkataraman, Sharon, MA (US); Naveen Cherukuri, San Jose, CA (US); James Leroy Deming, Madison, AL (US); Gobikrishna Dhanuskodi, Santa Clara, CA (US); Dwayne Swoboda, San Jose, CA (US); Lucien Dunning, Ramsey, NJ (US); Aruna Manjunatha, Pleasanton, CA (US); Aaron Jirbeck, Tualatin, OR (US); Mark Hairgrove, San Jose, CA (US); Michael Woodmansee, Sugarloaf Key, FL (US)
(21) Appl. No.: 18/185,654
(22) Filed: Mar. 17, 2023
Related U.S. Application Data
(60) Provisional application No. 63/322,187, filed on Mar. 21, 2022, now abandoned.
Publication Classification
(51) Int. Cl.
(52) U.S. Cl.
(57) ABSTRACT
In examples, a parallel processing unit (PPU) operates within a trusted execution environment (TEE) implemented using a central processing unit (CPU). A virtual machine (VM) executing within the TEE is provided access to the PPU by a hypervisor. However, data of an application executed by the VM is inaccessible to the hypervisor and other untrusted entities outside of the TEE. To protect the data in transit, the VM and the PPU may encrypt or decrypt the data for secure communication between the devices. To protect the data within the PPU, a protected memory region may be created in PPU memory where compute engines of the PPU are prevented from writing outside of the protected memory region. A write protect memory region is generated where access to the PPU memory is blocked from other computing devices and/or device instances.

practice

DOI:10.1145/3626827

The team at NVIDIA brings confidentiality and integrity to user code and data for accelerated computing.

BY GOBIKRISHNA DHANUSKODI, SUDESHNA GUHA, VIDHYA KRISHNAN, ARUNA MANJUNATHA, ROB NERTNEY, MICHAEL O'CONNOR, AND PHIL ROGERS

Creating the First Confidential GPUs

TODAY'S DATACENTER GPU has a long and storied 3D-graphics heritage. In the 1990s, graphics chips for PCs and consoles had fixed pipelines for geometry, rasterization, and pixels using integer and fixed-point arithmetic. In 1999, NVIDIA invented the modern GPU, which put a set of programmable cores at the heart of the chip, enabling rich 3D-scene generation with great efficiency. It did not take long for developers and researchers to realize: "I could run compute on those parallel cores, and it would be blazing fast." In 2004, Ian Buck created Brook at Stanford, the first compute library for GPUs, and in 2006, NVIDIA created CUDA, which is the gold standard for accelerated computing on GPUs today.

In addition to running 3D graphics and compute, GPUs also run video workloads, including the ability to play back protected content, such as Hollywood movies. To protect such content, NVIDIA GPUs include hardware and firmware to secure the area of GPU memory, which holds the decrypted and decoded output frames. This feature is referred to as video protected region (VPR). When an area of GPU memory is set up as VPR—except for a secured display engine that can read from VPR and write to HDMI or DisplayPort channels—any engine that reads from that region will fault if it attempts to write outside of VPR. When confidential computing (CC) emerged, a few of us at NVIDIA started brainstorming about the question, "Can we leverage VPR, or a similar approach, to do confidential compute?" We realized that NVIDIA's Ampere series of GPUs provided the building blocks for a partial CC mode. New firmware could enable an enclave in GPU memory for protected compute, where:

- Only the SEC2 secure microcontroller can read from the enclave and write outside; and when it writes outside, it will first encrypt the data.
- All other engines would fault if they tried to write outside the enclave.

CC requires both confidentiality and integrity for data and code. *Confidentiality* means data and code cannot be read by an attacker. *Integrity* means an attacker cannot modify the execution and, for example, cause wrong answers to be generated. The leveraged Ampere approach could provide confidentiality for data but not for code, and it could protect integrity for neither code nor data. This approach was called Ampere Protected Memory (APM) to prevent confusion with full CC capabilities. We built a proof of concept (POC) for APM and partnered with Microsoft to

60 COMMUNICATIONS OF THE ACM JANUARY 2024 VOL. 67 NO. 1

Patents: filed in 2021 and 2023

CACM Paper: 2024

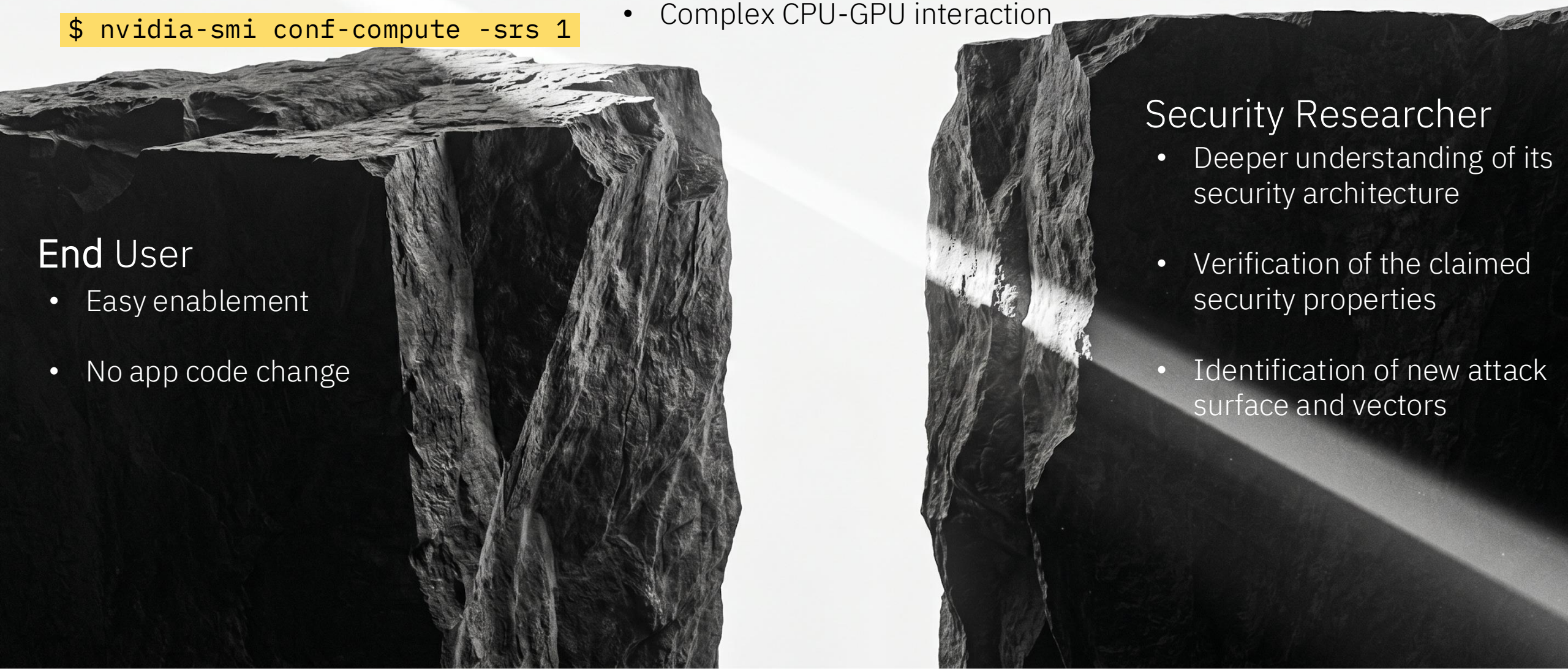
Provide hardware support since Hopper

Motivation

Knowledge Gap
Principle ↔ Implementation

- Proprietary components
- Undocumented features
- Complex CPU-GPU interaction

```
$ nvidia-smi conf-compute -srs 1
```



End User

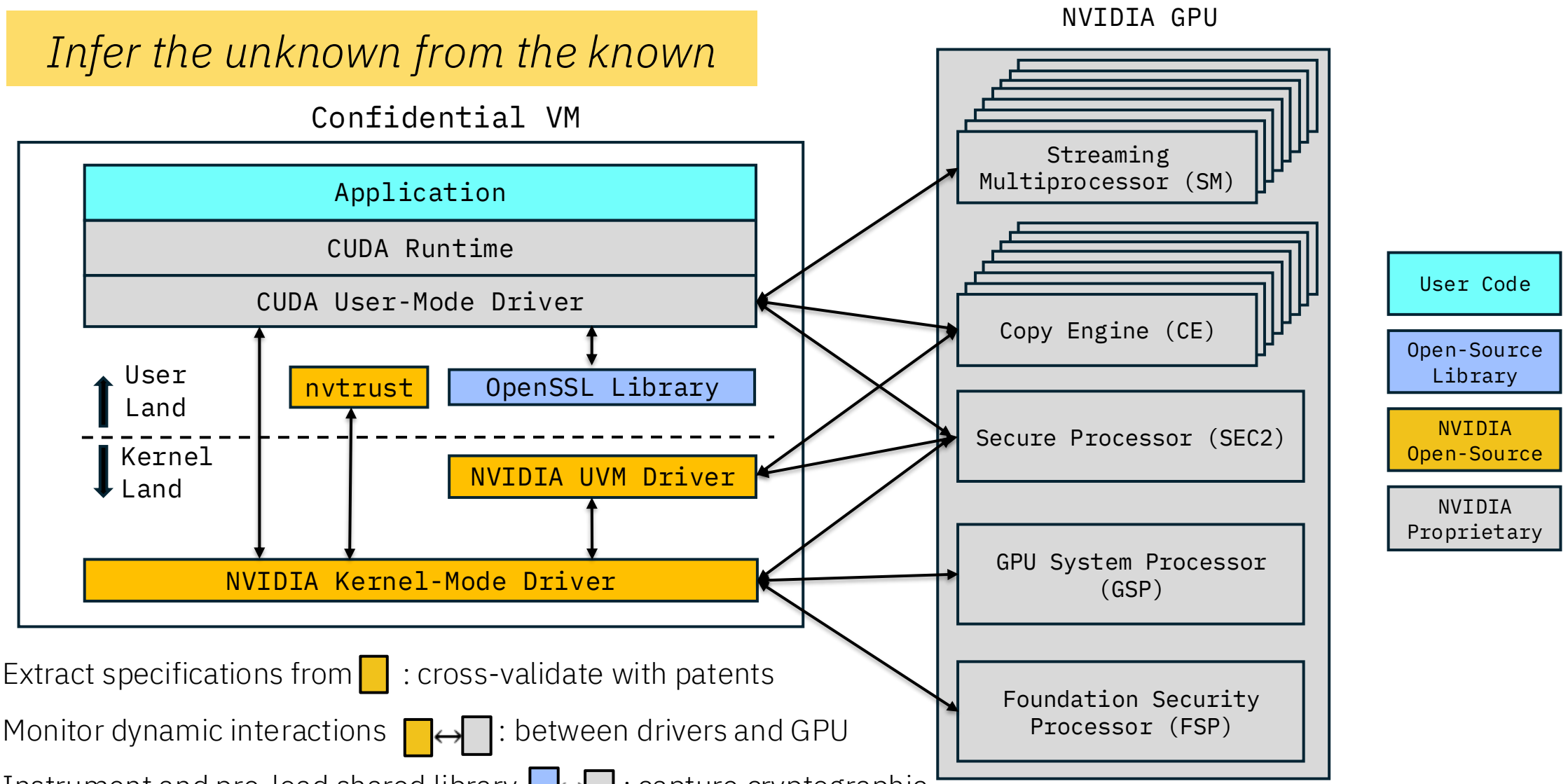
- Easy enablement
- No app code change

Security Researcher

- Deeper understanding of its security architecture
- Verification of the claimed security properties
- Identification of new attack surface and vectors

Methodology

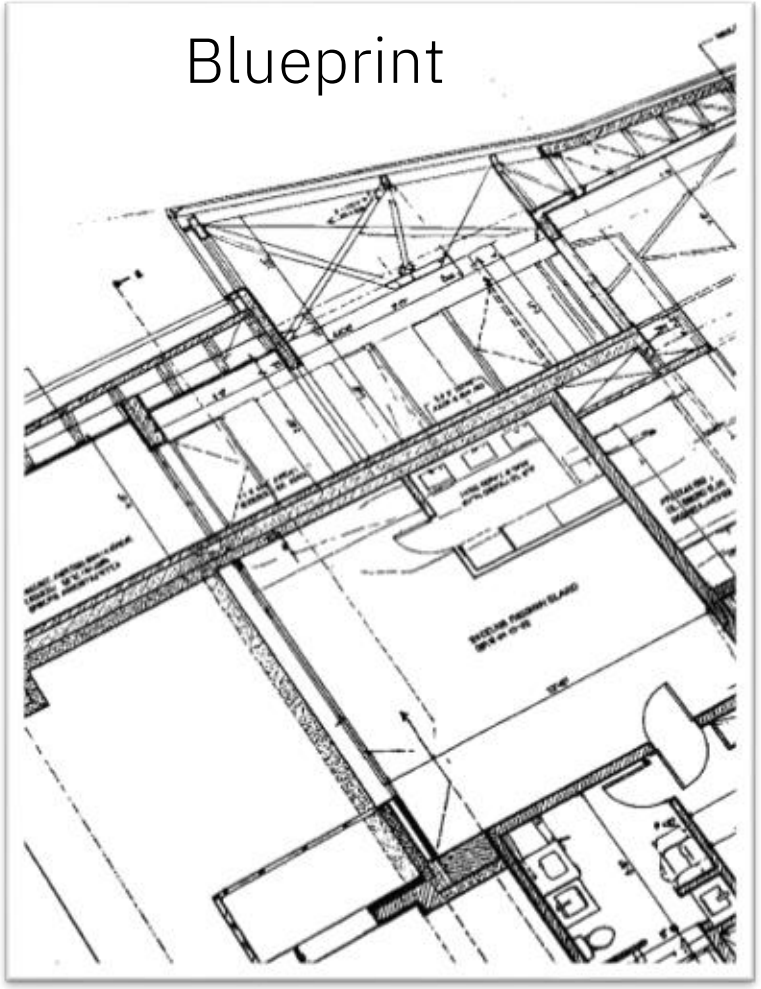
Infer the unknown from the known



- Extract specifications from : cross-validate with patents
- Monitor dynamic interactions : between drivers and GPU
- Instrument and pre-load shared library : capture cryptographic operations

Security Analysis

Blueprint

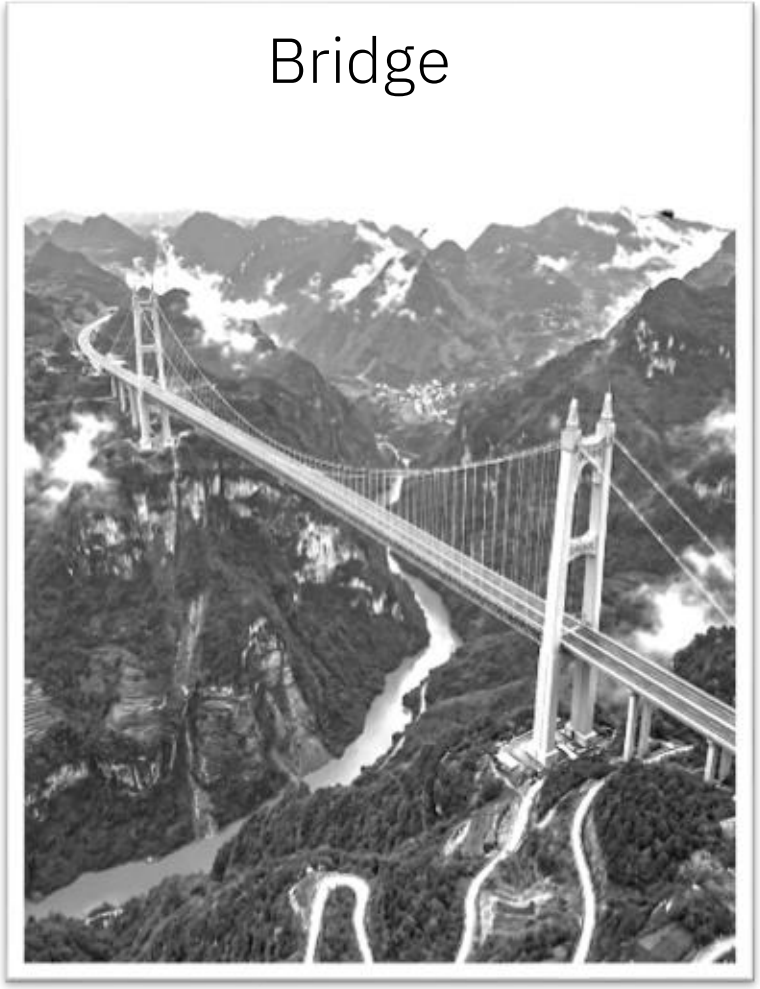


What are the key architectural components of GPU-CC?

Bootstrap

How is trust established between the CVM and the GPU?

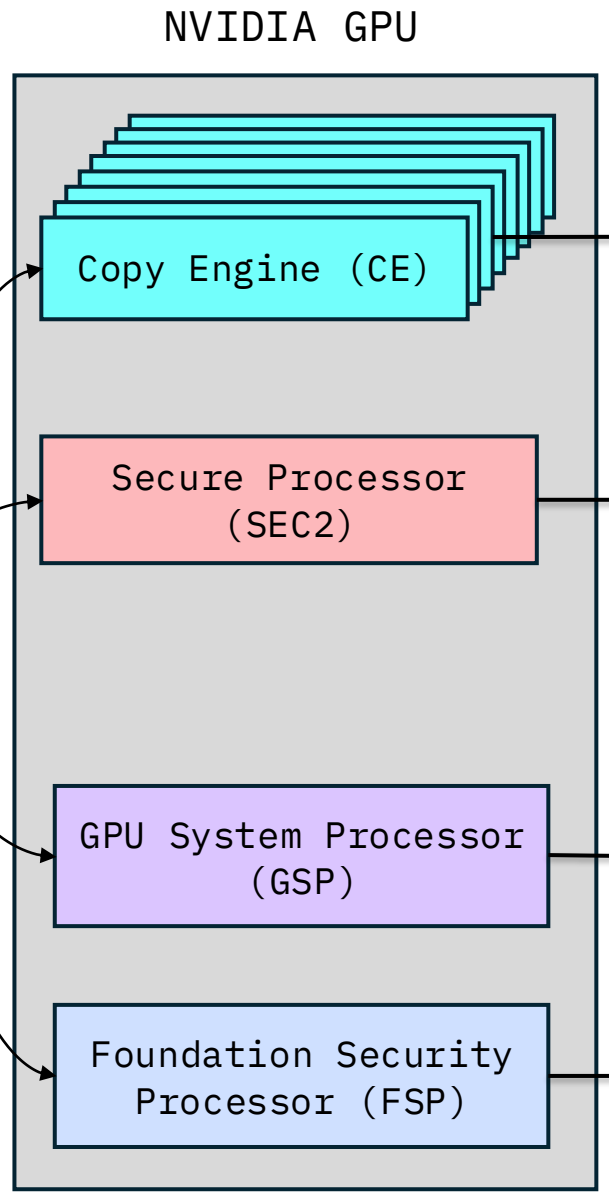
Bridge



Is data protected when crossing the untrusted interconnect?

Blueprint

GPU-CC functionalities are distributed across hardware engines



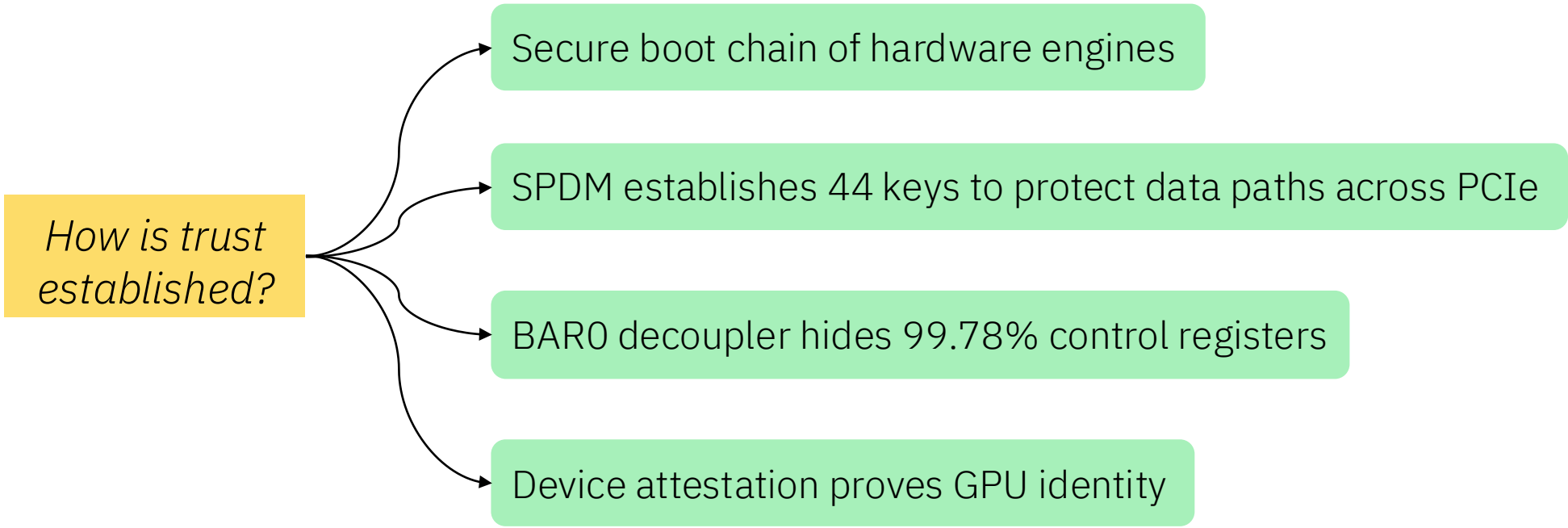
Manage memory transfers for GPU
→ H2D/D2H encrypted DMA copy

Specialized security engine for GPU-CC
→ Setup of GPU-CC memory and mode
→ Device attestation
→ Memory scrubbing
→ Secure data transmission
→ Secure command submission

Control plane of GPU
→ SPDM key negotiation
→ Resource manager

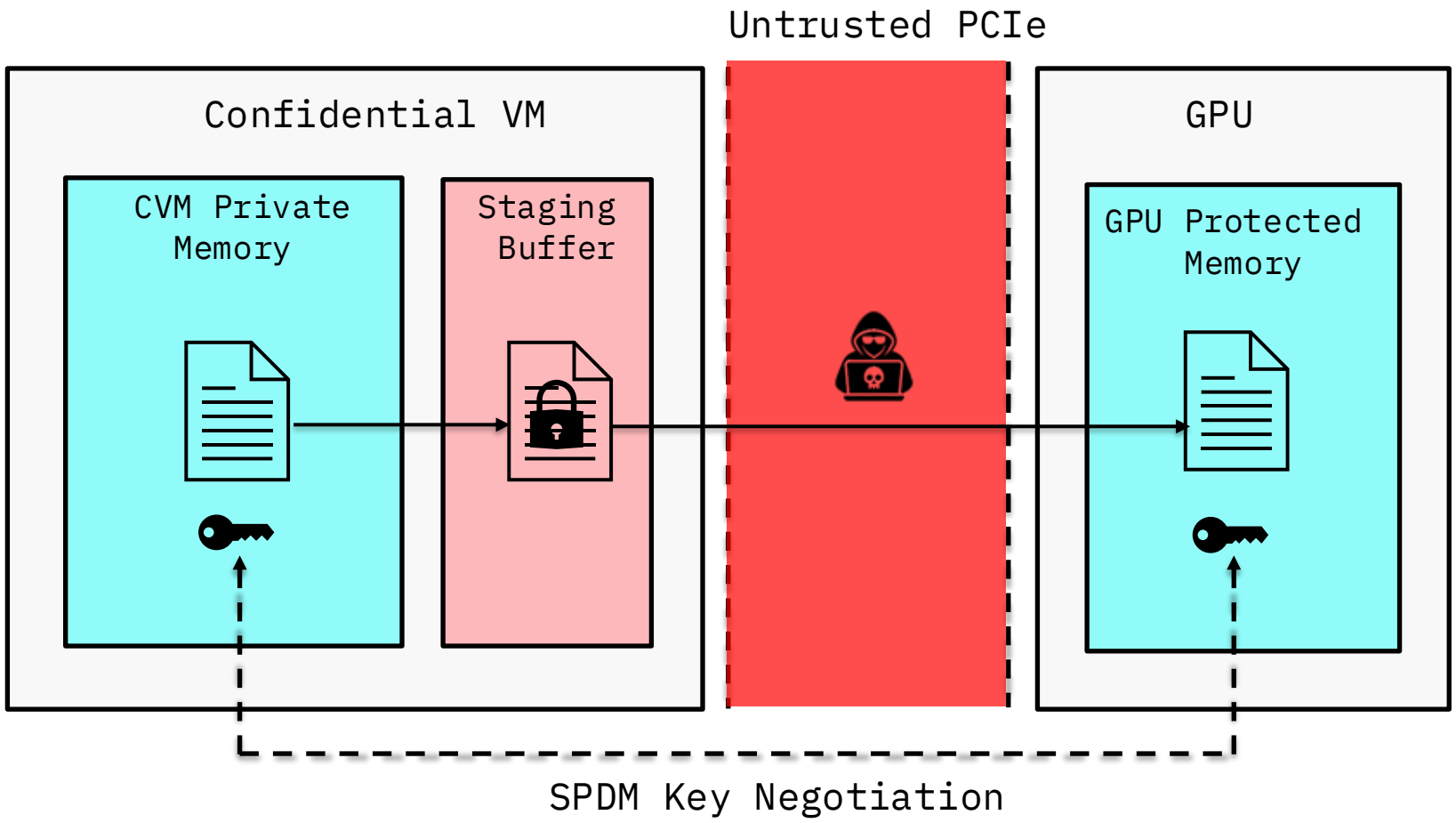
Hardware root of trust
→ Secure boot GSP

Bootstrap


How is trust established?
Secure boot chain of hardware engines
SPDM establishes 44 keys to protect data paths across PCIe
BAR0 decoupler hides 99.78% control registers
Device attestation proves GPU identity

Bridge

Data transfers across untrusted PCIe interface



Six data paths across PCIe

- CPU-GSP RPC
- CPU-GSP memory transfers
- GPU memory faults
- UVM operations
- Memory scrubbing command
- CUDA

Information exposure during data transmission

- RPC metadata
- Timing channels
- Command queue metadata
- Semaphore signals

Security Analysis on CPU-GSP RPC

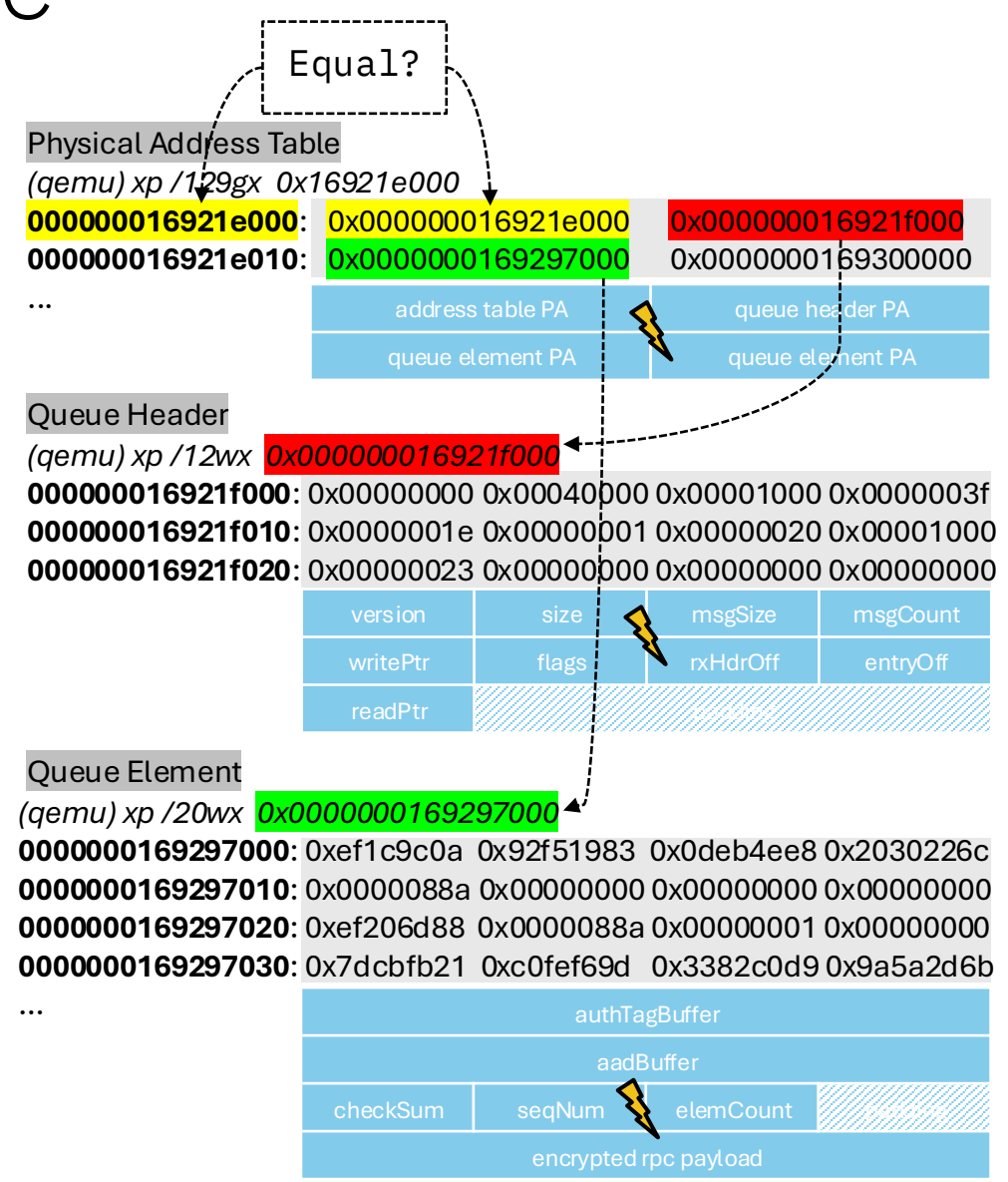
CPU-GSP RPC
- NVIDIA kernel-mode driver → GSP-RM API

Security Issue
- RPC payload is encrypted, but RPC queue metadata are not encrypted

Track Data Pointers
- Physical address table → queue header/element → queue metadata

Read/Write Access to Queue Metadata
- Violate the CPU-GSP RPC execution

Locate physical address table in a large memory space?
- Self-reference: first entry stores table's own address
- Scan memory for "address == address[0]"



Speculation on GPU-CC CUDA Data Transmission

CUDA is proprietary and self-contained

- *Limited interactions with open components*
- *Reasoned speculation*

```
__global__ void vecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    float A[256], B[256], C[256];
    ...
    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, 256*sizeof(float));
    cudaMalloc(&d_B, 256*sizeof(float));
    cudaMalloc(&d_C, 256*sizeof(float));

    cudaMemcpy(d_A, A, 256*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, 256*sizeof(float), cudaMemcpyHostToDevice);

    vecAdd<<<1,256>>>(d_A, d_B, d_C);

    cudaMemcpy(C, d_C, 256*sizeof(float), cudaMemcpyDeviceToHost);
}
```

User data

CUDA kernel

QMD: kernel launch configuration

Kernel arguments

Command queue structure

Traditional CUDA

```
CPU
  ├ Plaintext data
  ├ Plaintext kernel
  ├ Plaintext kernel Args
  ├ Plaintext QMD
  ├ Plaintext commands
                |
              PCIe
                |
              GPU
```

GPU-CC CUDA

```
CPU
  ├ Encrypted data
  ├ Encrypted CUDA kernel
  ├ Encrypted kernel args
  ├ Encrypted QMD
  ├ Encrypted commands
                |
          Staging buffer
                |
            PCIe DMA
                |
            GPU CPR
```

Key Takeaways (Session II)

CPU-CC + GPU-CC

- *Extend the trust boundary from CPU to GPU*
- *Use case: Confidential AI*
- *New attack surfaces and vectors*

Intel TDX

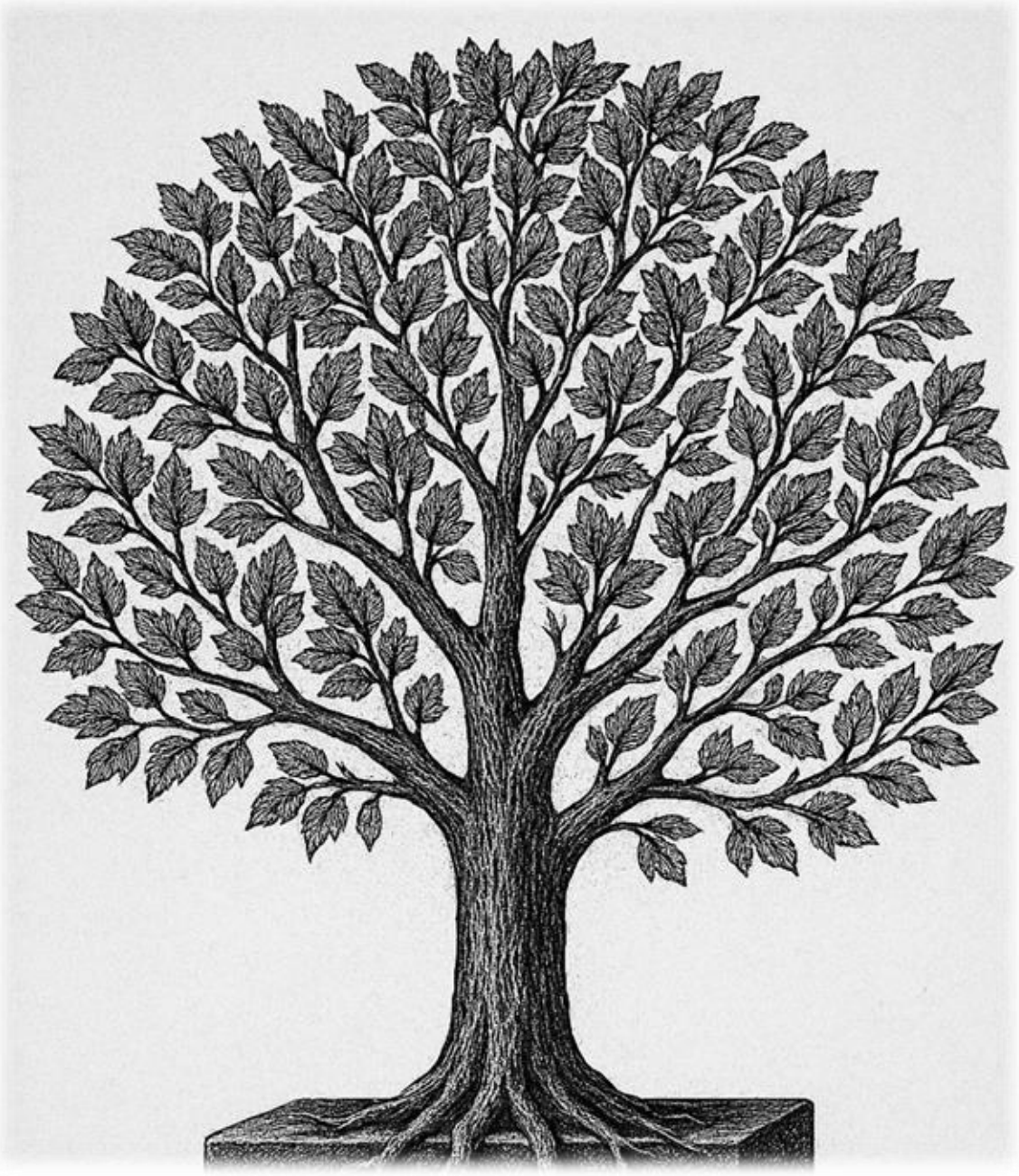
- *VM-based CPU-CC by Intel*
- *Confidentiality and integrity of data in confidential VM*
- *Remote verifiability*

Nvidia GPU-CC

- *Access control on GPU's VRAM and control registers*
- *Protect data-in-flight across untrusted I/O*
- *Device attestation*

Session II: Platforms
Q&A

Session III: Research



Session III: Research

Application

How to use confidential computing to address real world use cases?

DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation (Eurosys 24)

Pitfall

What are the security implications of misusing confidential computing?

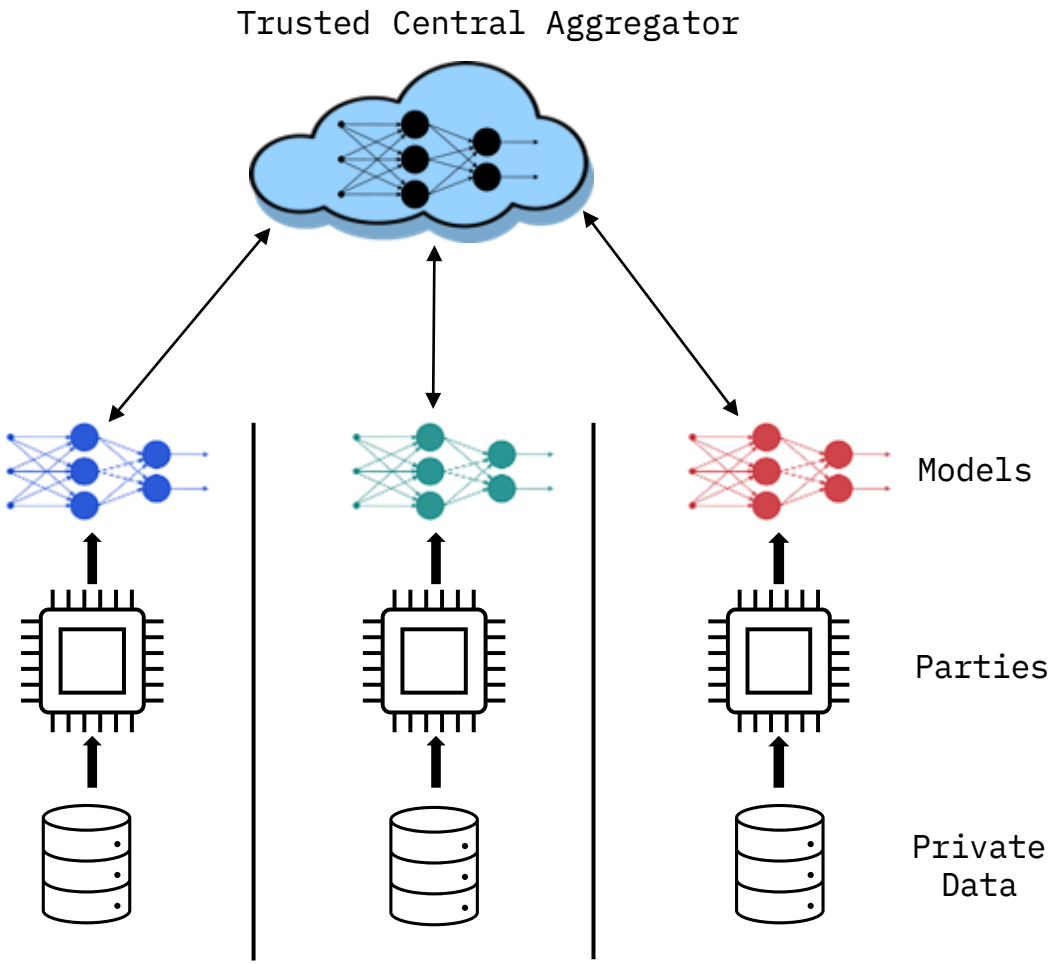
Crossing Shifted Moats: Replacing Old Bridges with New Tunnels to Confidential Containers (CCS 24)

Performance

How to overcome the performance bottleneck of confidential computing?

GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs (CCS 25)

Federated Learning



Federated learning paradigm

- Train a model together, but unwilling to share private data
- A central aggregator with multiple parties
- Local training → Models/Gradients uploading → Aggregation → Dispatching → Local updating

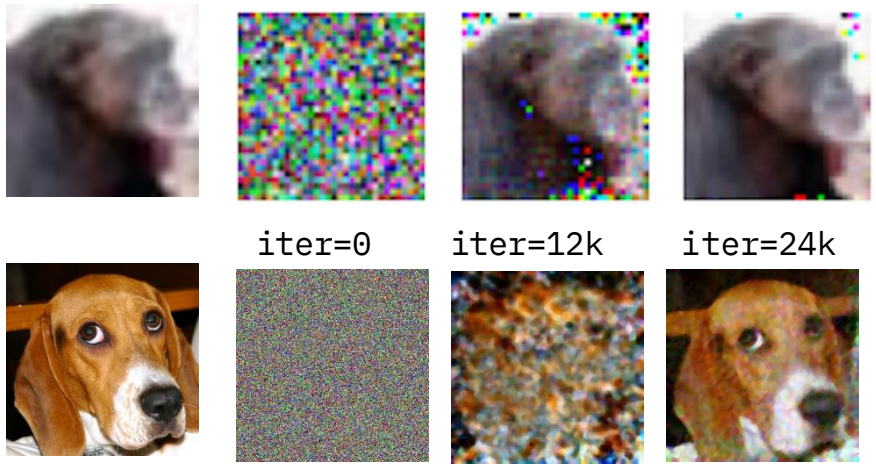
Security assumptions

- Model weights/gradients contain no info about training data
- Parties trust the central aggregator executing the process as claimed

What if the Central Aggregator is Controlled by Attackers?

Q: Can attackers reconstruct training data from model weights or gradients?
A: Yes!

Honest-but-curious attackers



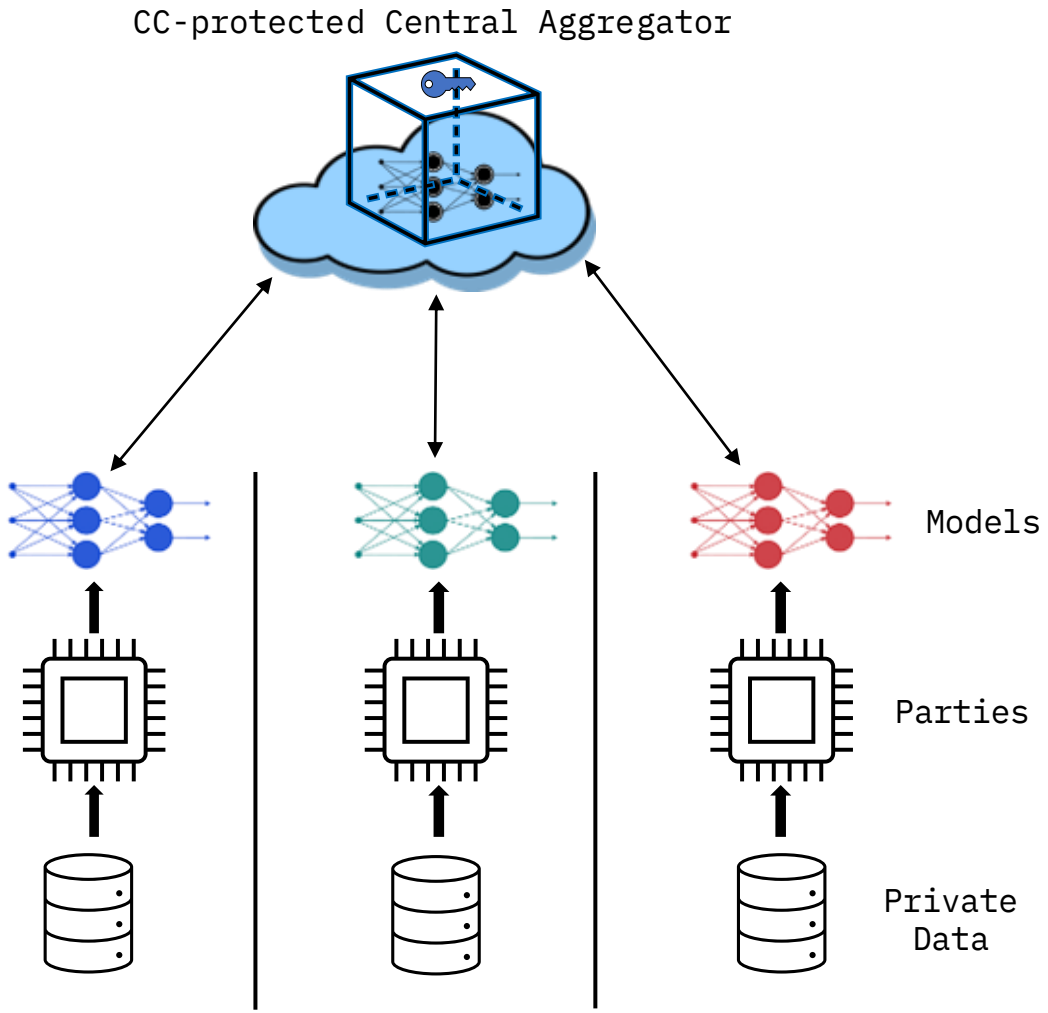
Active attackers

Actively intervene the aggregation process

- Modify model architecture (Fowl et al.)
- Modify model weights (Boenisch et al.)
- Achieve faster and better reconstruction

What if We Have Confidential Computing?

Q: Can CC make the central aggregator trustworthy?
A: Yes, but …



Benefits of CC

- Protect data-in-use in an isolated execution environment
- Parties can remotely verify the integrity of the central aggregator
- Establish secure channels for transferring data

Limitations

- Vulnerabilities and attack vectors targeting CC
- Unforeseen exploits in the future
- Breached CC may leak all data for reconstruction

Key Insights

Data Concentration

- Should not put all data into one aggregator

Aggregator Authenticity

- Should not work with compromised aggregators

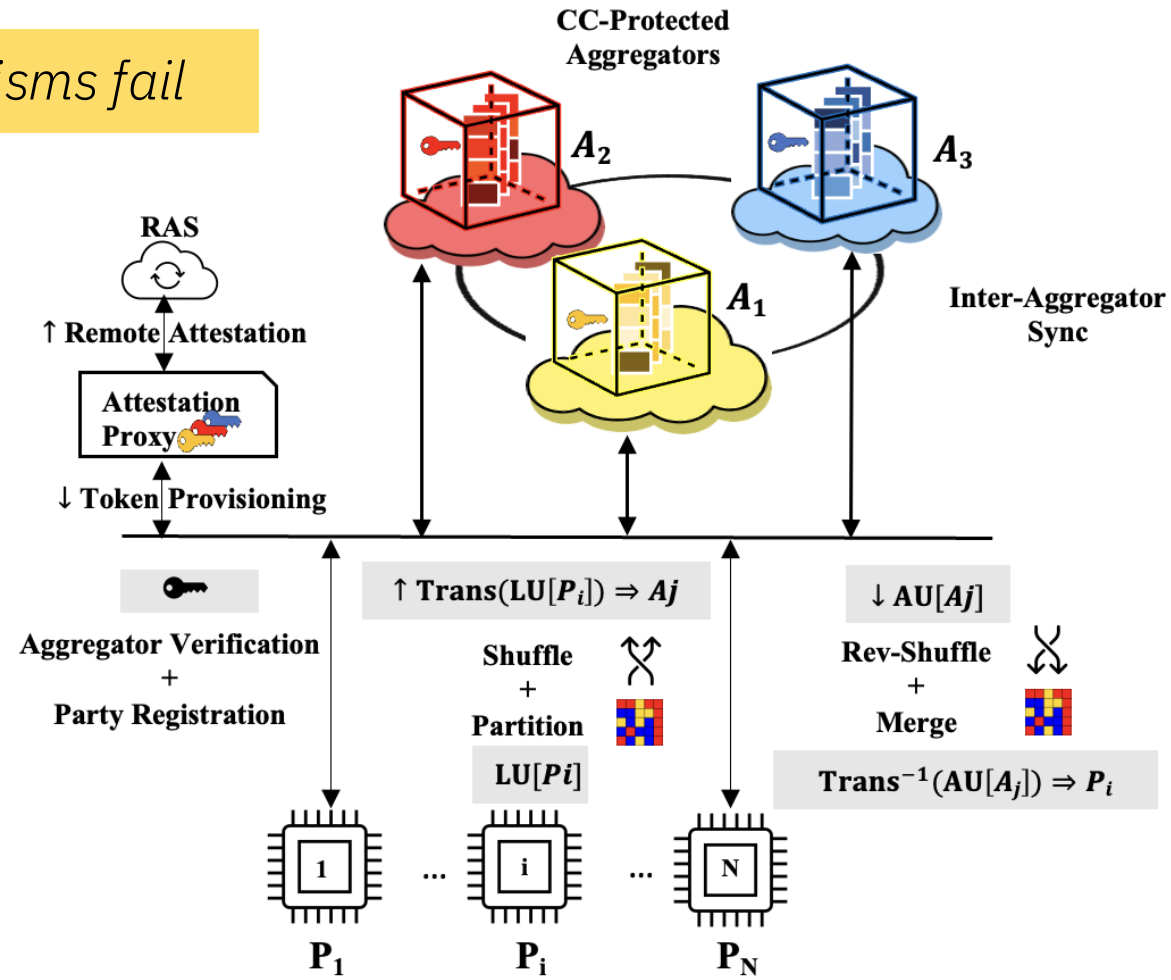
Worst-Case Scenario

- Prevent reconstruction even if aggregators are breached

Security Design

Defense-in-Depth : Resilient even if some mechanisms fail

- Decentralized Aggregation
  → Data Concentration
- Trustworthy Multi-Aggregator Authentication
  → Aggregator Authenticity
- Model Partitioning and Parameter Shuffling
  → Worst-case Scenario



Decentralized Aggregation

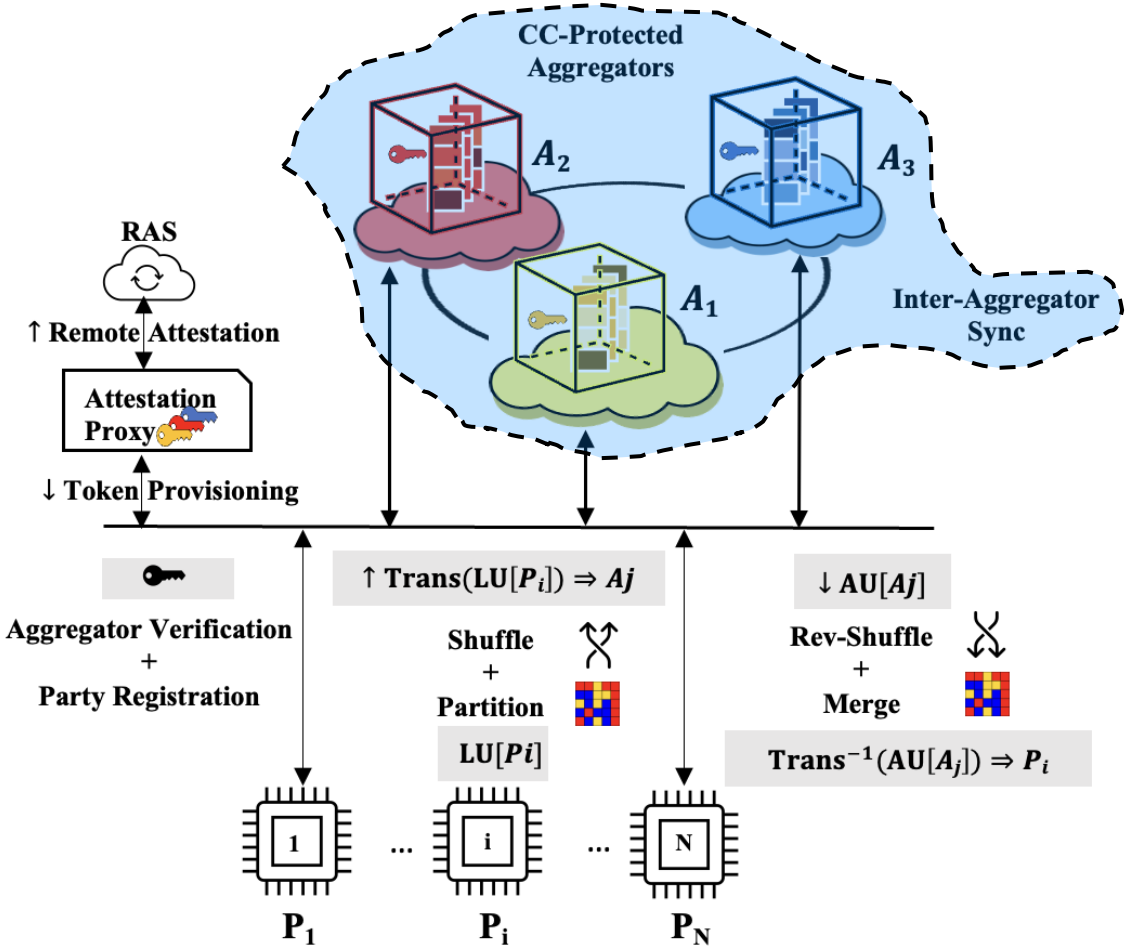
Separate model updates into multiple protected domains

Multiple CC-protected aggregators

- Each aggregator only has a subset of a model update
- Introduce inter-aggregator sync to orchestrate FL

Attack mitigation

- *Attackers need to breach all CC-protected aggregators to get a complete model update*



Trustworthy Multi-Aggregator Authentication

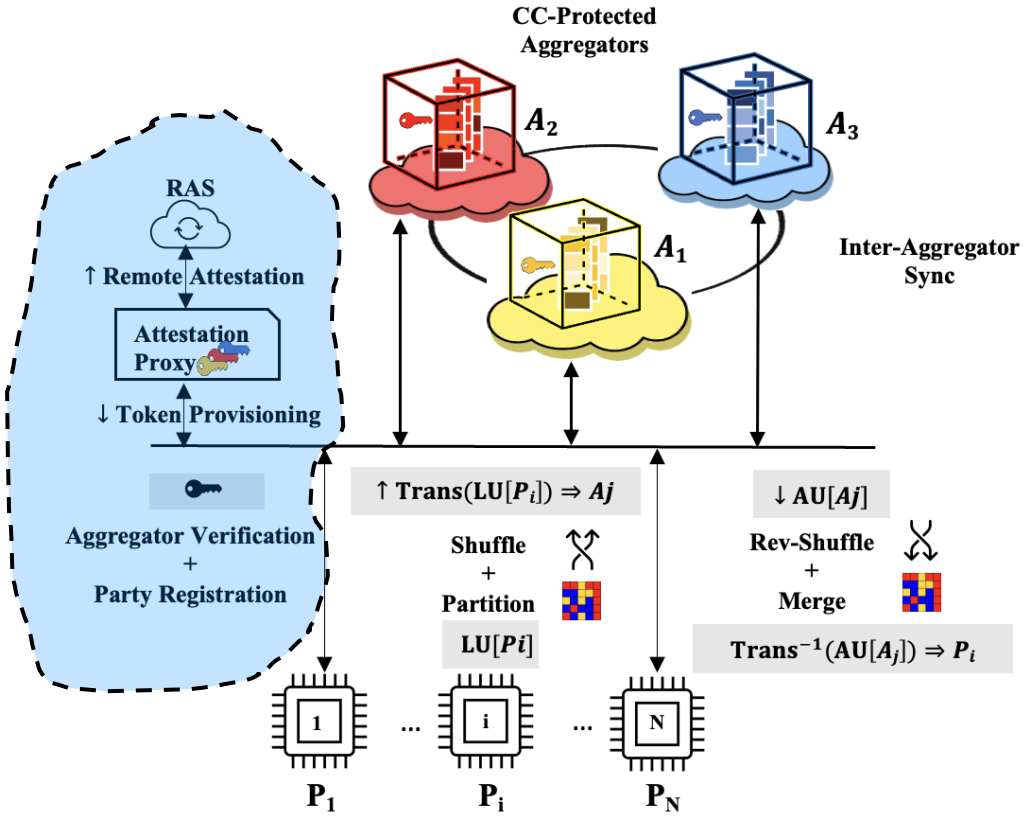
Only authenticated CC-protected aggregators are allowed in training

Phase I: Launching trustworthy aggregators

- Remote attestation of CC-protected aggregators
- Inject an authentication token into the launched aggregators

Phase II: Authenticating aggregators and parties

- Party verifies the embedded authentication token
- Party establishes a TLS channel and registers with the aggregator
- Repeat this step with all aggregators



Attack mitigation

- *Parties will not register with compromised aggregators*

Model Partitioning and Parameter Shuffling

Leaked data cannot be recovered to its original form

Model partitioning

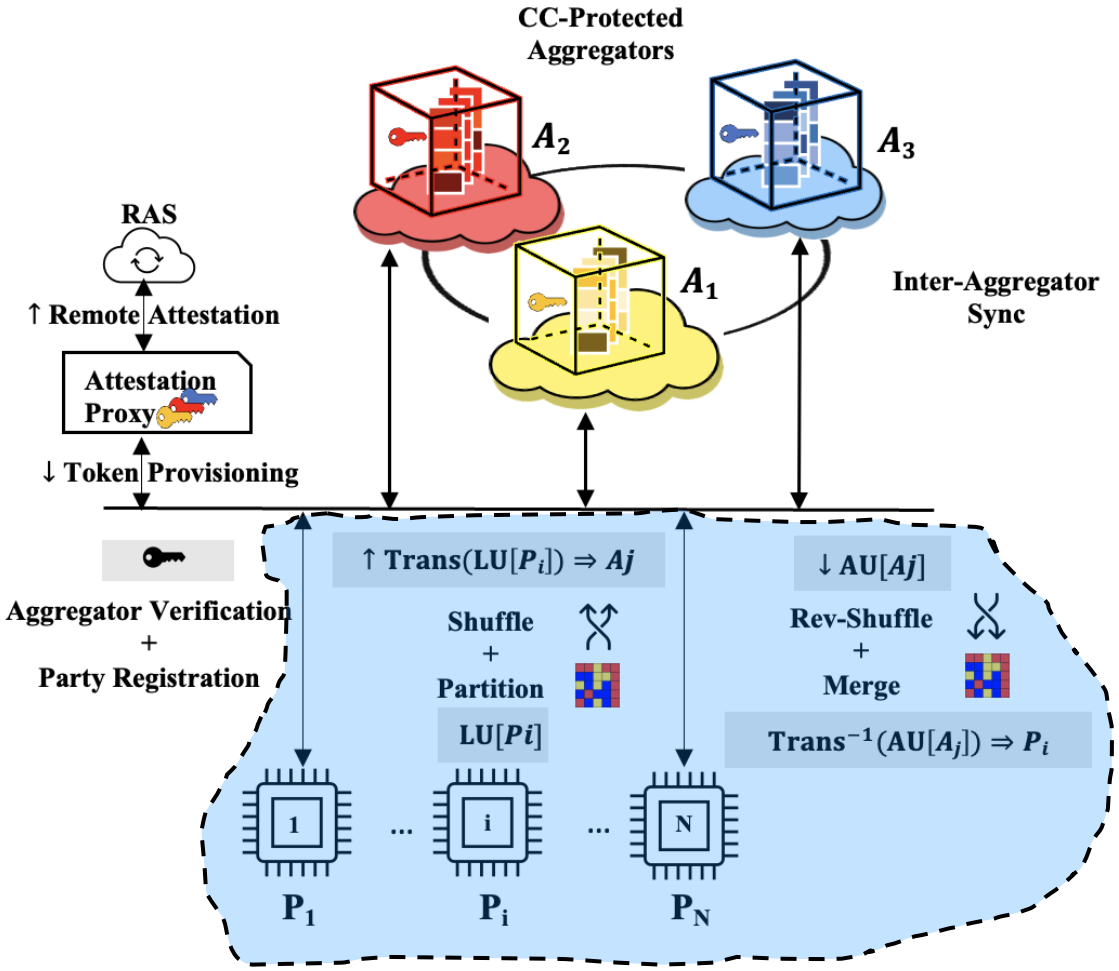
- Each model update can be treated as a flattened vector
- Partition to multiple sub-vectors (uploading to different aggregators) based on a shared *model mapper*

Parameter shuffling

- Each partitioned sub-vector can be further shuffled at the parameter level
- The shuffling is based on a shared *permutation key* and a dynamic *training id*

Attack mitigation

- *A fragmentary and shuffled model update cannot be used for reconstruction*



Security Analysis

Assume data are leaked from a CC-protected aggregator

Without DeTA (Full*)

- DLG: 66.6% recognizable (MSE $< 1 \times 10^{-3}$)
- iDLG: 83.7% recognizable (MSE $< 1 \times 10^{-3}$)
- IG: 100% recognizable (cosine distance < 0.01)

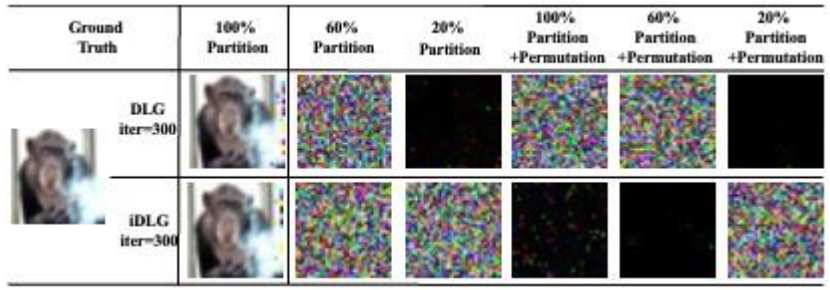
With DeTA (Five Scenarios)

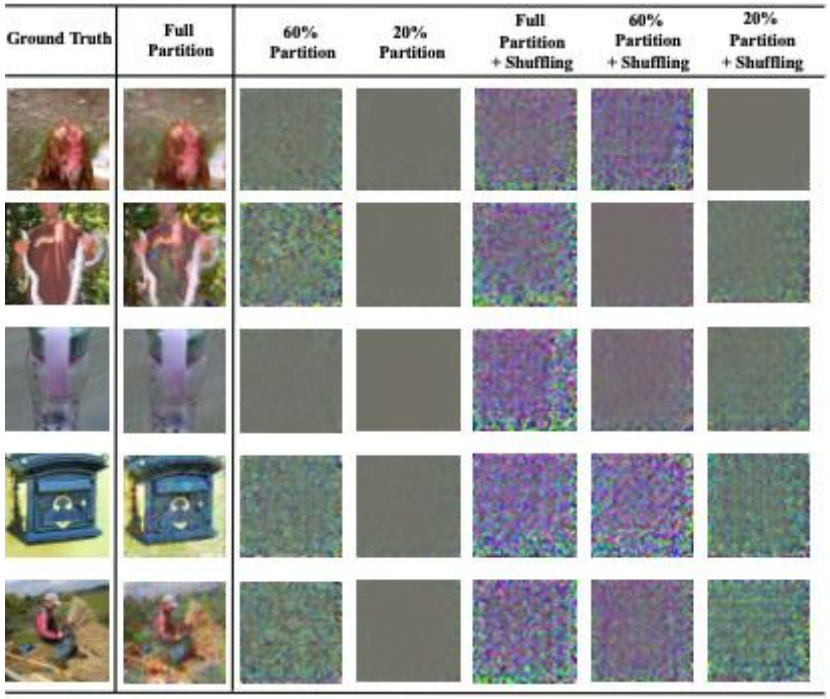
- 0% recognizable
- Smaller partition and combination w/ shuffling → Better results

DLG	Partition			Partition+Shuffle		
MSE	**Full***	0.6†	0.2†	Full‡	0.6‡	0.2‡
$[0, 1 \times 10^{-3})$	**66.6%**	0%	0%	0%	0%	0%
$[1 \times 10^{-3}, 1)$	**1.3%**	0%	0%	0%	0%	0%
$[1, 1 \times 10^{2})$	**8.1%**	38.9%	20.5%	0%	0%	0.2%
$\geq 1 \times 10^{2}$	**24.0%**	61.1%	79.5%	100%	100%	99.8%

iDLG	Partition			Partition+Shuffle		
MSE	**Full***	0.6†	0.2†	Full‡	0.6‡	0.2‡
$[0, 1 \times 10^{-3})$	**83.7%**	0%	0%	0%	0%	0%
$[1 \times 10^{-3}, 1)$	**1.5%**	0%	0%	0%	0%	0%
$[1, 1 \times 10^{2})$	**6.6%**	66.5%	18.3%	0.2%	0.2%	0.7%
$\geq 1 \times 10^{2}$	**8.2%**	33.5%	81.7%	99.8%	99.8%	99.3%

IG	Partition			Partition+Shuffle		
Cosine Distance	**Full***	0.6†	0.2†	Full‡	0.6‡	0.2‡
[0, 0.01)	**100%**	0%	0%	0%	0%	0%
[0.01, 0.2)	**0%**	0%	0%	0%	0%	0%
[0.2, 0.4)	**0%**	100%	0%	0%	0%	0%
[0.4, 0.6)	**0%**	0%	98%	0%	0%	0%
[0.6, 0.8)	**0%**	0%	2%	0%	0%	0%
[0.8, 1]	**0%**	0%	0%	100%	100%	100%





Session III: Research

Application

How to use confidential computing to address real world use cases?

DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation (Eurosys 24)

Pitfall

What are the security implications of misusing confidential computing?

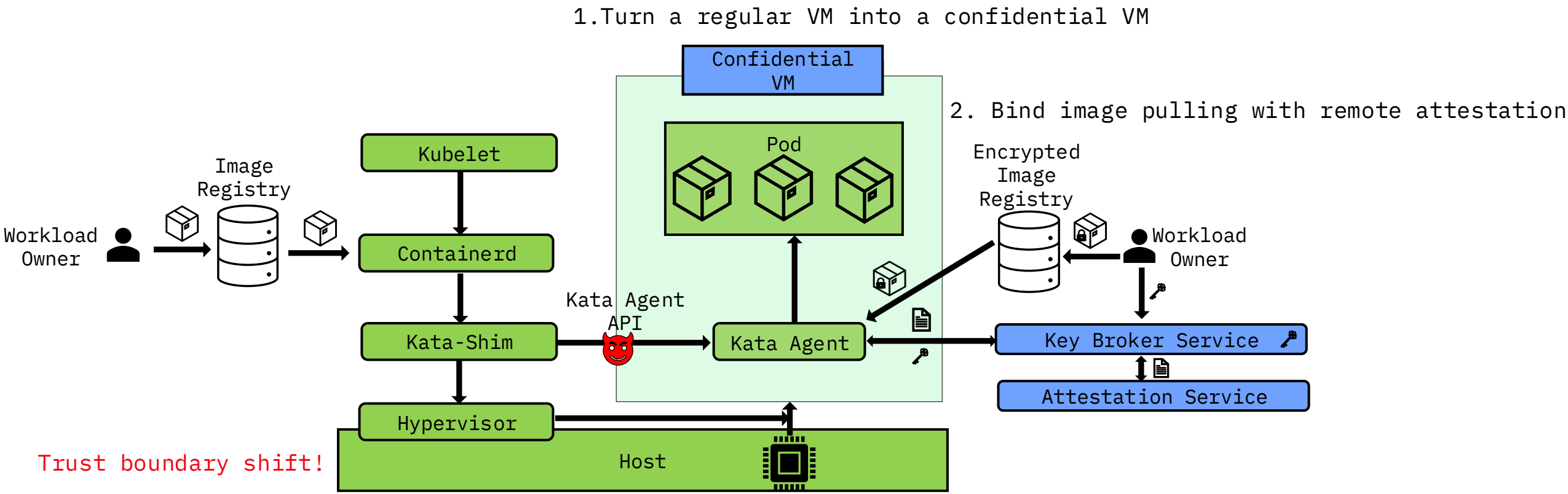
Crossing Shifted Moats: Replacing Old Bridges with New Tunnels to Confidential Containers (CCS 24)

Performance

How to overcome the performance bottleneck of confidential computing?

GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs (CCS 25)

Kata Containers → Confidential Containers



Misuse of Kata Agent API


Information Leakage


Execution Tampering

Attacks	Types	Invoked APIs
Leak neighboring container's memory		*PullImage, CreateContainer, StartContainer*
Obtain runtime metrics information		*GetMetrics*
Pause/resume/remove containers		*PauseContainer, ResumeContainer, RemoveContainer*
Set bad date and time		*SetGuestDateTime*
Write sensitive files		*CopyFile*

Categorization: Kata Agent API

Q: Can we simply disable Kata Agent API from host?
A: No, some APIs are essential for resource allocation

Host Exclusive

- *AddARPNeighbors*
- *CreateSandbox*
- *DestroySandbox*
- *GetIPTables*
- *GetVolumeStats*
- *ResizeVolume*
- *SetIPTables*
- *UpdateInterface*
- *UpdateRoutes*

- *Only resource allocation related*
- *Host access: should be allowed*
- *Owner access: not needed*

Owner Exclusive

- *CopyFile*
- *ExecProcess*
- *PauseContainer*
- *PullImage*
- *RemoveContainer*
- *ReseedRandomDev*
- *ResumeContainer*
- *SetGuestDateTime*
- *StatsContainer*
- *TtyWinResize*
- *UpdateContainer*
- *CreateContainer*
- *StartContainer*

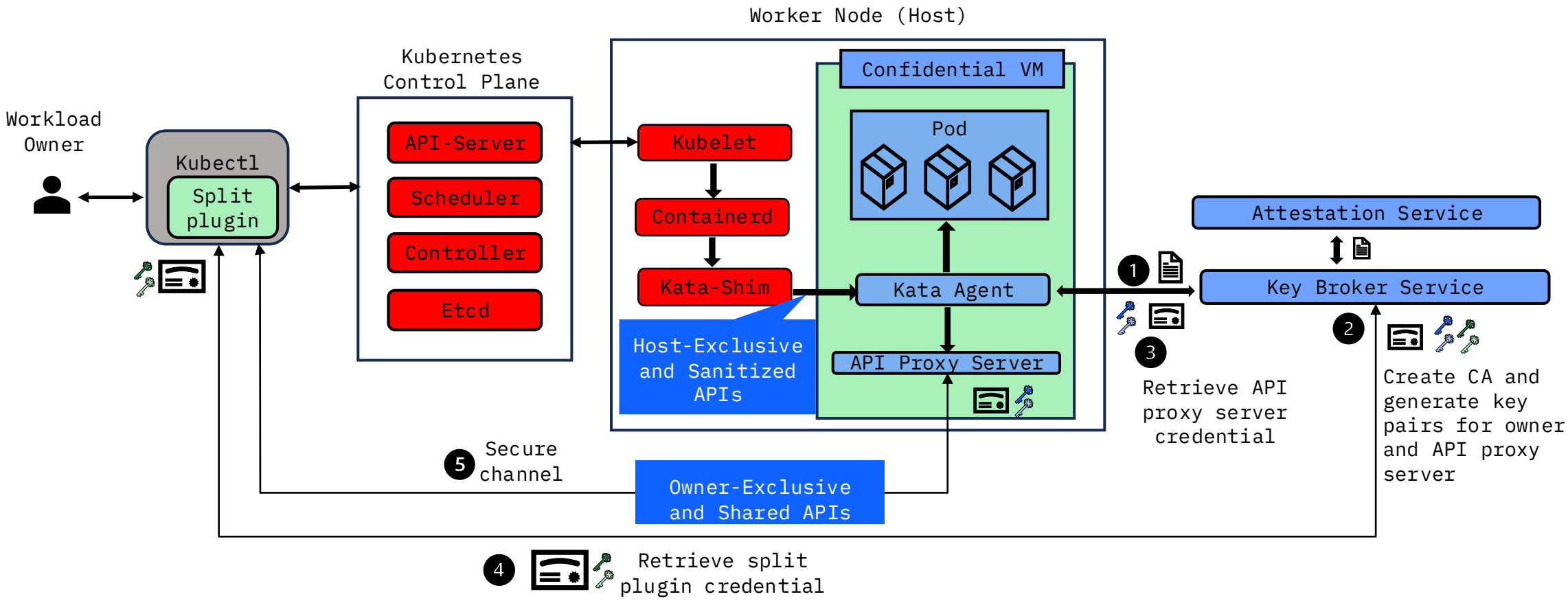
- *Risks of leaking information and compromising execution*
- *Host access: should be disabled*
- *Owner access: should be allowed*

Shared

- *Check*
- *GetOOMEvent*
- *ListInterfaces*
- *ListRoutes*
- *OnlineCPUMem*
- *Version*
- *WaitProcess*
- *CloseStdin*
- *GetMetrics*
- *GetGuestDetails*
- *ReadStderr*
- *ReadStdout*
- *SignalProcess*
- *WriteStdin*

- *Needed by both host and owner*
- *Host access: inspection and sanitization of the input/output*
- *Owner access: should be allowed*

SplitAPI Approach



Mitigation


Information Leakage

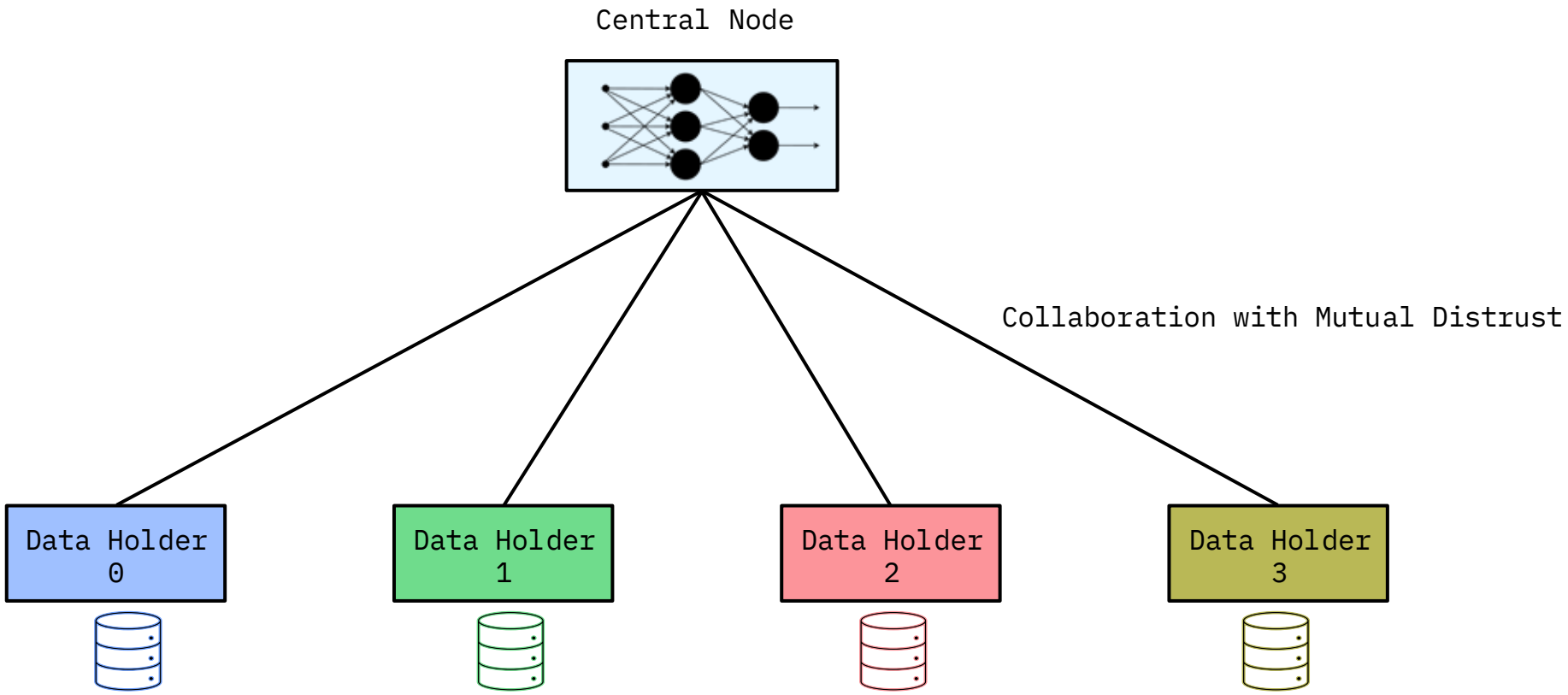

Execution Tampering

Attacks	Types	Invoked APIs	SplitAPI Mitigation
Leaking neighboring container's memory		*PullImage, CreateContainer, StartContainer*	Owner Exclusive
Obtaining runtime metrics information		*GetMetrics*	Sanitized
Pausing/resuming/removing containers		*PauseContainer, ResumeContainer, RemoveContainer*	Owner Exclusive
Setting bad date and time		*SetGuestDateTime*	Owner Exclusive
Writing sensitive files		*CopyFile*	Owner Exclusive

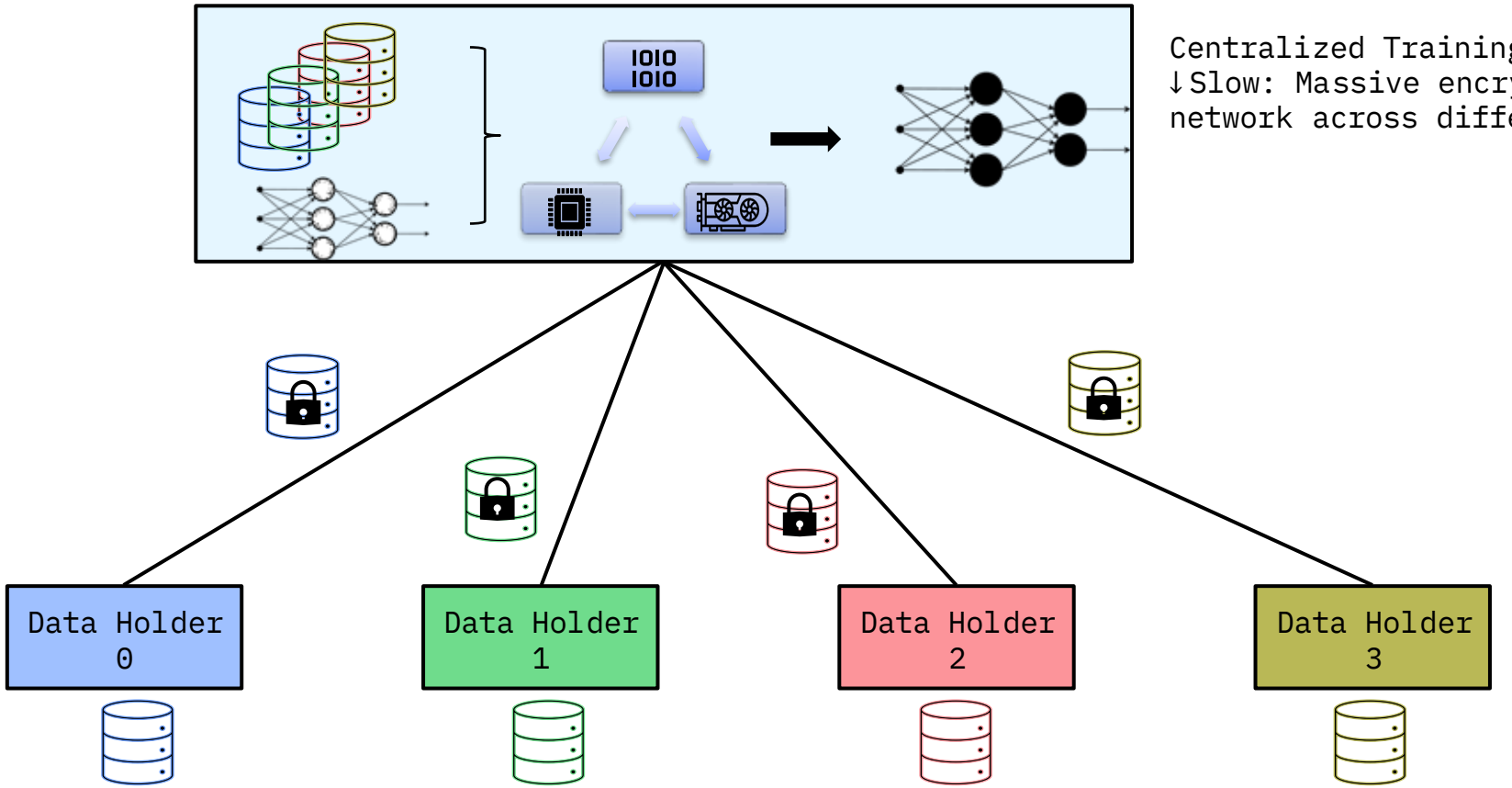
Session III: Research

Application	*How to use confidential computing to address real world use cases?*
	DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation (Eurosys 24)
Pitfall	*What are the security implications of misusing confidential computing?*
	Crossing Shifted Moats: Replacing Old Bridges with New Tunnels to Confidential Containers (CCS 24)
Performance	*How to overcome the performance bottleneck of confidential computing?*
	GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs (CCS 25)

Collaborative Learning

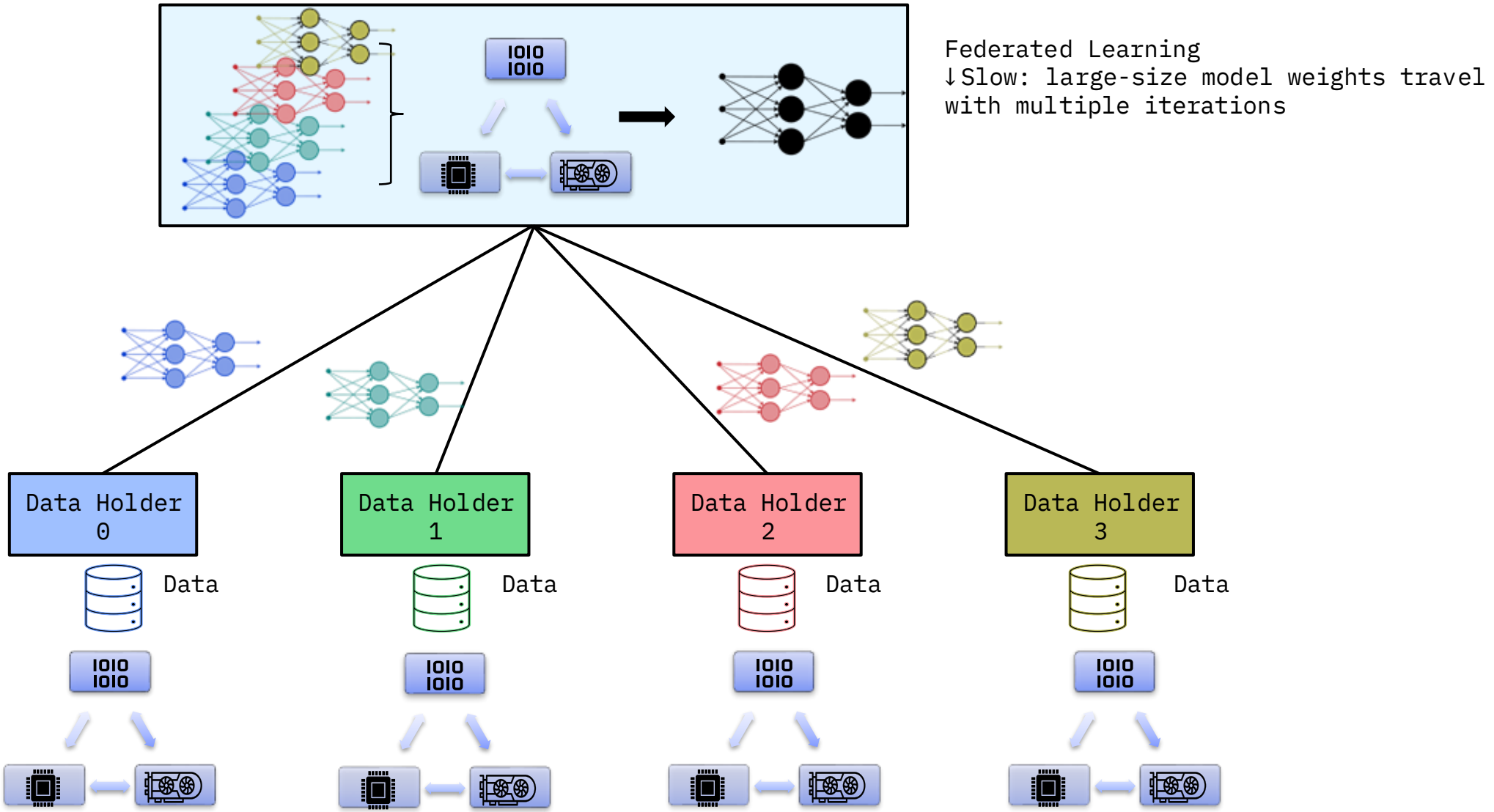


Collaborative Learning (Data Aggregation)

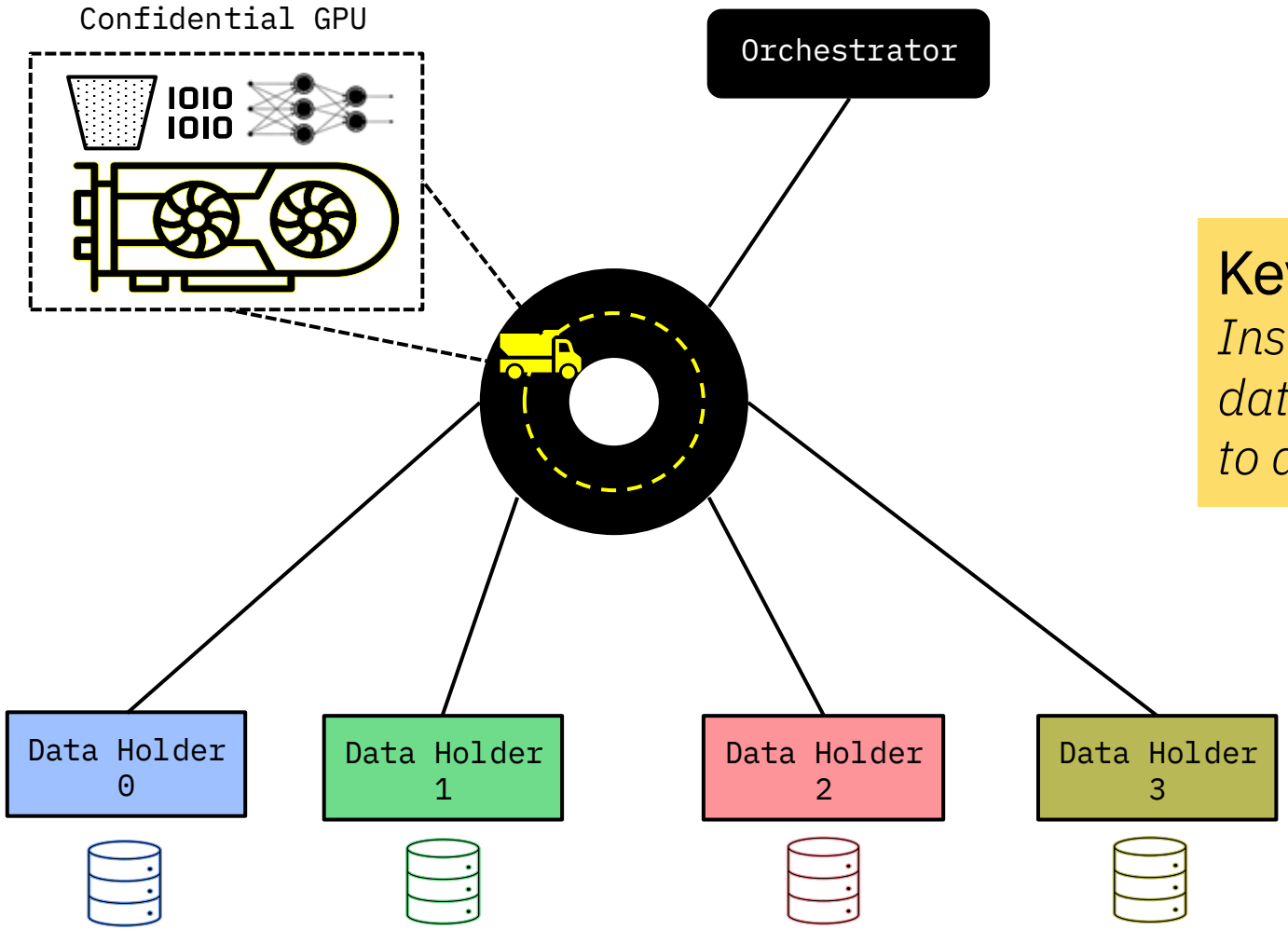


Centralized Training
↓Slow: Massive encrypted data sharing via network across different security domains

Collaborative Learning (Model Aggregation)



Our Solution

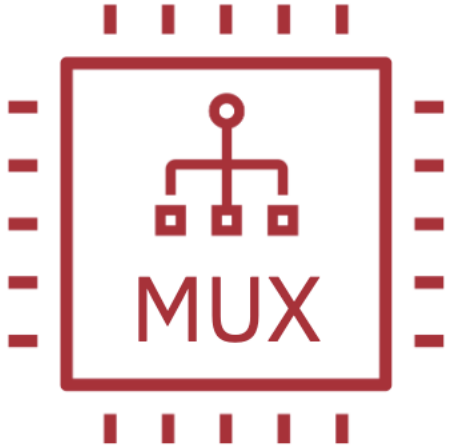


Key Idea: GPU Traveling

Instead of moving large models or data, we let confidential GPU travel to data holders to load data.

Key Techniques

Physical layer traveling

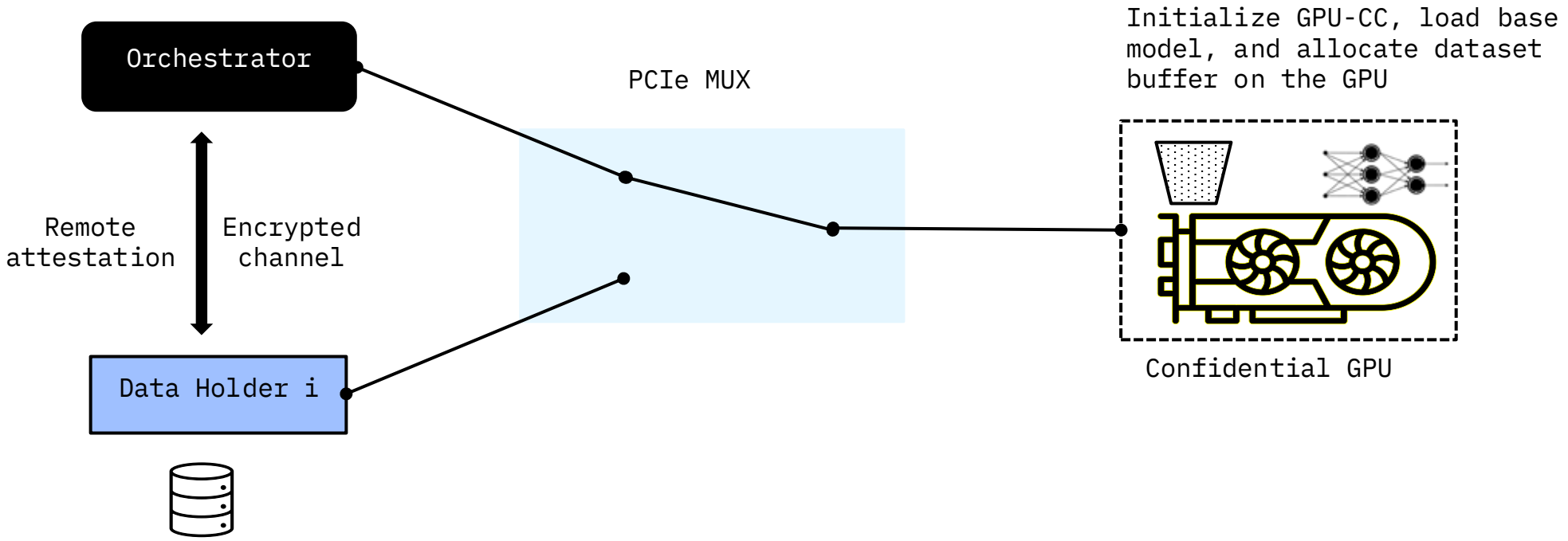


PCIe MUX
Routing a physical GPU to multiple CVMs

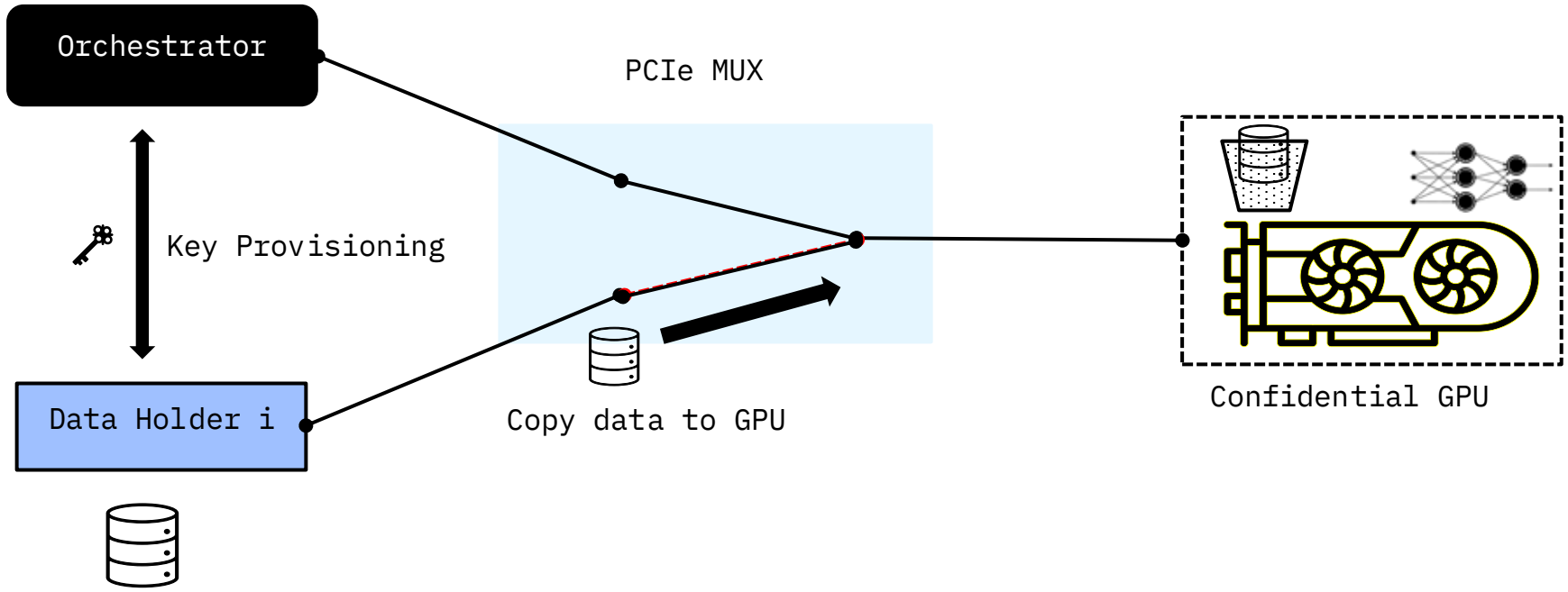
Security layer traveling

Encrypted transfer via PCIe
Sharing the key = sharing the GPU

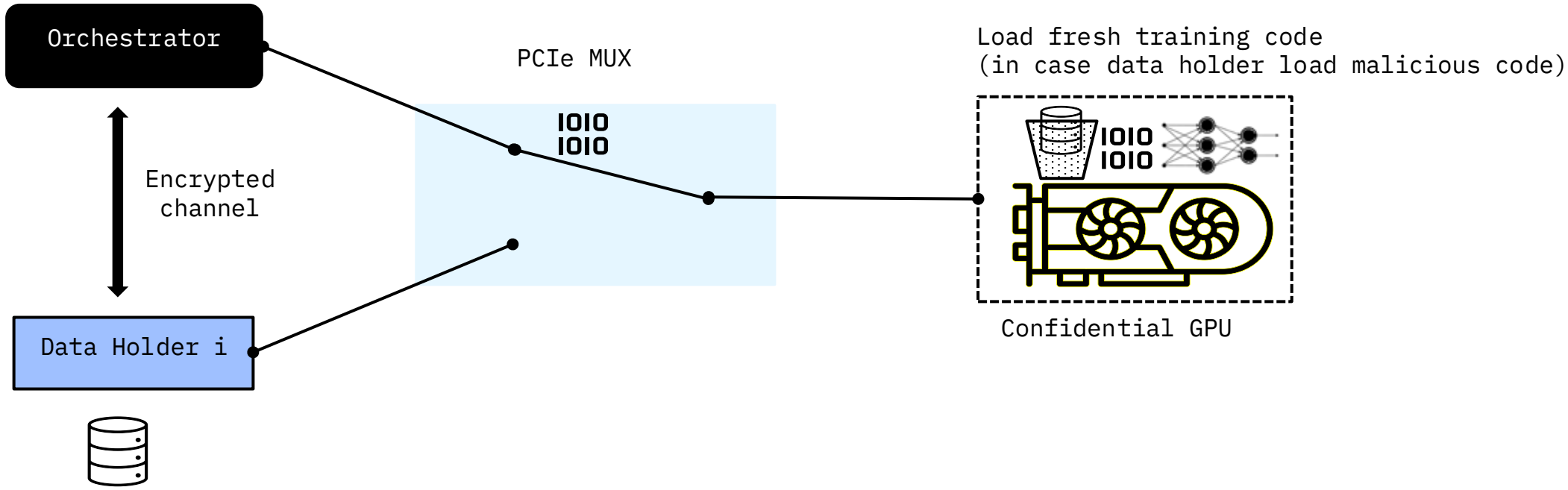
How it Works: Initialization



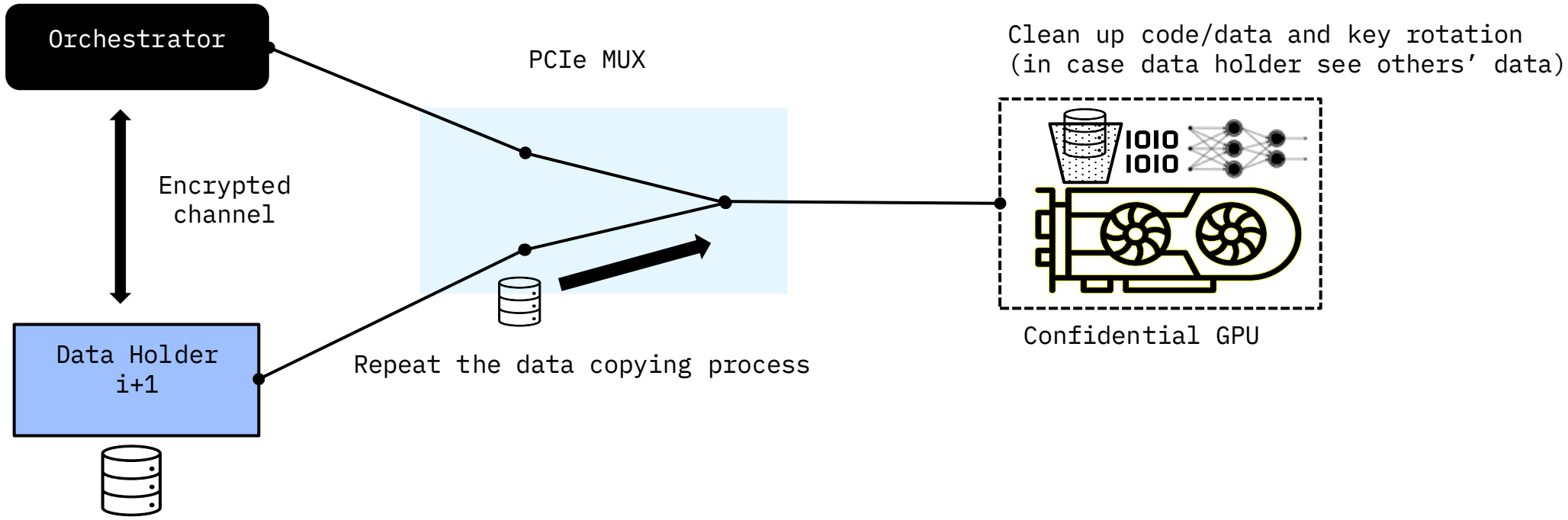
How it Works: Physical/Security Layer Traveling



How it Works: Training



How it Works: Cleanup and Switch

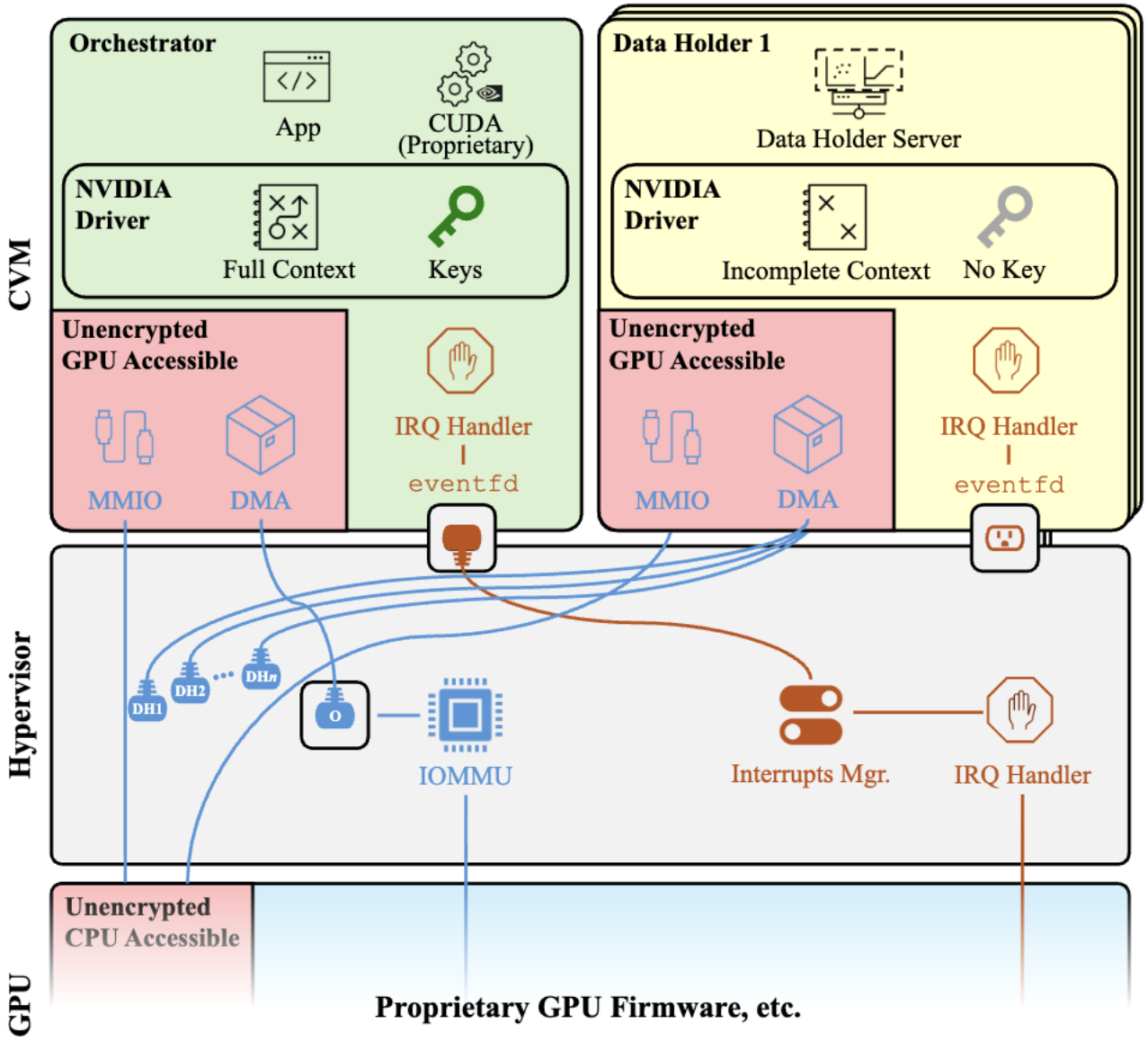


Implementation

Challenge

- *Cannot modify Nvidia's proprietary components (CUDA, GPU firmware, etc.)*

- Intel TDX
- Nvidia H100 GPU-CC
- VFIO-based PCIe MUX
- Modify open-source Nvidia kernel-mode and UVM driver
  - Key import/export
  - Security context sync



Evaluation

- Evaluation on *llm.c* with GPT-2
- Modify the *llm.c*'s data loader to use TLS and GPU Traveling respectively
- 512 MB buffer size on the GPU
- Test multiple network bandwidths (4Mbps to Uncapped)
- Keep data transmission time constant and beat TLS with different bandwidths

	Baseline			Ours		
	Training (s)	**Tx (s)**	**Tx Percentage**	**Training (s)**	**Tx (s)**	**Tx Percentage**
4M	1230.00	1115.88	47.568%	1230.26	1.01	0.082%
20M	1231.39	230.84	15.787%	1229.39	1.03	0.084%
100M	1231.17	60.36	4.674%	1230.57	1.02	0.083%
500M	1230.73	15.86	1.272%	1229.67	0.98	0.080%
Uncapped	1229.36	7.34	0.594%	1231.46	0.98	0.079%

Key Takeaways (Session III)

DeTA

- *Model updates can leak private training data in FL*
- *Centralized aggregation = a single point of failure*
- *Decentralized aggregation with multiple CC-protected aggregators and model update obfuscation*

SplitAPI

- *Trust boundary shifts in confidential containers*
- *Vulnerable Kata APIs enable data leaks & execution tampering*
- *SplitAPI for confidential container's control interface*

GPU Traveling

- *Model/data movements in collaborative learning incur high communication cost*
- *Let GPU travel to data holders to load data*
- *Enable physical and security layer traveling to hot-plug confidential GPU*

Session III: Research Q&A

Reference

Session II

- “Intel TDX Demystified: A Top-Down Approach,” ACM Computing Surveys 2024
  Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, James Bottomley
- “Blueprint, Bootstrap, and Bridge: A Security Look at NVIDIA GPU Confidential Computing,” MLSys 2026
  Zhongshu Gu, Enriquillo Valdez, Salman Ahmed, Julian James Stephen, Michael Le, Hani Jamjoom, Shixuan Zhao, Zhiqiang Lin

Session III

- “DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation,” Eurosys 2024
  Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, K. R. Jayaram, Enriquillo Valdez, Ashish Verma
- “Crossing Shifted Moats: Replacing Old Bridges with New Tunnels to Confidential Containers,” CCS 2024
  Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Christophe de Dinechin, Pau-Chen Cheng, Hani Jamjoom
- “GPU Travelling: Efficient Confidential Collaborative Training with TEE-Enabled GPUs,” CCS 2025
  Shixuan Zhao, Zhongshu Gu, Salman Ahmed, Enriquillo Valdez, Hani Jamjoom, Zhiqiang Lin