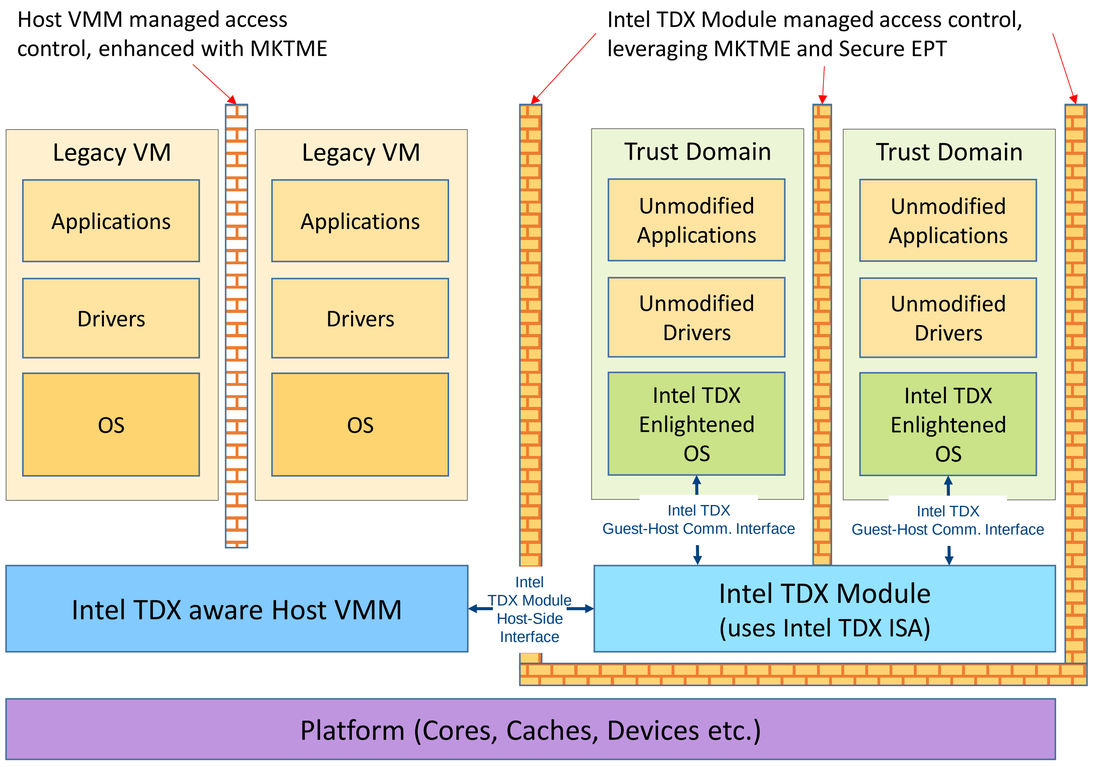
A Case Study in Firmware Verification
Applying Formal Methods to Intel® TDX Module

Dirk Beyer[1], Po-Chun Chien[1], Bo-Yuan Huang[2],
Nian-Ze Lee[3,1], and Thomas Lemberger[1]
[1]LMU Munich [2]Intel INT31 [3]National Taiwan University

Intel Trust Domain Extensions



Source: Fig. 2-1 in Intel TDX Module v1.5 Base Arch. Spec. [1]

Example ABI Specification

Excerpted specification of `TDH.MNG.KEY.CONFIG`

- **Input Operands**
  RAX: SEAMCALL instruction leaf number and version
  RCX: The physical address of a TDR page (HKID bits must be 0)
- **Output Operands**
  RAX: SEAMCALL instruction return code
  Other registers: Unmodified
- **State-Operand Information**
  TDR page: Explicitly accessed, read/write permission, 4KB-alignment
  Key Encryption Tables: Implicitly accessed
- **Completion-Status Codes**
  `TDX_OPERAND_INVALID`: An input operand of the ABI is invalid
  `TDX_LIFECYCLE_STATE_INCORRECT`: In an incorrect lifecycle state
  `TDX_KEY_GENERATION_FAILED`: Failed to generate a random key

Challenges for C Verifiers

- Extensive use of nested type punning (`union` types)
  – Many tools fail already at frontend parsing
- Initialization and preconditions for complex structures
  – Native havocking primitives greatly improves tool effectiveness
  – Constraints over array elements need standardized annotations (e.g., ∀ quantifier) for better effectiveness in automated tools.
- Handling of compiler attributes for memory-layout control
  – E.g., `aligned(n)` and `__packed__`
- Handling of inline assembly code
- Need for standardized annotation scheme for standard library functions (e.g., `memcpy`) and system calls
- Limited effectiveness for negative-space safety properties
  – Current tools lack semantic slicing of unreachable core logic

Contributions

- Adapted code-level methodology for firmware
- Developed proof harnesses and assembled verification tasks for TDX Module v1.5.05 using HarnessForge [2]
- Released harnesses and tasks publicly [3]

More Information

SoSy-Lab Software Systems

Proof Harness

- Initialize global data and assume preconditions
  – Havoc memory region byte by byte
  – Havoc object field by field
  – Use verifier builtins (e.g., `__CPROVER_havoc_object` in CBMC)
- Mock access to externally defined data and model inline assembly
- Assert postconditions

```
1  // proof harness components
2  void hmkc__setup() {
3    fv_setup_module_state();
4    fv_setup_tdr();
5    fv_setup_tdcs();
6  }
7  void hmkc__teardown() {
8    fv_teardown_tdcs();
9    fv_teardown_tdr();
10   fv_teardown_module_state();
11 }
12 void hmkc__invalid_input_rcx__precond() {
13   ASSUME(
14     !is_valid_hmkc_input_rcx() &&
15     is_valid_hmkc_state_metadata() &&
16     is_valid_hmkc_state_lifecycle());
17 }
18 void hmkc__invalid_input_rcx__postcond() {
19   ASSERT(
20     get_local_data()->vmm_regs.rax ==
21     api_error_with_operand_id(
22       TDX_OPERAND_INVALID,OPERAND_ID_RCX));
23 }
24 // auto-generated task entry point
25 void main() {
26   hmkc__setup();
27   hmkc__invalid_input_rcx__precond();
28   hmkc__function_call();
29   hmkc__invalid_input_rcx__postcond();
30   hmkc__teardown();
31 }
```

Verification Results (#solved)

	Interface function	#tasks	Mem. havoc.	Obj. havoc.	Tool-spec. havoc.
Guest-side TDG	MR.REPORT	10	0	0	5
	SERVTD.WR	10	0	0	7
	SYS.RD	10	0	0	8
	VM.WR	17	0	0	14
	VP.ENTER	18	0	0	12
	VP.VMCALL	10	0	0	8
Host-side TDH	EXPORT.RESTORE	10	3	0	7
	IMPORT.ABORT	9	2	0	7
	MNG.ADDCX	33	5	2	11
	MNG.CREATE	18	3	0	17
	MNG.INIT	33	5	0	11
	MNG.KEY.CONFIG	18	4	0	16
	MNG.KEY.FREEID	17	2	0	16
	MNG.VPFLUSHDONE	21	4	0	19
	MR.FINALIZE	29	3	0	7
	PHYMEM.PAGE.RECLAIM	25	0	0	5
	SYS.CONFIG	29	0	0	8
	SYS.INIT	17	0	0	13
	SYS.KEY.CONFIG	13	10	0	13
	SYS.SHUTDOWN	17	0	0	13
	SYS.UPDATE	25	0	0	19
	VP.ENTER	29	0	0	9
	Overall	418	41	2	245

References

[1] Intel Trust Domain Extensions, https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html
[2] Beyer, D., Chien, P.C., Huang, B.Y., Lee, N.Z., Lemberger, T.: Harnessforge: Automated extraction of verification tasks from industry-scale software projects (2026), (manuscript available upon request)
[3] Beyer, D., Chien, P.C., Huang, B., Lee, N.Z., Lemberger, T.: The Intel TDX Module benchmark set. Zenodo (2025)