# **OpenTDX:** Emulating TDX Machine

**Jaewon Hur**

**SSLAB Georgia Tech**

**Supervisor: Taesoo Kim**

# Agenda

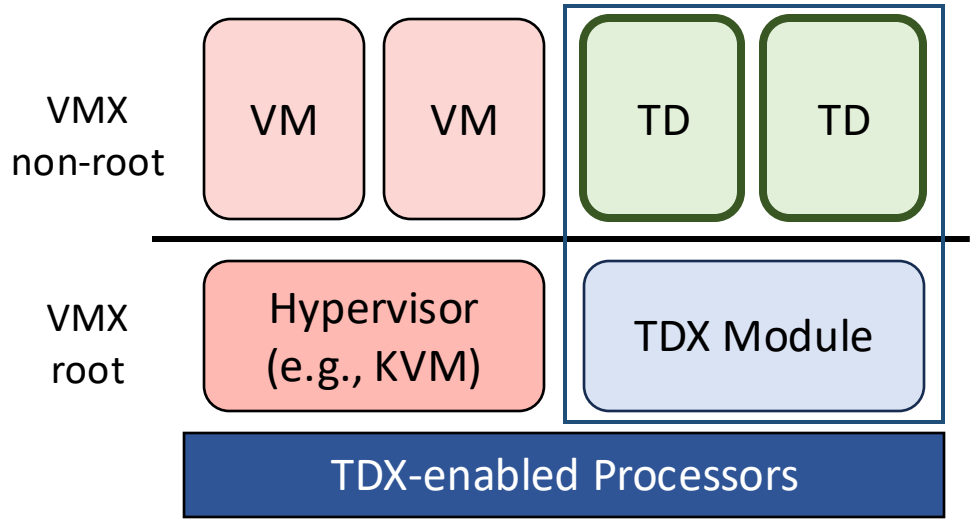
- Introduction of Intel TDX
  - Motivation of OpenTDX
- Demo of Working Example
- Use Cases of OpenTDX
- Technical Details
- Conclusion

# What is TDX?

- **Intel TDX (Trust Domain Extensions)**
  - Set of HW & SW extensions for **VM based confidential computing**
    - Protect user's workloads on possibly compromised cloud environments
    - Open new data applications by ***guaranteeing the security of data***
      - E.g., medical dataset analysis on public cloud → regulation compliance
    - Convince end users that their private data is securely processed
      - E.g., secure LLM serving system

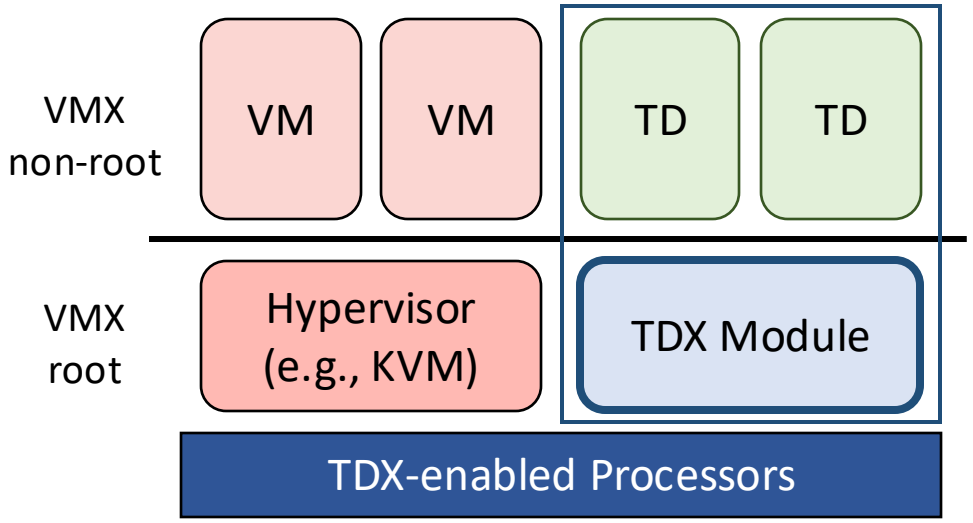
# What is TDX?

- **Intel TDX (Trust Domain Extensions)**
  - Set of HW & SW extensions for **VM based confidential computing**
  - **TDs** (i.e., confidential VM) protected on cloud environment



# What is TDX?

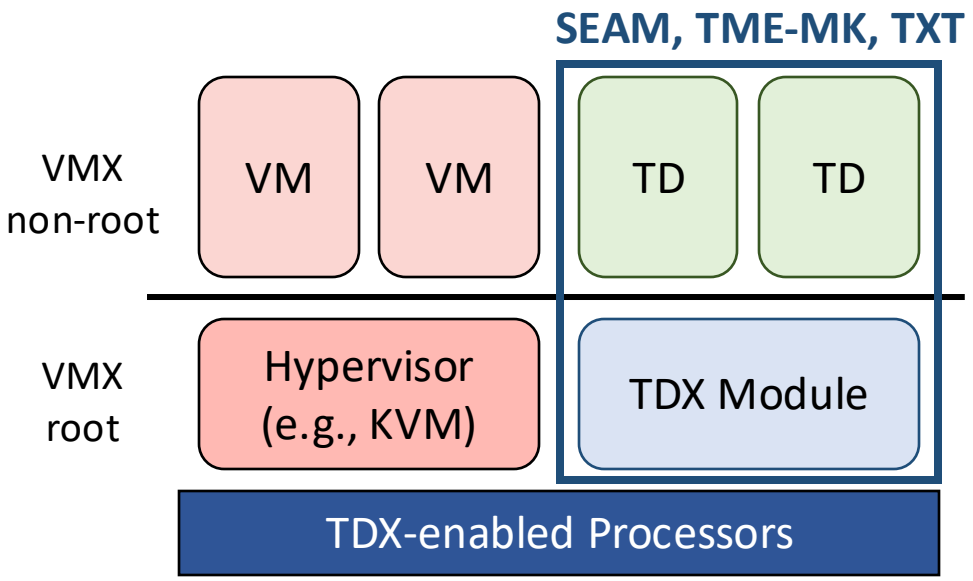
- **Intel TDX (Trust Domain Extensions)**
  - Set of HW & SW extensions for **VM based confidential computing**
  - **TDs** (i.e., confidential VM) protected on cloud environment
  - **TDX Module** (i.e., trusted thin hypervisor) protecting life-cycle of TDs



# What is TDX?

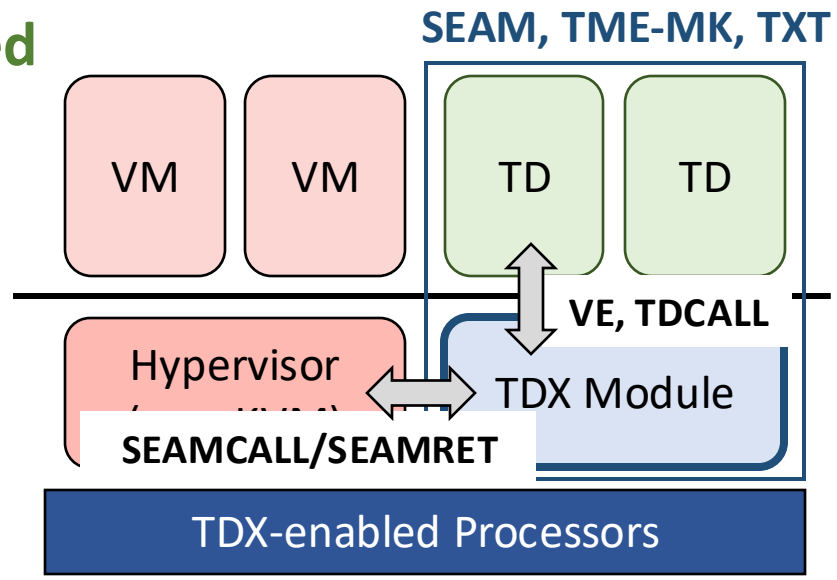
- **Intel TDX (Trust Domain Extensions)**
  - Set of HW & SW extensions for **VM based confidential computing**
  - **TDs** (i.e., confidential VM) protected on cloud environment
  - **TDX Module** (i.e., trusted thin hypervisor) protecting life-cycle of TDs

  - Hardware extensions
    - **SEAM** (SEcure Arbitration Mode)
      - For access control in CPU
    - **TME-MK** (Total Memory Encryption Multi-Key)
      - For encrypting data at memory
    - **TXT** (Trusted Execution Technology)
      - For root-of-trust



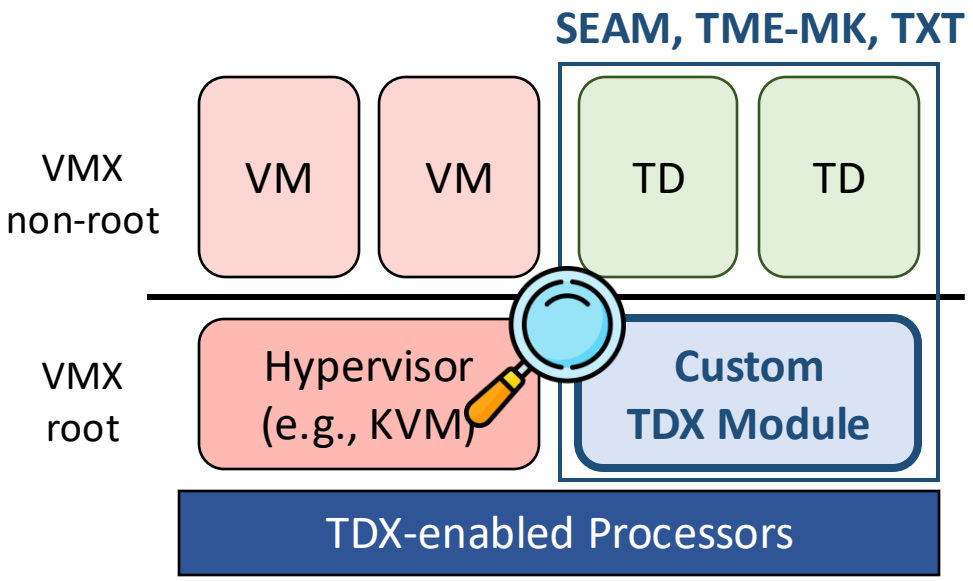
# What is TDX Module?

- **TDX Module**
  - Trusted thin hypervisor for constructing, operating, destructing TDs
    - Emulate CPU instructions, manage memories, etc.
  - Interacts with host hypervisor (i.e., **SEAMCALL/SEAMRET**), and TDs (i.e., **Virtualization Exception, TDCALL**)
  - **Only the TDX Module signed by Intel can be loaded**
    - **Root of Trust for TDs**
  - **Unable to customize TDX Module**
    - **Limited contributions from research community**
    - **Difficulty of implementing and debugging TDX SW stacks**



# Bridging the Gap: OpenTDX

- **OpenTDX**
  - We emulate a TDX machine (on a normal Intel processor) to run **custom TDX modules**
  - We extend **QEMU-KVM** to emulate a TDX machine
  - **Any one can develop and test their own TDX modules on OpenTDX**



# Demo: Working Example

# Project Description

- https://github.com/sslab-gatech/open-tdx

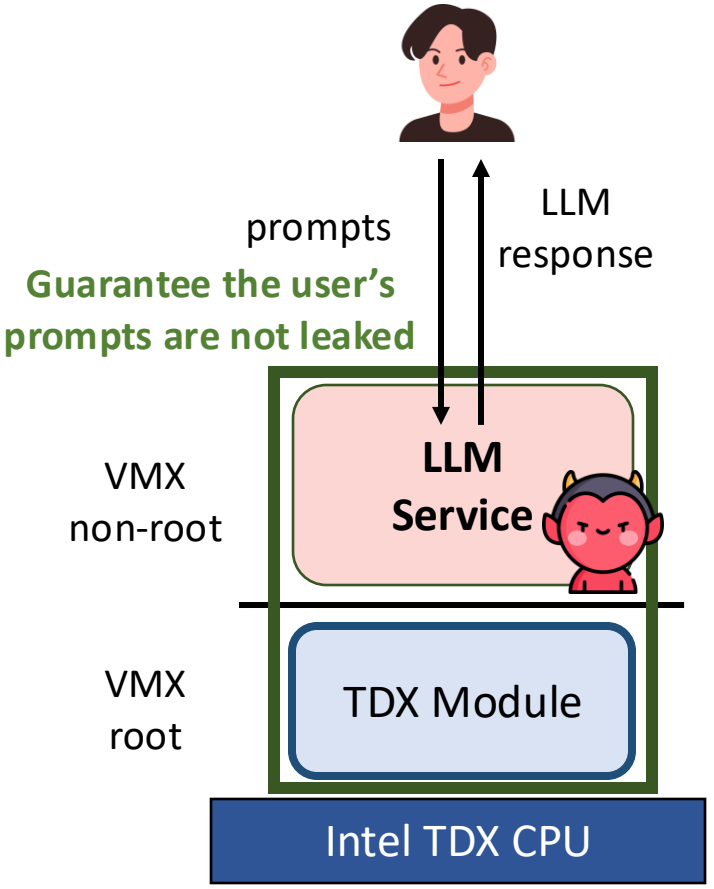
# Use Cases of OpenTDX

- Re-implementing TDX Module in RUST
- Secure & efficient shared memory between TDs
  - TDX Module can assign the same shared TME-MK key for the TDs
- Fuzzing TDX Module (i.e., white-box fuzzing)
  - Instrument TDX Module for sanitization, coverage guidance, etc.
  - Cornelius [1] can be used also, but some functionalities would be limited (e.g., VE injection)
- Implementing & debugging software stack for TDX
  - New host hypervisor
  - New TD kernel

[1] https://github.com/microsoft/Cornelius

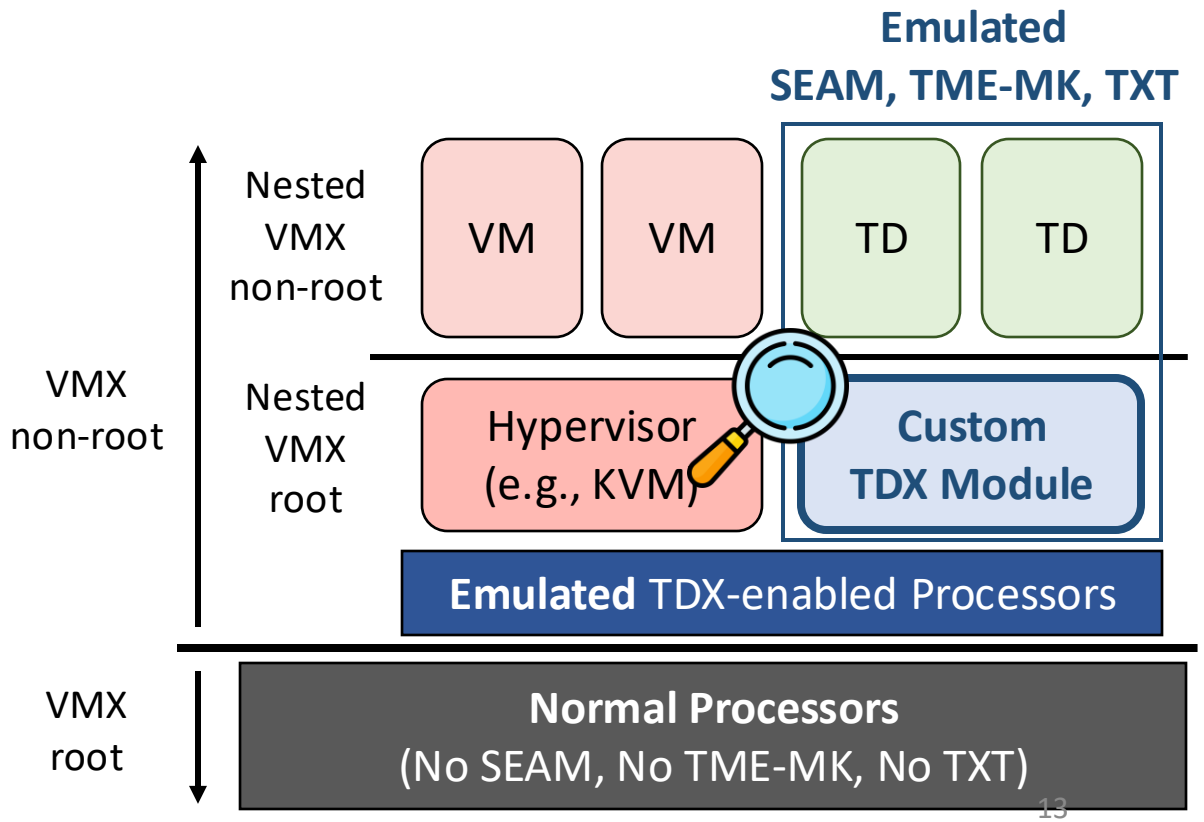
# Use Cases of OpenTDX

- New Design of Secure LLM Serving System using TDX Module
  - Motivation
    - LLM service may be compromised (or even malicious) due to its large code base
  - Goal
    - Protect the user's prompts even when the LLM service is compromised
  - Key Insight
    - TDX module can provide a stronger protection on the prompts thanks to its higher privilege
    - TDX module can be open-sourced and verified by end users



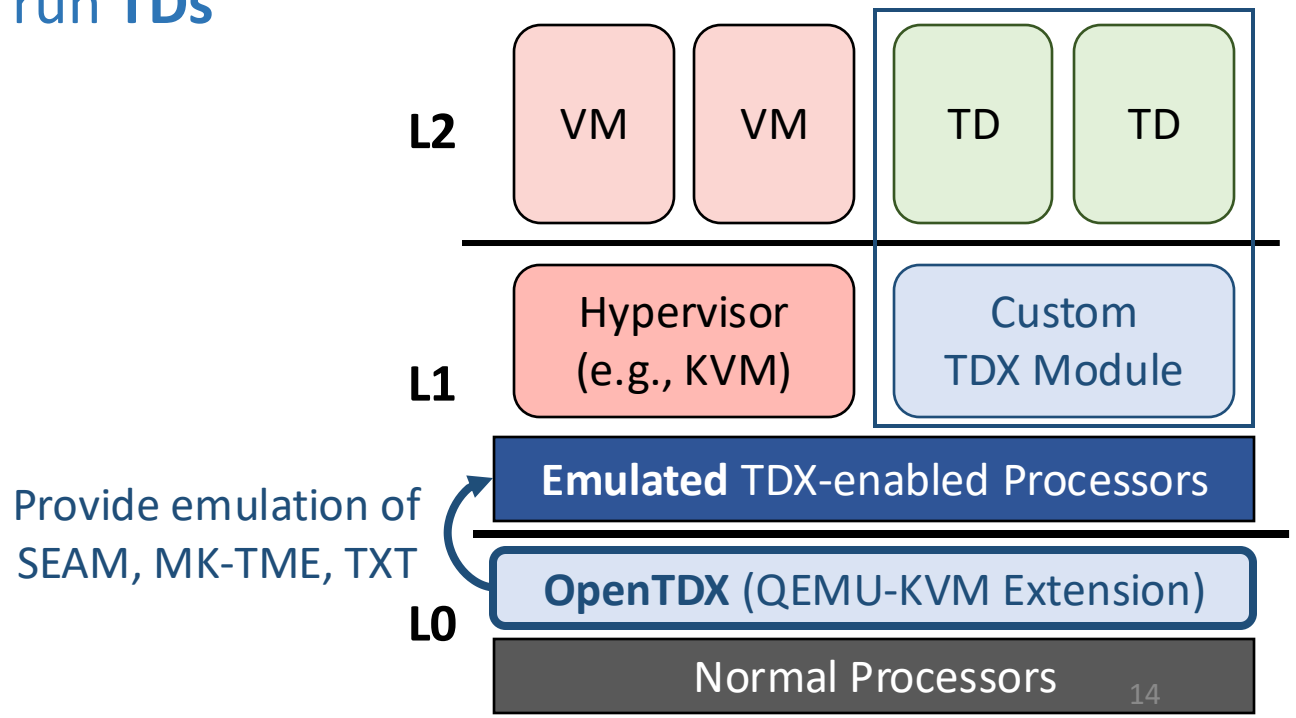
# Deep Dive into OpenTDX

- Components of OpenTDX
  - Nested virtualization
    - Host TDs as nested VMs
  - ISA emulation
    - TXT
    - TME-MK
    - SEAM
    - Other TDX features



# Nested Virtualization for Serving TDs

- Virtualization Levels
  - **L0:** Normal Intel CPU (i.e., no TDX support) → run **OpenTDX**
  - **L1:** Emulated TDX CPU → run **TDX Module**
  - **L2:** VMs on emulated TDX CPU → run **TDs**



# ISA Emulation: TXT

- Trusted Loading Procedure of TDX Module
  - ***NP-SEAMLDR → P-SEAMLDR → TDX Module***
  - Intel TXT guarantees only the ***NP-SEAMLDR*** signed by Intel is loaded
    - ***NP-SEAMLDR*** further guarantees only the correct ***TDX Module*** is loaded
  - ISA
    - GETSEC[ENTERACCS]
      - Given a signed NP-SEAMLDR binary,
        check signature, parse and load, then execute in a protected mode
    - GETSEC[EXITAC]
      - Exit from the protected mode

Instructions for loading *NP-SEAMLDR* in TDX enabled CPU

```
…
xor %eax, %eax
mov %eax, %ebx
getsec
mov %eax, -0x1c(%ebp)
…
```

# ISA Emulation: TXT

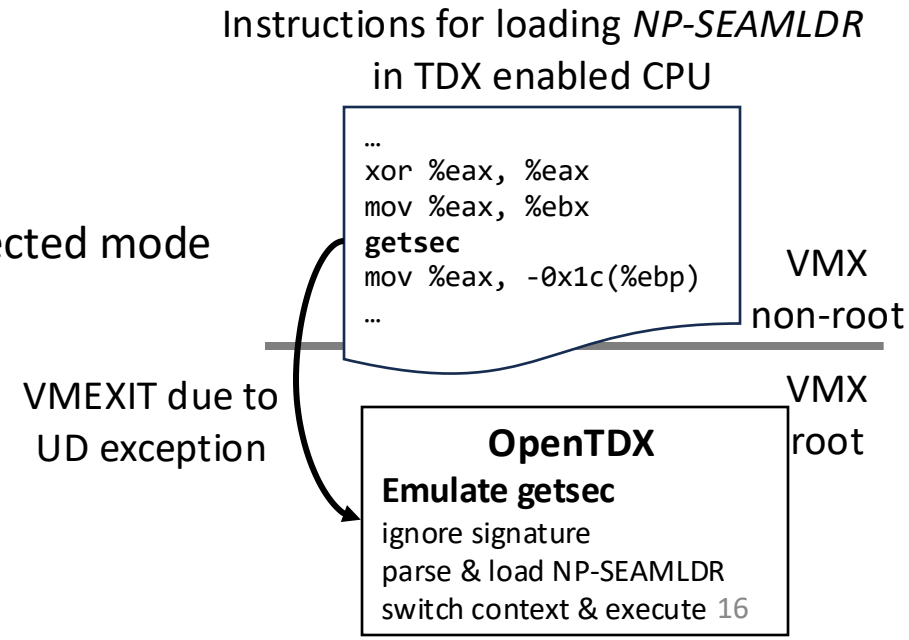
- Trusted Loading Procedure of TDX Module
  - ***NP-SEAMLDR → P-SEAMLDR → TDX Module***
  - Intel TXT guarantees only the ***NP-SEAMLDR*** ~~signed by Intel~~ is loaded
    - ***NP-SEAMLDR*** further guarantees only the correct ***TDX Module*** is loaded

**Removed signature verification**

- ISA

**Emulate them by catching *UD* exception**

  - GETSEC[ENTERACCS]
    - Given a signed NP-SEAMLDR binary,
      check signature, parse and load, then execute in a protected mode
  - GETSEC[EXITAC]
    - Exit from the protected mode



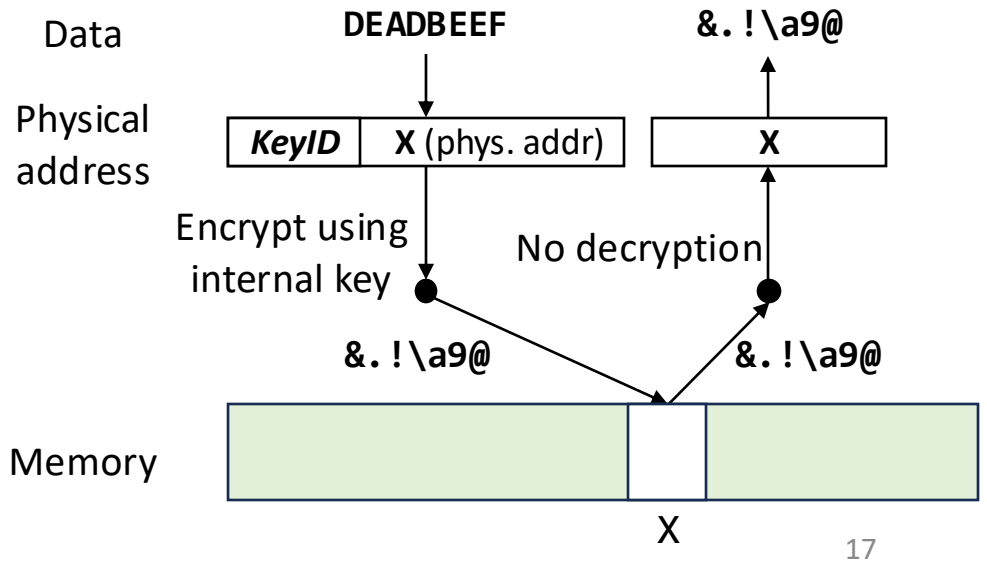
# ISA Emulation: TME-MK

- Memory encryption of TDX module and TDs
  - CPU repurposes topmost bits of physical address as ***KeyID***
    - Encrypts the outgoing data using ***internal key bound to KeyID***
    - Read with different (or empty) ***KeyID*** returns trash value
  - TDX module protects memory by programming ***KeyIDs***

  - ISA
    - Memory encryption based on KeyID

    - TME-MK MSRs
      - Configure TME-MK keyID space

    - PCONFIG
      - Program KeyIDs



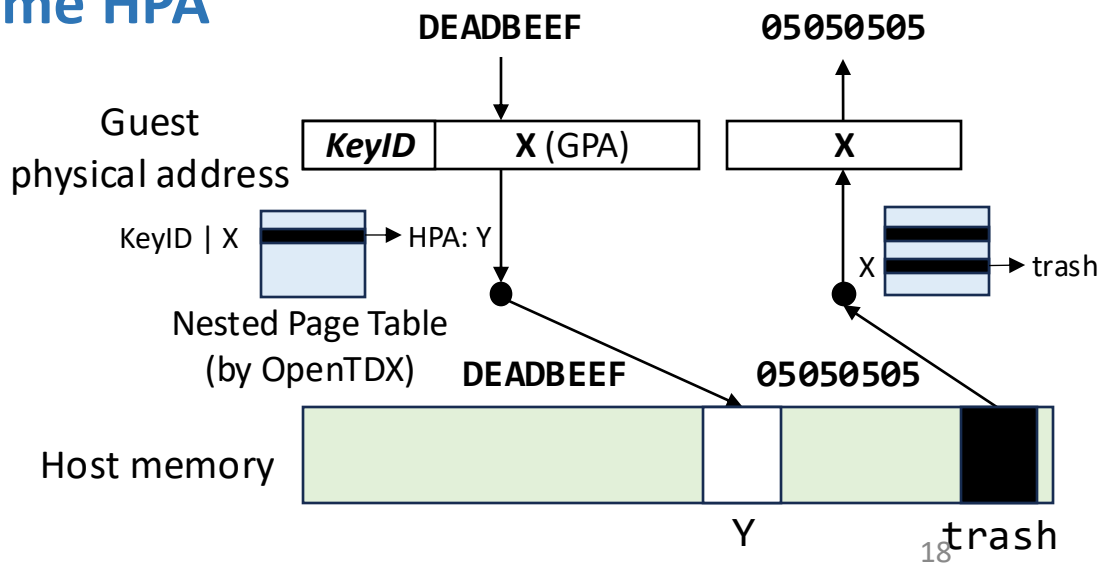
# ISA Emulation: TME-MK

- Memory encryption of TDX module and TDs
  - CPU repurposes topmost bits of physical address as ***KeyID***
    - ~~Encrypts the outgoing data using ***internal key bound to KeyID***~~
    - Read with different (or empty) ***KeyID*** returns trash value
  - TDX module protects memory by programming ***KeyIDs***
  - ISA
    - Memory encryption based on KeyID
    - TME-MK MSRs
      - Configure TME-MK keyID space
    - PCONFIG
      - Program KeyIDs

**Emulate MSRs & instruction**

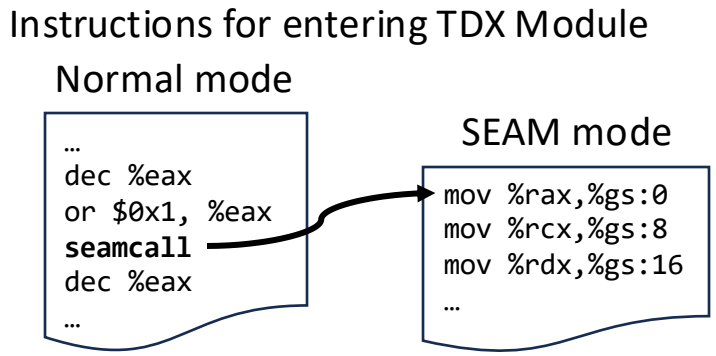
**Do not encrypt data**

**Manipulate nested page table to make (keyID | GPA) point to the same HPA**



# ISA Emulation: SEAM

- Privilege Separation of TDX Module vs. Normal Hypervisors
  - CPU enters privileged mode (i.e., SEAM) upon executing ***SEAMCALL***
    - ***SEAMCALL*** loads TDX module's context from ***SEAM VMCS***
    - Only SEAM mode SW can access TME-MK encrypted data
  - TDX Module in SEAM mode serves TDs as like normal hypervisors
    - Using VT-X features (e.g., VMCS, VMENTER, etc.)
  - ISA
    - Context switch using SEAM VMCS
    - Memory access control based on SEAM mode
      - Access permission to TME-MK encrypted memory
    - SEAM MSRs
      - For protecting SEAM memory regions



# ISA Emulation: SEAM

- Privilege Separation of TDX Module vs. Normal Hypervisors
  - CPU enters privileged mode (i.e., SEAM) upon executing ***SEAMCALL***
    - ***SEAMCALL*** loads TDX module's context from ***SEAM VMCS***
    - Only SEAM mode SW can access TME-MK encrypted data
  - TDX Module in SEAM mode serves TDs as like normal hypervisors
    - Using VT-X features (e.g., VMCS, VMENTER, etc.)
  - ISA
    - Context switch using SEAM VMCS
    - Memory access control based on SEAM mode
      - ~~Access permission to TME-MK encrypted memory~~
    - SEAM MSRs
      - For protecting SEAM memory regions

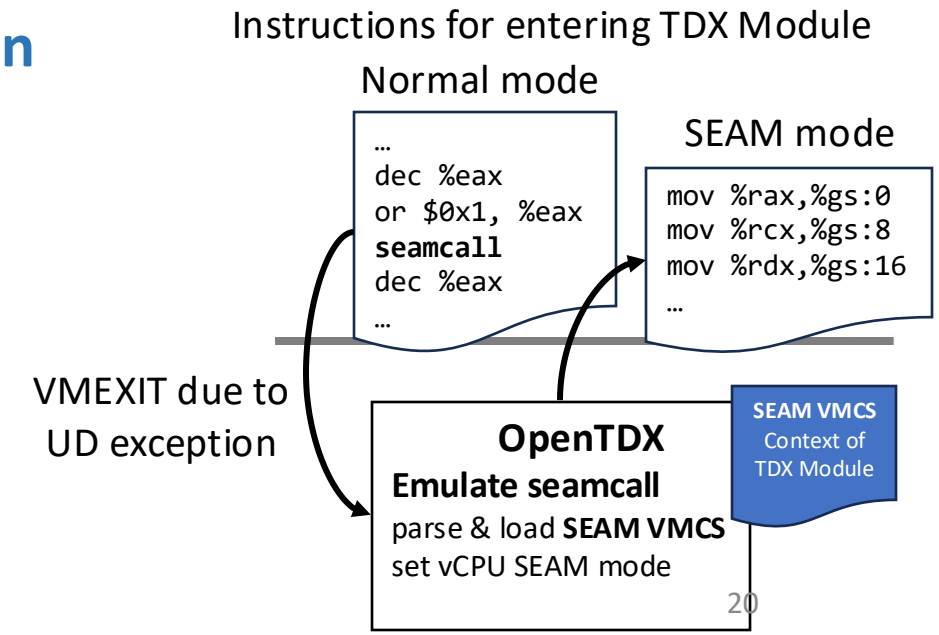
**Emulate SEAMCALL/SEAMRET**

**Use nested virtualization**

**Emulate VMCS structure**

**Currently not supported**

**Emulate MSRs**



# ISA Emulation: Other TDX Features

- GPA separation
  - Topmost bit = 0 in GPA → Normal EPT (encrypted using TD's KeyID)
  - Topmost bit = 1 in GPA → Shared EPT (non-encrypted)
  - Shared EPTP field added to VMCS

# ISA Emulation: Other TDX Features

- GPA separation
  - Topmost bit = 0 in GPA → Normal EPT (encrypted using TD's KeyID)
  - Topmost bit = 1 in GPA → Shared EPT (non-encrypted)
  - Shared EPTP field added to VMCS

**On nested EPT violation, walk normal/shared EPT depending on the topmost bit**

**Add field to nested VMCS**

# ISA Emulation: Other TDX Features

- GPA separation

**On nested EPT violation, walk normal/shared EPT depending on the topmost bit**

  - Topmost bit = 0 in GPA → Normal EPT (encrypted using TD's KeyID)
  - Topmost bit = 1 in GPA → Shared EPT (non-encrypted)
  - Shared EPTP field added to VMCS

**Add field to nested VMCS**

- Virtualization Exception
  - EPT violation & specific EPT entry setting
    → Inject virtualization exception in TD without VMEXIT

# ISA Emulation: Other TDX Features

- GPA separation

**On nested EPT violation, walk normal/shared EPT depending on the topmost bit**

  - Topmost bit = 0 in GPA → Normal EPT (encrypted using TD's KeyID)
  - Topmost bit = 1 in GPA → Shared EPT (non-encrypted)
  - Shared EPTP field added to VMCS

**Add field to nested VMCS**

**On nested EPT violation, check condition & inject**

- Virtualization Exception
  - EPT violation & specific EPT entry setting
    → Inject virtualization exception in TD without VMEXIT

# Conclusion

- OpenTDX emulates TDX machines on normal Intel CPUs
- Wish you enjoy playing with OpenTDX!
- https://github.com/sslab-gatech/open-tdx

# Thank you

For any question, please contact **jwhur19@gmail.com**