Virtual Firmware for Intel® Trust Domain Extensions

UEFI 2020 Virtual Plugfest
December 15, 2020
Jiewen Yao, Principal Engineer, Intel Corporation
Jiewen Yao

• Principal engineer in Intel Architecture, Graphics, and Software (IAGS).
• Firmware developer for over 15 years.
• Member of the UEFI Security sub team and the TCG PC Client sub working group.
• He is the architect of Intel® TDX Virtual Firmware.
Agenda

- Intel TDX
- TDVF
- TDShim
- Disk Encryption
- Summary
Intel TDX
Why Intel TDX?

Tranditional virtual machine (VM)/hypervisor
• Hypervisor has highest privilege.
• Hypervisor may tamper VM.

What can Intel Trust Domain Extensions (TDX) offer?
• Trust Domain (TD) can resist the attack from hypervisor.
• A TD can be used for confidential computing.
Intel® Trust Domain Extensions (TDX)

Intel TDX aware Virtual Machine Manager (VMM)

Legacy VMs

OS
Virtual BIOS

OS
Virtual BIOS

OS
Virtual BIOS

Trust Domain

TDX Enlightened OS
TD Virtual Firmware

Trust Domain

TDX Enlightened OS
TD Virtual Firmware

Trust Domain

TDX Enlightened OS
TD Virtual Firmware

Intel TDX Module

www.uefi.org
Intel TDX related components

Intel TDX CPU Hardware

Intel TDX Module
- Run in Secure Arbitration Mode (SEAM), protected by SEAM range register (SEAMRR)
- Provide SEAMCALL service to a VMM and TDCALL service to a TD.
- Manage the transition between the VMM and the TD.

SEAM Loader (SEAMLDN) Authenticated Code Module (ACM)
- Check the hardware configuration
- Load Intel TDX Module to protected memory

TD Quoting Enclave
- Support remote attestation for a TD

www.uefi.org
System Boot Flow

SEAMLDR ACM → VMM → Intel TDX module

TDX aware BIOS → SEAMCALL

TD

VMENTRY

VMEXIT

SEAMRET
TDVF
TD Virtual Firmware (TDVF)

Responsibility:
• Own 1\textsuperscript{st} instruction of a trust domain (TD) at reset vector
• Provide service to a TD operating system (TD-OS)
• Build chain-of-trust from Intel TDX Module to TD-OS

Implementation:
• Based upon EDK II Open Virtual Machine Firmware (OVMF)
• Simplified boot flow (no PEI phase)
Threat Model Difference

Do not trust any VMM input.

• Always validate the input before use.
• Always measure the input for remote attestation.
General Boot Flow – TDVF

**Protected Mode Entry**
- Reset Vector
  - Parse TdHob
  - Measure TdHob
  - Prepare InitMemory
  - Prepare DxeHob
  - Jump to DxeCore

**SEC**
- Volatile Only
  - Measure to RTMR
  - Accept Rest PrivMem
  - Share-PrivMem Switch

**DXE**
- Init MemoryMngm
  - Initialize CPU MP
  - Initialize UefiVar
  - Enable SecureBoot
  - Enable TrustedBoot
  - Prepare AcpiTable
  - Prepare DxeMemory
  - Init DMA Mngm

**APs also jump to Reset Vector**
- Enumerate VirtIo
- Start Console(i/o)
- Start Storage Dev
- Boot OS

**BDS**
- Similar to OVMF
  - Updated in TDVF
  - Unique in TDVF

www.uefi.org
TDVF Binary Layout

Guest Mem

Boot Code (BFV)

Configuration (CFV)

Temporary Memory

TD Hob

CFV

TDVF Descriptor

Base=0xFFD00000, Len=300000
Type = BFV, Attrib=EXTENDMR
RawOffset=100000, Size=300000

Base=0xFFFFC00000, Len=100000
Type = CFV, Attrib=0
RawOffset=0, Size=100000

Base=0x1010000, Len=F000
Type = TempMem, Attrib=0
RawOffset=0, Size=0

Base=0x1000000, Len=1000
Type = TD_HOB, Attrib=0
RawOffset=0, Size=0

Configuration FV

FV Header

VARIABLE_STORE_HEADER

VARIABLE_HEADER

PK

VARIABLE_HEADER

KEK

VARIABLE_HEADER

db

......

VARIABLE_HEADER

OtherVar

www.uefi.org
Launch State

Reset Vector:
• Protected mode reset vector (0xFFFFFFFF0)

General Purpose Register:
• **RBX**: Guest Physical Address Width (GPAW), 48 or 52
• **RCX/R8**: hold a pointer of TD Hob. TD Hob contains the TD information, such as memory information, MMIO/IO information, which is passed from VMM.
• **RSI**: Virtual CPU (VCPU) Index (0 ~ N-1)
Launch State

• Multi Processor Support
  – All CPUs jump to reset vector at same time.
  – VCPU 0 is selected as Bootstrap Processor (BSP).
  – VCPUs (1~N-1) are Application Processors (APs), parking and waiting to be waken up by BSP.

• TDCALL[TDG.VP.INFO]
  – R8: NUM_VCPUS
<table>
<thead>
<tr>
<th>TDCALL</th>
<th>Usage</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDG.VP.VMCALL</td>
<td>Invoke service from the VMM</td>
<td>(See next page)</td>
</tr>
<tr>
<td>TDG.VP.INFO</td>
<td>Get TD information</td>
<td>GPAW, NUM_CPUS</td>
</tr>
<tr>
<td>TDG.MR.RTMR.EXTEND</td>
<td>Extend to TD runtime measurement register (RTMR)</td>
<td>SHA384 hash</td>
</tr>
<tr>
<td>TDG.VP.VEINFO.GET</td>
<td>Get #VE information</td>
<td>Exit Reason, Instruction Information</td>
</tr>
<tr>
<td>TDG.MR.REPORT</td>
<td>Get TD_REPORT</td>
<td>Measurement of the TD, TD configuration, Intel TDX module, etc.</td>
</tr>
<tr>
<td>TDG.VP.CPUIDVE.SET</td>
<td>Control unconditional #VE on CPUID</td>
<td>Supervisor mode, user mode.</td>
</tr>
<tr>
<td>TDG.MEM.PAGE.ACCEPT</td>
<td>Accept a pending, private page.</td>
<td>Guest physical address/size</td>
</tr>
</tbody>
</table>
# TDCALL[TDG.VP.VMCALL]

<table>
<thead>
<tr>
<th>TDG.VP.VMCALL</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetTdVmCallInfo</td>
<td>Enumerate VMCALL capabilities</td>
</tr>
<tr>
<td>MapGPA</td>
<td>Request VMM to map a GPA range as private or shared memory</td>
</tr>
<tr>
<td>GetQuote</td>
<td>Request a Quote-Enclave to sign the TD_REPORT to a TD_QUOTE</td>
</tr>
<tr>
<td>ReportFatalError</td>
<td>Report fatal error in TD.</td>
</tr>
<tr>
<td>SetupEventNotifyInterrupt</td>
<td>Request VMM specify which interrupt vector to use as an event notify vector.</td>
</tr>
<tr>
<td>Instruction.CPUID</td>
<td>Request VMM to emulate CPUID instruction</td>
</tr>
<tr>
<td>#VE.RequestMMIO</td>
<td>Request VMM to emulate the MMIO access</td>
</tr>
<tr>
<td>Instruction.HLT</td>
<td>Request VMM to emulate HLT instruction</td>
</tr>
<tr>
<td>Instruction.IO</td>
<td>Request VMM to emulate IO instruction</td>
</tr>
<tr>
<td>Instruction.RDMSR</td>
<td>Request VMM to emulate RDMSR instruction</td>
</tr>
<tr>
<td>Instruction.WRMSR</td>
<td>Request VMM to emulate WRMSR instruction</td>
</tr>
<tr>
<td>Instruction.PCONFIG</td>
<td>Request VMM to emulate PCONFIG instruction</td>
</tr>
</tbody>
</table>
Memory Management

VMM

- ACCESS
- SEAMCALL [TDH.MEM.PAGE.ADD]
- ADD
- SEAMCALL [TDH.MEM.PAGE.AUG]
- AUG
- TDCALL [TDG.VP.VMCALL] <#VE.REQUESTMMIO> ACCESS

Trust Domain

- Shared Memory
  - TDCALL [TDG.MEM.PAGE.ACCEPT]

- Private Memory
  - TDCALL [TDG.MEM.PAGE.ACCEPT] <#MAPGPA>

- Unaccepted Memory
<table>
<thead>
<tr>
<th>Type</th>
<th>Usage</th>
<th>Setup</th>
<th>Guest Page Table</th>
<th>Access</th>
</tr>
</thead>
</table>
| Private Memory     | Default                                    | 1) **VMM**:SEAMCALL[TDH.MEM.PAGE.ADD]  
2) **VMM**:SEAMCALL[TDH.MEM.PAGE.AUG]  
**TD**:TDCALL[TDG.MEM.PAGE.ACCEPT] | S-bit cleared        | Direct Access   |
| Shared Memory      | Hypervisor communication buffer, Virtual device DMA buffer | Same as private memory                                                | S-bit set         | Direct Access |
| Unaccepted Memory  | Private memory, not accepted yet.          | **VMM**:SEAMCALL[TDH.MEM.PAGE.AUG]                                      | N/A               | N/A           |
| MMIO               | MMIO emulation                             | N/A                                                                   | N/A               | TDCALL[TDG.VP.VMC ALL]<#VE.REQUEST MMIO> |

www.uefi.org
## Memory State Transition

<table>
<thead>
<tr>
<th>Transition</th>
<th>Usage</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unaccepted -&gt; Private</td>
<td>Lazy memory init</td>
<td>TDCALL[TDG.MEM.PAGE.ACCEPT]</td>
</tr>
<tr>
<td>Private -&gt; Shared</td>
<td>Communication buffer setup</td>
<td>Set S-bit in page table. TDCALL[TDG.VP.VMCALL]&lt;MAPGPA&gt;</td>
</tr>
<tr>
<td>Shared -&gt; Private</td>
<td>Communication buffer reclaim</td>
<td>Clear S-bit in page table. TDCALL[TDG.VP.VMCALL]&lt;MAPGPA&gt; TDCALL[TDG.MEM.PAGE.ACCEPT]</td>
</tr>
</tbody>
</table>
Memory State Transition

- **Private Memory (TD Hob)**
- **Unaccepted Memory**

Init

Partial Memory Init

Full Memory Init

During Device Access

After Device Access
## UEFI/PI Memory Indicator

<table>
<thead>
<tr>
<th>Type</th>
<th>UEFI Memory Map</th>
<th>PI GCD</th>
<th>PI Hob</th>
<th>ACPI E820</th>
<th>ACPI ASL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Memory</td>
<td>Normal UEFI Memory Type</td>
<td>EfiGcdMemoryTypeSystemMemory</td>
<td>EFIRESOURCE_SYSTEM_MEMORY (EFIRESOURCE_ATTRIBUYTE_ENCRYPTED)</td>
<td>Normal Memory Range</td>
<td>N/A</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>Normal UEFI Memory Type</td>
<td>EfiGcdMemoryTypeSystemMemory</td>
<td>EFIRESOURCE_SYSTEM_MEMORY</td>
<td>Normal Memory Range</td>
<td>N/A</td>
</tr>
<tr>
<td>Unaccepted Memory</td>
<td>EfiUnaccepted Memory (*)</td>
<td>EfiGcdMemoryTypeUnaccepted (*)</td>
<td>EFIRESOURCE_SYSTEM_MEMORY (EFIRESOURCE_ATTRIBUYTE_UNACCEPTED) (*)</td>
<td>AddressRange Unaccepted (*)</td>
<td>N/A</td>
</tr>
<tr>
<td>MMIO</td>
<td>N/A</td>
<td>EfiGcdMemoryTypeMemoryMappedIO</td>
<td>EFIRESOURCE_MEMORY_MAPPED_IO</td>
<td>N/A</td>
<td>Memory</td>
</tr>
</tbody>
</table>
ACPI – MP Wakeup

• AP init in OS
  – All APs are reported via MADT ACPI table.
  – A new MPWK structure is defined to describe a 4KiB mailbox.
    • APs loop to check the vector in the mailbox.
    • OS fills the AP Wakeup vector, then AP jumps to the Wakeup vector.

```c
typedef struct {
    UINT8    Type;
    UINT8    Length;
    UINT16   MailBoxVersion;
    UINT32   Reserved2;
    UINT64   MailBoxAddress;
} ACPI_MADT_MPWK_STRUCT;

typedef struct {
    UINT16   Command;
    UINT16   Reserved;
    UINT32   ApicId;
    UINT64   WakeupVector;
    UINT8    OsReserved[SIZT_TO_2K];
    UINT8    FirmwareReserved[SIZE_TO_4K];
} ACPI_MPWK_MAIL_BOX;
```
TD Trusted Boot

- TDVMM
- Config FV
- TD Hob
- Load & Measure
- Verify OSLoader
- Disk & OSLoader
- Remote
- Attestation
- TDMR
- MRConfigID
- RTMR[0]
- RTMR[1]
## TD Trusted Boot (TDMR + 4 RTMR)

<table>
<thead>
<tr>
<th>PCR</th>
<th>Typical Usage</th>
<th>TD Register</th>
<th>TD Reg Index</th>
<th>Extended by</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Firmware Code</td>
<td>TDMR</td>
<td>0</td>
<td><strong>VMM</strong>:SEAMCALL[TDH.MR.EXTEND]</td>
<td>VF code (BFV, initial page table)</td>
</tr>
<tr>
<td>1</td>
<td>Firmware Data</td>
<td>RTMR [0]</td>
<td>1</td>
<td><strong>TDVF</strong>:TDCALL[TDG.MR.RTMR.EXTEND]</td>
<td>Dynamic Configuration Data (TD HOB, ACPI) Data from FW_CFG_IO_SELECTOR/ FW_CFG_IO_DATA</td>
</tr>
<tr>
<td>2</td>
<td>Option ROM code</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Option ROM data</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>OS loader code</td>
<td>RTMR [1]</td>
<td>2</td>
<td><strong>TDVF</strong>:TDCALL[TDG.MR.RTMR.EXTEND]</td>
<td>OS loader</td>
</tr>
<tr>
<td>5</td>
<td>Boot Configuration</td>
<td>RTMR [1]</td>
<td>2</td>
<td><strong>TDVF</strong>:TDCALL[TDG.MR.RTMR.EXTEND]</td>
<td>GPT, Boot Variable</td>
</tr>
<tr>
<td>6</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Secure Boot</td>
<td>RTMR [0]</td>
<td>1</td>
<td><strong>TDVF</strong>:TDCALL[TDG.MR.RTMR.EXTEND]</td>
<td>SecureBootConfig (CFV)</td>
</tr>
<tr>
<td></td>
<td>Configuration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TD Trusted Boot Interface

• EFI_TD_PROTOCOL
  – Similar to TCG: EFI_TCG2_PROTOCOL
  – Provide measurement services for OS loader and OS kernel

• TD EventLog ‘TDEL’ ACPI table
  – Similar to TCG: TPM2 ACPI table
  – Provide TD Event Log – similar to TCG2 event log
UEFI Secure Boot

• Image Verification is same
• UEFI Auth Variable
  — *Volatile only*
  — All keys be provisioned at TD build time
  — All keys are measured into RTMR
TDVF Design Principle

• Keep it Simple
  – Do remove unnecessary feature (e.g. PEI)
  – Don’t expose unrequired external interface (e.g. network)

• Apply best Security Practice
  – Do validate all input before use.
  – Don’t trust VMM (new threat model)

• Build Chain-of-Trust
  – Do measure all input before use.
  – Don’t use mutable non-volatile storage (causing MR change)
TD Shim
TD Shim - Motivation

• Non-UEFI OS
  – No UEFI services required.

• Container OS
  – Special OS interface. (no UEFI)

• A small service TD
  – Bare metal environment. (no UEFI)
# TD Shim – A tiny TDVF

<table>
<thead>
<tr>
<th></th>
<th>TDVF</th>
<th>TD Shim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Vector</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SEC (Initial Program loader - IPL)</td>
<td>IPL to boot a UEFI Core</td>
<td>IPL to boot a payload (UEFI, Container, Zephyr, etc)</td>
</tr>
<tr>
<td>UEFI Core</td>
<td>UEFI Services</td>
<td>NO</td>
</tr>
<tr>
<td>Device Driver</td>
<td>Virtio, PCI, etc</td>
<td>NO</td>
</tr>
<tr>
<td>ACPI Table</td>
<td>MADT / DSDT</td>
<td>Static ACPI table only. Or MP table extension</td>
</tr>
<tr>
<td>(MultiProcessor)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Map</td>
<td>UEFI Memory Map</td>
<td>E820 table</td>
</tr>
<tr>
<td>Trusted Boot</td>
<td>TD Measurement + TD Event Log (ACPI)</td>
<td>TD Measurement + TD Event Log Table</td>
</tr>
<tr>
<td>Secure Boot</td>
<td>UEFI Secure Boot</td>
<td>NO</td>
</tr>
</tbody>
</table>
TD Shim - Boot Flow

Reset Vector: 32KB (PageTable)
IPL: 25KB (Hash Alg)

UEFI Services
ACPI Table
Boot UEFI

MP Table
Dev Tree
Boot Container

UEFI Payload
Container Payload
Zephyr Payload

Boot Zephyr OS
Launch APP

Measure TD hob
Parse TD hob
Get memory info
Accept Memory
Locate/Load payload
Jump to payload

Park AP
Switch to long mode
Setup stack
Jump to IPL

TD Shim

Initial Program Loader - IPL (c)

Reset Vector (asm)

www.uefi.org
Disk Encryption
Storage Volume Key

• Today, the VMM/OS gets the “storage volume key” and decrypts the VM disk.
• In TDX, the VMM/OS is not trusted.
• The TD need get the “storage volume key” and decrypt the VM disk.
High Level Component Layout

- Customer
  - Key Server
    - Key
  - OS Data/APP (enc)
  - OS Loader/Kernel
  - Virtual Firmware
  - Key Agent
    - Key
  - CSP
  - VMM

- Decrypt the disk
  - Load data
- Get the Storage Volume Key
  - Put Key location to ACPI table
Secure Communication Channel

• Confidentiality/Integrity:
  – Transport Layer Security (TLS)

• Mutual Authentication
  – Client TD -> Key Server: Server certificate verification
  – Key Server -> Client TD: TD Attestation
TD Attestation

1. TDREPORT(Data)
2. QUOTE(TDREPORT(Data))
3. sign
4. Verify

www.intel.com
Intel Signing Key (Pub)

3rd Party

www.uefi.org
Disk Encryption Key Passing

Remote Key Server -> KeyAgent in TD
• TLS + mutual authentication

Key Agent in TD -> OS loader/kernel in TD
• Storage Volume Key Location ‘SVKL’ ACPI Table

Key Agent Location (Use case specific)
• TDVF – PROs: A common OS
• OS – PROs: Keep TDVF simple
Summary
Summary

Intel® TDX
- Supports memory and CPU state confidentiality and integrity
- Supports measurement and remote attestation

TD Virtual Firmware (TDVF)
- TDVF: Build the chain of trust & Launch a TD-OS
- TDShim: A tiny TD to boot a payload.

Disk Encryption
- KeyAgent: Get disk encryption key from remote key after TD attestation
- OsLoader/Kernel: Get key from KeyAgent and decrypt the disk
Reference

Intel® TDX Specification and Whitepaper
- Intel® Trust Domain Extensions (Intel® TDX)
- Intel® TDX Virtual Firmware Design Guide
- Intel® TDX Guest-Hypervisor Communication Interface

TDVF Pre-Production Code
- TdvfPkg at TianoCore (edk2-staging)
- TdShimPkg POC
Questions?
More Questions?

Following today’s webinar, join the live, interactive WebEx Q&A for the opportunity to chat with the presenter

Visit this link to attend: https://bit.ly/3oW5SdD
Meeting number (access code): 126 544 0541
Meeting password: q2TPMRqMw36 (72876776 from phones and video systems)
Thanks for attending the UEFI 2020 Virtual Plugfest

For more information on UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by

intel®