# Benchmark Presentation: Verifying Intel TDX Module

Dirk Beyer[1], **Po-Chun Chien**[1], Nian-Ze Lee[2,1], and Thomas Lemberger[1]

[1]LMU Munich · [2]National Taiwan University

SV-COMP 2025 @ Hamilton, Canada

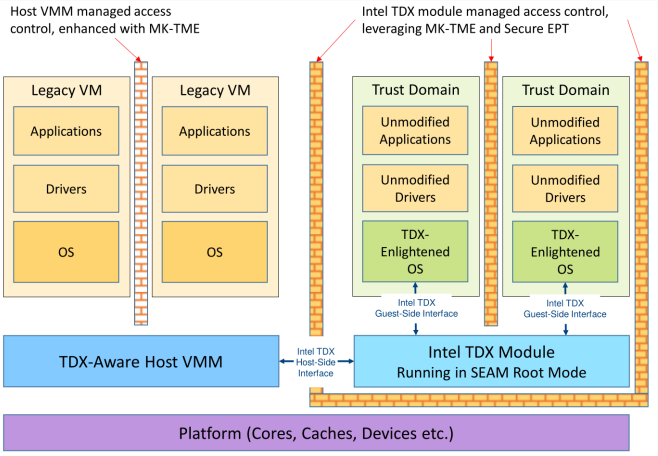


This work is supported by a research gift from Intel

# Intel TDX: Overview

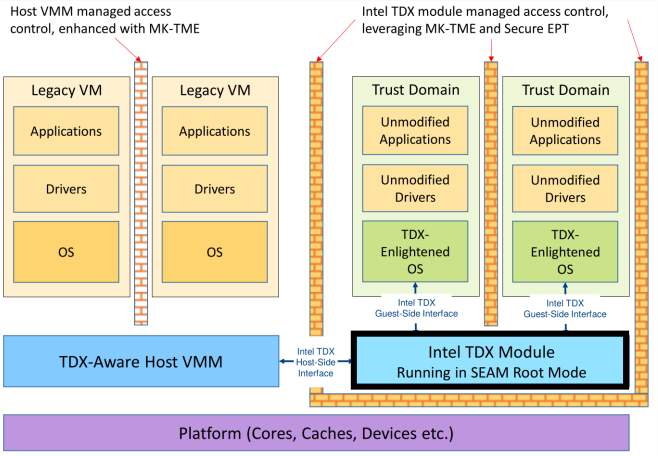
- Intel Trust Domain Extensions (TDX) [1]
  - Protect guest VMs against host VM manager (VMM) or OS
  - Encrypted memory and page tables, secure address translation, etc.
- Similar techniques:
  - ARM Confidential Compute Architecture [2, 3, 4]
  - AMD Infinity Guard [5]

# Intel TDX: Overview




Source: Fig. 2.1 in Intel TDX Module v1.5 Base Architecture Spec. [1]

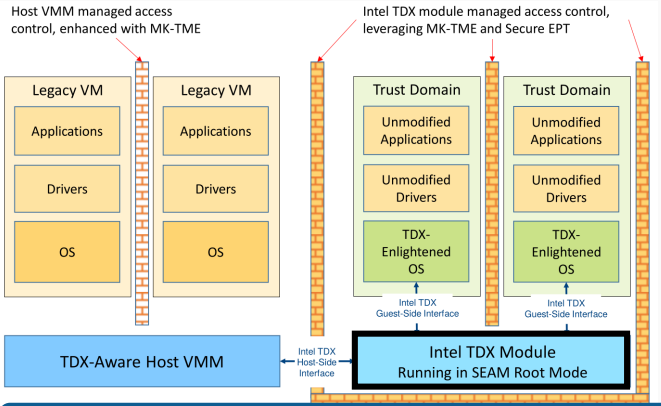
# Intel TDX: Overview




Source: Fig. 2.1 in Intel TDX Module v1.5 Base Architecture Spec. [1]

- Host: VMM
- Guest: TD
- Communicate via application binary interfaces (ABIs)

# Intel TDX: Overview




- Host: VMM
- Guest: TD
- Communicate via application binary interfaces (ABIs)

Goal: verify ABIs of TDX module (implemented as C code plus assembly), assuming VMM and TDs can call any ABI with arbitrary inputs

# Intel TDX: ABIs

- `TDH.*`: Host-side interface functions
- `TDG.*`: Guest-side interface functions
- Examples:
  - `TDH.MNG.CREATE`: Create a new guest TD and its root page
  - `TDH.VP.ENTER`: Enter TDX non-root operation
  - `TDG.MEM.PAGE.ACCEPT`: Accept a pending private page into the TD
  - `TDG.VM.WR`: Write a TD-scope metadata field
- Source: `https://github.com/intel/tdx-module`

# Intel TD

- TDH
- TDG
- Exa
  - 
  - 
  - 
  - 
- Sou



### 5.3.36. TDH.MNG.CREATE Leaf

Create a new guest TD and its TDR root page.

**Table 5.145: TDH.MNG.CREATE Input Operands Definition**

| Operand | Description | | |
|---|---|---|---|
| RAX | SEAMCALL instruction leaf number and version, see 5.3.1 | | |
| | **Bits** | **Field** | **Description** |
| | 15:0 | Leaf Number | Selects the SEAMCALL interface function |
| | 23:16 | Version Number | Selects the SEAMCALL interface function version<br>Must be 0 |
| | 63:24 | Reserved | Must be 0 |
| RCX | The physical address of a page where TDR will be created (HKID bits must be 0) | | |
| RDX | **Bits** | **Name** | **Description** |
| | 15:0 | HKID | The TD's ephemeral private HKID |
| | 63:16 | Reserved | Reserved: must be 0 |

**Table 5.146: TDH.MNG.CREATE Output Operands Definition**

| Operand | Description |
|---|---|
| RAX | SEAMCALL instruction return code – see 5.3.1 |
| Other | Unmodified |

# Defining Verification Tasks

```yaml
- name: tdh_mng_create__requirement__expected
  target:
    filename: formal/harness/tdh_mng_create_harness.c
    method: tdh_mng_create__valid_entry
  before_target:
    - filename: formal/src/initialization.c
      method: init_tdx_general
    - filename: formal/src/initialization.c
      method: init_vmm_dispatcher
    - filename: formal/harness/tdh_mng_create_harness.c
      method: tdh_mng_create__common_precond
  after_target:
    - filename: formal/harness/tdh_mng_create_harness.c
      method: tdh_mng_create__common_postcond
  properties:
    - property_file: unreach-call.prp
      expected_verdict: true
```

# Verification Harnesses I

- Initialize global (shadow) data

```
void init_tdx_general() {
  // Initialize global variables with non-deterministic values
  TDXFV_NONDET_struct_tdx_module_local_t(&local_data_fv);
  TDXFV_NONDET_struct_tdx_module_global_t(&global_data_fv);
  TDXFV_NONDET_struct_sysinfo_table_t(&sysinfo_table_fv);
  // .. snip ..
}
void init_vmm_dispatcher() {
  // .. snip ..
  shadow_td_regs_precall= local_data->td_regs;
  shadow_vmm_regs_precall = local_data->vmm_regs;
  shadow_guest_gpr_state_precall = local_data->vp_ctx.tdvps->guest_state.gpr_state;
  // .. snip ..
}
```

# Verification Harnesses II

- Initialize global (shadow) data
- Mock access to externally defined data

```
_STATIC_INLINE_ tdx_module_local_t* get_local_data(void) {
#ifdef TDXFV_NO_ASM
  return &local_data_fv;
#else
  uint64_t local_data_addr;
  _ASM_("movq %%gs:%c[local_data], %0\n\t"
        : "=r"(local_data_addr)
        : [local_data] "i"(
            offsetof(tdx_module_local_t, local_data_fast_ref_ptr)));
  return (tdx_module_local_t*)local_data_addr;
#endif
}
```

# Verification Harnesses III

- Initialize global (shadow) data
- Mock access to externally defined data
- Check postconditions

```
void tdh_mng_create__common_postcond() {
  tdx_module_local_t* tdx_local_data_ptr = get_local_data();

  TDXFV_ASSERT(tdx_local_data_ptr->td_regs.rax == shadow_td_regs_precall.rax);
  TDXFV_ASSERT(tdx_local_data_ptr->td_regs.rbx == shadow_td_regs_precall.rbx);
  TDXFV_ASSERT(tdx_local_data_ptr->td_regs.rcx == shadow_td_regs_precall.rcx);
  TDXFV_ASSERT(tdx_local_data_ptr->td_regs.rdx == shadow_td_regs_precall.rdx);
  // .. snip ..
}
```

# Challenges in Verification

- Handled through verification harnesses:
  - Replacing inline assembly with shadow C code
  - Mocking externally-defined variables
- Handled through HarnessForge:
  - Assembling single-file verification tasks
  - Slicing off code irrelevant to verification tasks
- Open challenges:
  - Bit-precise modeling of memory layouts (bit-fields, unions)
  - Modelling of huge complex types

# Modelling Complex Types: Nondet Elements ("havoc_object")

- Initialize each field of a complex type using dedicated initialization functions

```
void __VERIFIER_nondet_struct_tdvps_s(struct tdvps_s *dest) {
  __VERIFIER_nondet_struct_tdvps_ve_info_s(&((*dest).ve_info));
  __VERIFIER_nondet_array_1D_unsigned_char(&((*dest).reserved_0), 128);
  __VERIFIER_nondet_struct_tdvps_management_s(&((*dest).management));
  __VERIFIER_nondet_array_1D_unsigned_long_long(&((*dest).last_epf_gpa_list), 32);
  __VERIFIER_nondet_array_1D_unsigned_char(&((*dest).reserved_1), 256);
  __VERIFIER_nondet_array_1D_union_cpuid_control_s(&((*dest).cpuid_control), 128);
  __VERIFIER_nondet_struct_tdvps_guest_state_s(&((*dest).guest_state));
  // .. snip more field initializations ..
}
void __VERIFIER_nondet_array_1D_unsigned_char(unsigned char (*dest)[], int dim0) {
  for (int i = 0; i < dim0; i++) {
    (*dest)[i] = __VERIFIER_nondet_uchar();
  }
}
// .. snip more nondet functions ..
```

# Modelling Complex Types: Nondet Memory ("havoc_memory")

- Initialize each complex type byte by byte

```
static inline void TDXFV_NONDET_struct_tdvps_t(tdvps_t* dest) {
  TDXFV_NONDET_custom_type(dest, sizeof(tdvps_t));
}
void TDXFV_NONDET_custom_type(void* base, unsigned int size) {
  for (int i = 0; i < size; i++) {
    *((char*)base + i) = TDXFV_NONDET_uint8t();
  }
}
```

# Modelling Complex Types: Verifier-Specific Functions

```
static inline void TDXFV_NONDET_struct_tdvps_t(tdvps_t* dest) {
    __CPROVER_havoc_object(dest);
}
```

- CBMC delivered more results on tasks using its built-in initialization.
- Custom annotations enable verifiers to reason about complex types as they see fit.

# Contributions

- 290 tasks from 16 host-side ABIs (`TDH`) and 5 guest-side ABIs (`TDG`)
  - both element-wise initialization and nondet memory allocation
- 🔧 HarnessForge:
  - Assembles single-file verification tasks from real-world C projects
  - Slices off code irrelevant to verification tasks
- Next steps:
  - Experiment with effect of different initialization strategies
  - More tooling to support harness generation (e.g., harness-specific linter)
  - Moving towards `__VERIFIER_nondet_object`

# References i


[1] Intel Trust Domain Extensions, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html, accessed: 2024-05-01

[2] Arm Confidential Compute Architecture, https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, accessed: 2024-09-02

[3] Li, X., Li, X., Dall, C., Gu, R., Nieh, J., Sait, Y., Stockwell, G.: Design and verification of the Arm confidential compute architecture. In: Proc. OSDI. pp. 465–484. USENIX Association (2022), https://www.usenix.org/system/files/osdi22-li.pdf

[4] Fox, A.C.J., Stockwell, G., Xiong, S., Becker, H., Mulligan, D.P., Petri, G., Chong, N.: A verification methodology for the Arm confidential computing architecture: From a secure specification to safe implementations. Proc. ACM Program. Lang. **7**(OOPSLA1), 376–405 (2023). doi:10.1145/3586040

# References ii


[5] AMD Infinity Guard, `https://www.amd.com/en/products/processors/server/epyc/infinity-guard.html`, accessed: 2024-09-02

[6] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). doi:10.1007/978-3-642-22110-1_16

[7] Clarke, E.M., Kröning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proc. TACAS. pp. 168–176. LNCS 2988, Springer (2004). doi:10.1007/978-3-540-24730-2_15