TeenyTinyTrustyCore (3TC)

Members: Raha Moradishahmir and Raheel Afsharmazayejani

Advisors: Dr Benjamin Tan

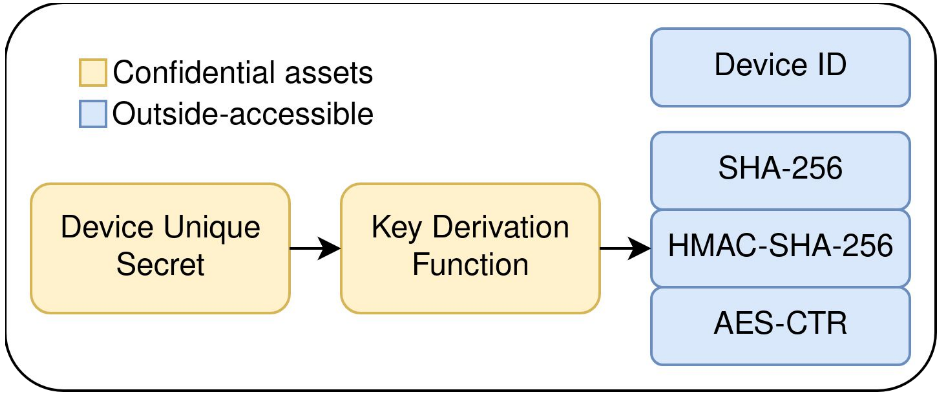
Department of Electrical and Software Engineering,
Schulich School of Engineering,
University of Calgary

Cognichip Hackathon, February 2026

UNIVERSITY OF
CALGARY

Problem statement & motivation (1)

- TeenyTinyTrustyCode: a tiny Hardware Root of Trust



Code available at https://github.com/raha96/teenytinytrustycore

Problem statement & motivation (2)

- Hardware Root of Trust (HoT)
  - TeenyTinyTrustyCore (3TC)
    - Minimal, yet fully functional
    - Verified for functionality and security
- Structure
  - Unique ID
  - Device Secret + Key derivation
  - Cryptographic functions
- Feasibility Study

Approach #1: LLM at the Helm

	Planned	Approach / Tool
1	Design Specification	Interactive LLM-based
2	Prompt Engineering	Verification Planning and Execution
3	Simulation	Verilator
4	UVM	Not Applicable
5	Syntactic and Semantic Check	CogniChip
6	Functional Verification	Dynamic Verification
7	Security Verification	Dynamic Verification
8	Functional / Security Sign-off	Simulation-based

UNIVERSITY OF CALGARY

Prompts

The original prompt:

We are implementing a minimal hardware root of trust. The system must implement SHA-256, HMAC-SHA-256 and AES-CTR algorithms. The design should include a PUF module that is used to generate the Device Unique Secret (DUS). It should also include a Key Derivation Function (KDF) that generates the keys needed for the abovementioned functions, so that the unique secret itself never leaks. There should also be another PUF that is used to generate an immutable Device ID. The design should include an AXI4 interface for communicating with the outside world. The three cryptographic modules and the Device ID should be accessible through the bus, but the DUS and the KDF must be isolated. Generate a list of the modules, then implement each one.

Prompts

Verification Planning Prompt:

```
We have implemented a minimal hardware root of trust that:
* Implements SHA-256, HMAC-SHA-256 and AES-CTR algorithms
* Includes a PUF module that is used to generate the Device Unique Secret (DUS)
* Includes a Key Derivation Function (KDF) that generates the keys needed for the
abovementioned functions, so that the unique secret itself never leaks
* Contains an immutable Device ID generated using another PUF
* Uses an AXI4 interface for communicating with the outside world. The three cryptographic
modules and the Device ID should be accessible through the bus, but the DUS and the KDF must
be isolated.
The code is available in the working directory. Write a verification plan that covers
functionality and security.
```

General Approach

- In this attempt, CogniChip was instructed to implement the system and chalk up a verification plan for it.
- Initially, CogniChip chose UVM as the verification method of choice. However, as no compatible tool was available on the system, it offered to switch to run similar tests using a pure testbench-based approach.
- CogniChip was instructed to use a local copy of Verilator.
- Over several iterations, CogniChip debugged the testbench itself, then started to detect bugs in the design and was instructed to fix them.

Results and Observations

- The design was not fully debugged before we hit the token limit at the last day of the competition.
- However, numerous bugs were fixed, and partial functionality was achieved.
- In the end, the design did not meet the requirements for functional or security closure. Interestingly enough, the LLM insisted that such goals had been met, and some modules were “ready for production” - even though the system as a whole failed basic functional tests and the said modules were not fully tested.

UNIVERSITY OF CALGARY

Approach #1: LLM at the Helm

	Planned	Approach / Tool	CogniChip Compatible	Done/Achieved
1	Design Specification	Interactive LLM-based	Yes	Yes
2	Prompt Engineering	Verification Planning and Execution	Yes	Yes
3	Simulation	Verilator	Yes	Yes
4	UVM	Not Applicable	No	Attempted
5	Syntactic and Semantic Check	CogniChip	Yes	Yes
6	Functional Verification	Dynamic Verification	Yes	Partial
7	Security Verification	Dynamic Verification	Yes	Partial*
8	Functional / Security Sign-off	Simulation-based	Yes	Partial**

* Some assertions were generated based on the desired security properties, but were never checked
** The tool insisted that the design is production-ready, but major bugs remained unrectified

UNIVERSITY OF CALGARY

Approach #2: UVM-based

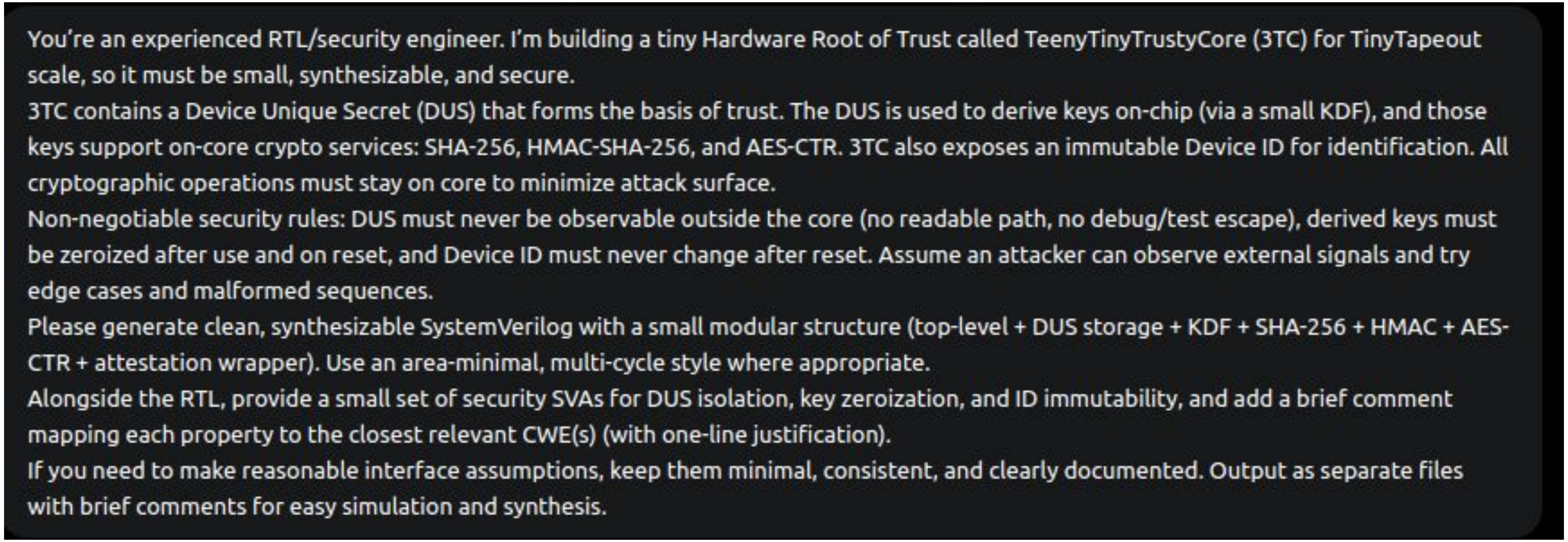
	Planned	Approach
1	Design Specification	Interactive LLM-based
2	Prompt Engineering	UVM-based Verification Planning
3	Simulation	Icarus Verilog
4	UVM	Questa
5	Syntactic and Semantic Check	CogniChip
6	Functional Verification	Dynamic Verification
7	Security Verification	Dynamic + Formal Verification
8	Functional / Security Sign-off	Simulation-based + UVM test + Security SVAs

UNIVERSITY OF CALGARY

General Approach

- In this investigation, we provided the 3TC spec and had CogniChip generate synthesizable RTL + security SVAs.
- Then we prompted it to produce a minimal runnable UVM environment.
    - Stimulus+driver+monitor+scoreboard+coverage matched to the RTL.
    - Key sequences (e.g., security-focused and constrained-random).
    - Running UVM in Questa
- For simulation, we also asked CogniChip to generate testbenches
    - Using Icarus Verilog
- Through interactive back-and-forth prompts
    - CogniChip iteratively helped debug the UVM/testbench and uncover design bugs

Prompts (1)



You're an experienced RTL/security engineer. I'm building a tiny Hardware Root of Trust called TeenyTinyTrustyCore (3TC) for TinyTapeout scale, so it must be small, synthesizable, and secure.

3TC contains a Device Unique Secret (DUS) that forms the basis of trust. The DUS is used to derive keys on-chip (via a small KDF), and those keys support on-core crypto services: SHA-256, HMAC-SHA-256, and AES-CTR. 3TC also exposes an immutable Device ID for identification. All cryptographic operations must stay on core to minimize attack surface.

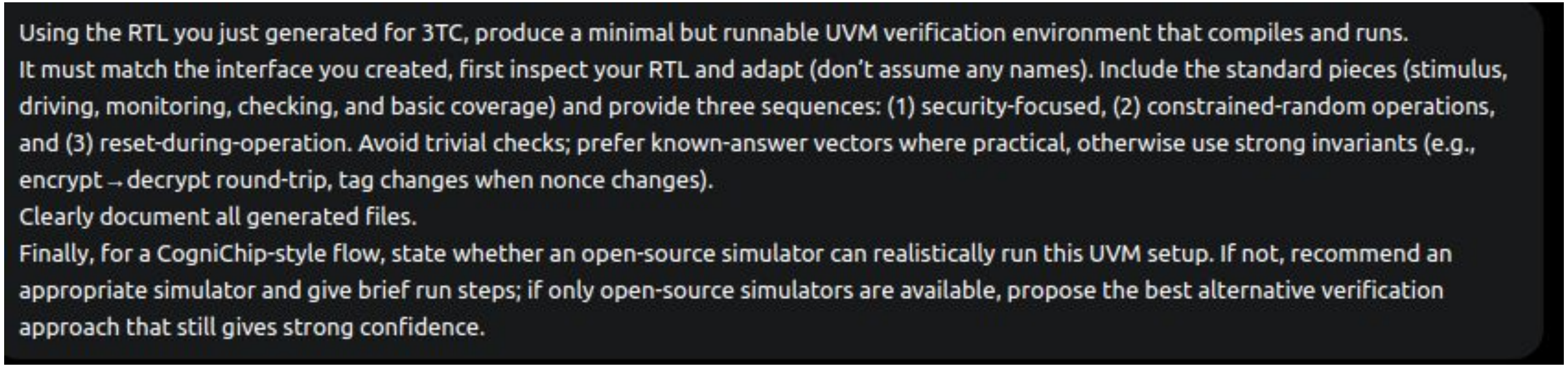
Non-negotiable security rules: DUS must never be observable outside the core (no readable path, no debug/test escape), derived keys must be zeroized after use and on reset, and Device ID must never change after reset. Assume an attacker can observe external signals and try edge cases and malformed sequences.

Please generate clean, synthesizable SystemVerilog with a small modular structure (top-level + DUS storage + KDF + SHA-256 + HMAC + AES-CTR + attestation wrapper). Use an area-minimal, multi-cycle style where appropriate.

Alongside the RTL, provide a small set of security SVAs for DUS isolation, key zeroization, and ID immutability, and add a brief comment mapping each property to the closest relevant CWE(s) (with one-line justification).

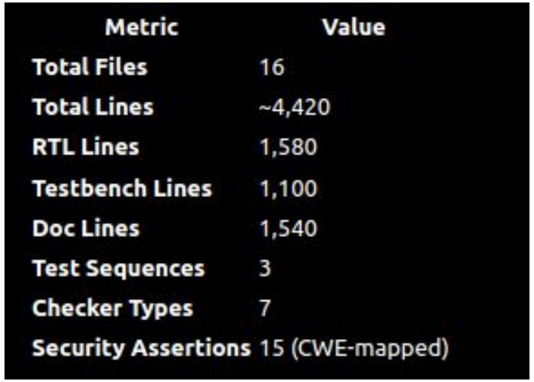
If you need to make reasonable interface assumptions, keep them minimal, consistent, and clearly documented. Output as separate files with brief comments for easy simulation and synthesis.

Prompts (2)


Using the RTL you just generated for 3TC, produce a minimal but runnable UVM verification environment that compiles and runs.
It must match the interface you created, first inspect your RTL and adapt (don't assume any names). Include the standard pieces (stimulus, driving, monitoring, checking, and basic coverage) and provide three sequences: (1) security-focused, (2) constrained-random operations, and (3) reset-during-operation. Avoid trivial checks; prefer known-answer vectors where practical, otherwise use strong invariants (e.g., encrypt → decrypt round-trip, tag changes when nonce changes).
Clearly document all generated files.
Finally, for a CogniChip-style flow, state whether an open-source simulator can realistically run this UVM setup. If not, recommend an appropriate simulator and give brief run steps; if only open-source simulators are available, propose the best alternative verification approach that still gives strong confidence.

UVM Environment for 3TC

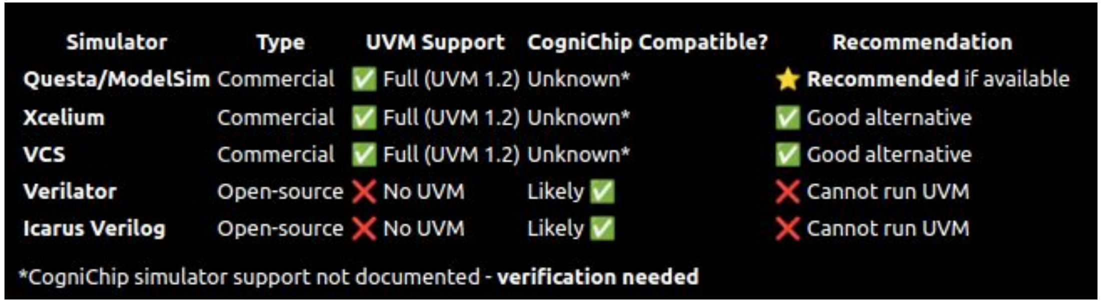
- **RTL Design (8 files)**
  - ttc3_top.sv - Top-level integration
  - ttc3_dus_storage.sv - DUS storage
  - ttc3_device_id.sv - Device ID module
  - ttc3_kdf.sv - Key derivation
  - ttc3_sha256.sv - SHA-256 core
  - ttc3_hmac.sv - HMAC engine
  - ttc3_aes_ctr.sv - AES-CTR
  - ttc3_security_sva.sv - Security assertions
- **UVM Verification (3 files)**
  - ttc3_if.sv - Interface with clocking blocks
  - ttc3_pkg.sv - Complete UVM package (~900 lines)
  - ttc3_tb_top.sv - Testbench top (DUT + clock + UVM launch)
- **Build & Documentation (5 files)**
  - Makefile.uvm - Build system for Questa/Xcelium/VCS
  - UVM_VERIFICATION_README.md - Complete UVM architecture doc (~600 lines)
  - UVM_QUICKSTART.md - Quick start guide (~400 lines)
  - TTC3_UVM_MANIFEST.md - Complete file manifest
  - TTC3_README.md - Design documentation



Metric	Value
Total Files	16
Total Lines	~4,420
RTL Lines	1,580
Testbench Lines	1,100
Doc Lines	1,540
Test Sequences	3
Checker Types	7
Security Assertions	15 (CWE-mapped)

CogniChip Compatibility + UVM Support

- Decision:
  - Choosing Questa for our primary UVM simulation baseline
- Reasons:
  - CogniChip-compatible simulators (e.g., Icarus) do not support UVM
  - Simulators with full UVM support (e.g., Questa, Xcelium) have undocumented/uncertain CogniChip compatibility



Simulator	Type	UVM Support	CogniChip Compatible?	Recommendation
Questa/ModelSim	Commercial	✅ Full (UVM 1.2)	Unknown*	⭐ **Recommended** if available
Xcelium	Commercial	✅ Full (UVM 1.2)	Unknown*	✅ Good alternative
VCS	Commercial	✅ Full (UVM 1.2)	Unknown*	✅ Good alternative
Verilator	Open-source	❌ No UVM	Likely ✅	❌ Cannot run UVM
Icarus Verilog	Open-source	❌ No UVM	Likely ✅	❌ Cannot run UVM

*CogniChip simulator support not documented - **verification needed**

Approach #2: UVM-based

	Planned	Approach	CogniChip Compatible	Done/Achieved
1	Design Specification	Interactive LLM-based	Yes	Yes
2	Prompt Engineering	UVM-based Verification Planning	Yes	Yes
3	Simulation	Icarus Verilog	Yes	Yes
4	UVM	Questa	No*	Yes*
5	Syntactic and Semantic Check	CogniChip	Yes	Yes
6	Functional Verification	Dynamic Verification	Yes	Partial
7	Security Verification	Dynamic + Formal Verification	Not Completed	Partial**,***
8	Functional / Security Sign-off	Simulation-based + UVM test + Security SVAs	Yes	Partial

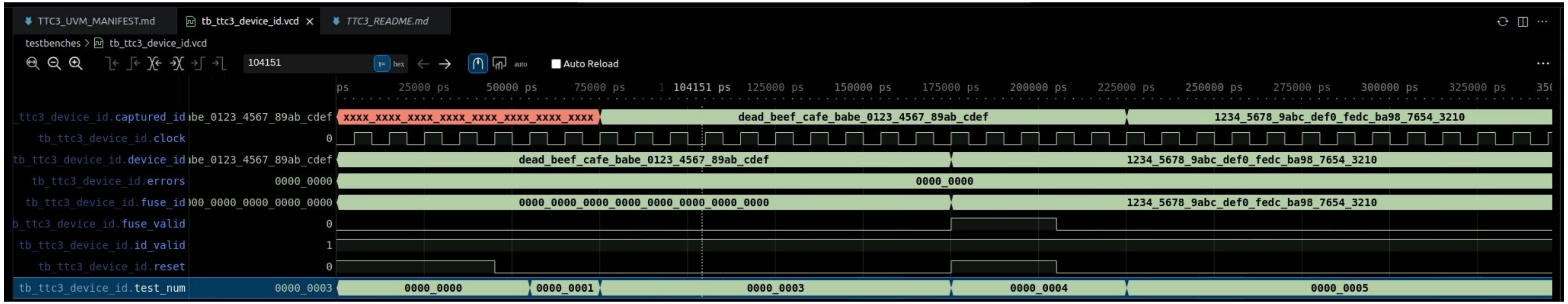
* Human-in-the-loop
** Some assertions were generated based on the desired security properties, but were never checked.
*** Through UVM verification generated environment, a security-based test generated, but was never checked.

UNIVERSITY OF CALGARY

Results and Observations

- CogniChip has no access to formal verification tools (e.g., JasperGold and VC Formal) through its available functions.
    - Thus, it could not run the generated assertions.
- We planned to investigate a complete design + verification package, but faced major practical limitations:
    - Token limits.
    - Partial verification execution (e.g., UVM tests not fully run).
    - Human-in-the-loop was required to run/debug in Questa.
    - Some CogniChip outputs were described as “ready-to-use,” but still required additional verification investigation before we could confidently claim they were mature enough to proceed to the next step.

Waveform Sample



```
_ttc3_device_id.captured_id  abe_0123_4567_89ab_cdef
     tb_ttc3_device_id.clock                        0
tb_ttc3_device_id.device_id  abe_0123_4567_89ab_cdef
    tb_ttc3_device_id.errors                0000_0000
   tb_ttc3_device_id.fuse_id  )00_0000_0000_0000_0000
b_ttc3_device_id.fuse_valid                         0
  tb_ttc3_device_id.id_valid                        1
     tb_ttc3_device_id.reset                        0
  tb_ttc3_device_id.test_num                0000_0003
```