galois

bluespec

A Formally Verified Cryptographic Extension to a RISC-V Processor

Joseph R. Kiniry, Daniel M. Zimmerman, and Robert Dockins, Galois

Rishiyur Nikhil, Bluespec

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-18-C-0013. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

Security in Computing Systems

- security in computing systems depends on having *correct and secure software*, *firmware*, *and hardware*
- formal reasoning about correctness can provide solid assurance for software and firmware, less solid but improving assurance for hardware
- formal reasoning about *system security* is just getting started

Security Assurance

- for software and firmware, security typically relies on cryptographic foundations and open development artifacts, e.g., protocol and algorithm specifications and proofs
- for hardware, security typically relies on secrecy and limited amounts of testing
  - i.e., “security by wishful thinking”
- an open ISA like RISC-V is fertile ground for changing this!

Our Proof of Concept

- a formally verified cryptographic extension (a full AES block cipher) to Bluespec's Piccolo RISC-V RV32I
- a small system integrating the modified Piccolo with an assurance case spanning hardware, firmware, and software

- here, we describe the cryptographic extension and the assurance techniques we used

Cryptographic Implementation Assurance

- assurance of crypto hardware and software is typically achieved today through validation, such as in NIST's FIPS and CAVP programs
- the only part of this validation that deals with *correctness* is CAVP tests
  - a validation lab generates large set of test vectors
  - vendor is given the test vectors and told to run them on the implementation under validation
  - vendor returns result vectors to the lab
  - lab evaluates result vectors—if all correct, thumbs up, otherwise thumbs down

Problems with Current Validation Processes

- vendors are trusted to do the right thing, instead of gaming the system by generating result vectors in whatever way is convenient
- test vectors test a tiny fraction (for AES, on the order of 10^{-72}%) of the state space
- test vectors test only *correctness*, not *security*—many insecure validated implementations (e.g., Heartbleed and the other dozen branded vulns)
- once an implementation is validated, *any change*—even security bug fixes—requires expensive and time consuming re-validation!

Applied Formal Methods for Crypto

- we can formally verify correctness and security properties of cryptographic *models* (algorithms, protocols) and *implementations*
- models are reference specifications for formal verification or rigorous validation of hardware or software implementations
- models can also be used to synthesize implementations, generate test benches, measure coverage, perform bisimulation, etc.

Applied Formal Methods for Crypto

- software implementations can be formally reasoned about in multiple ways
  - is it correct with respect to a model?
  - does it have side channel vulnerabilities?
- formal reasoning for hardware implementations is significantly less mature than for software
- current hardware “formal” tools (e.g., Cadence’s JasperGold etc.) are unable to reason about the correctness of even simple crypto algorithms
- no problem for vendors seeking FIPS or CAVP validation… but *big problem* for high-assurance secure systems!

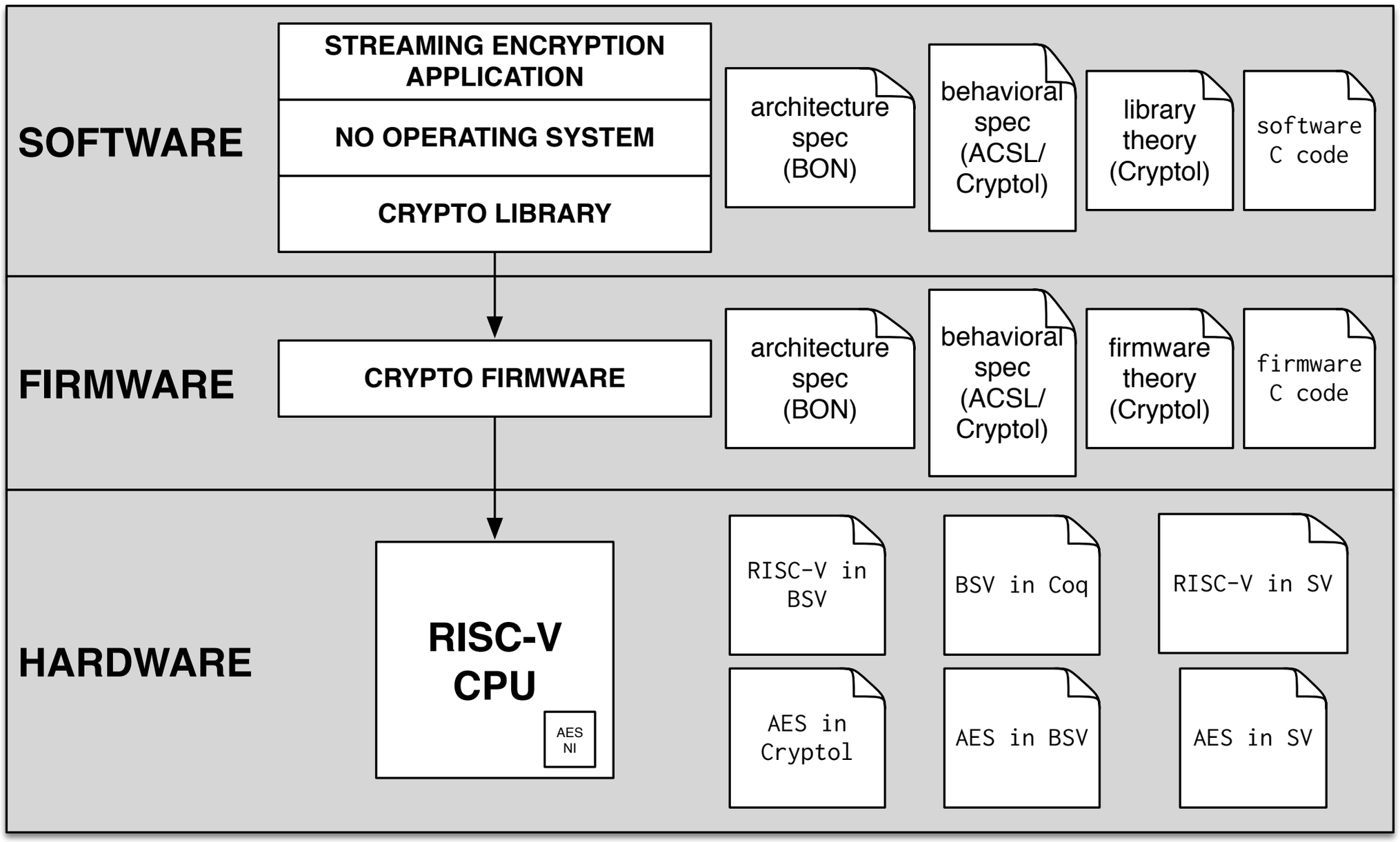
The Near Future—Validation

- improve the security validation process by introducing/mandating applied formal methods
  - enormously better assurance of correctness than sets of test vectors
  - detection of information flow issues that lead to security vulnerabilities (no side channels)
  - reuse of assurance artifacts for faster (possibly differential) revalidation

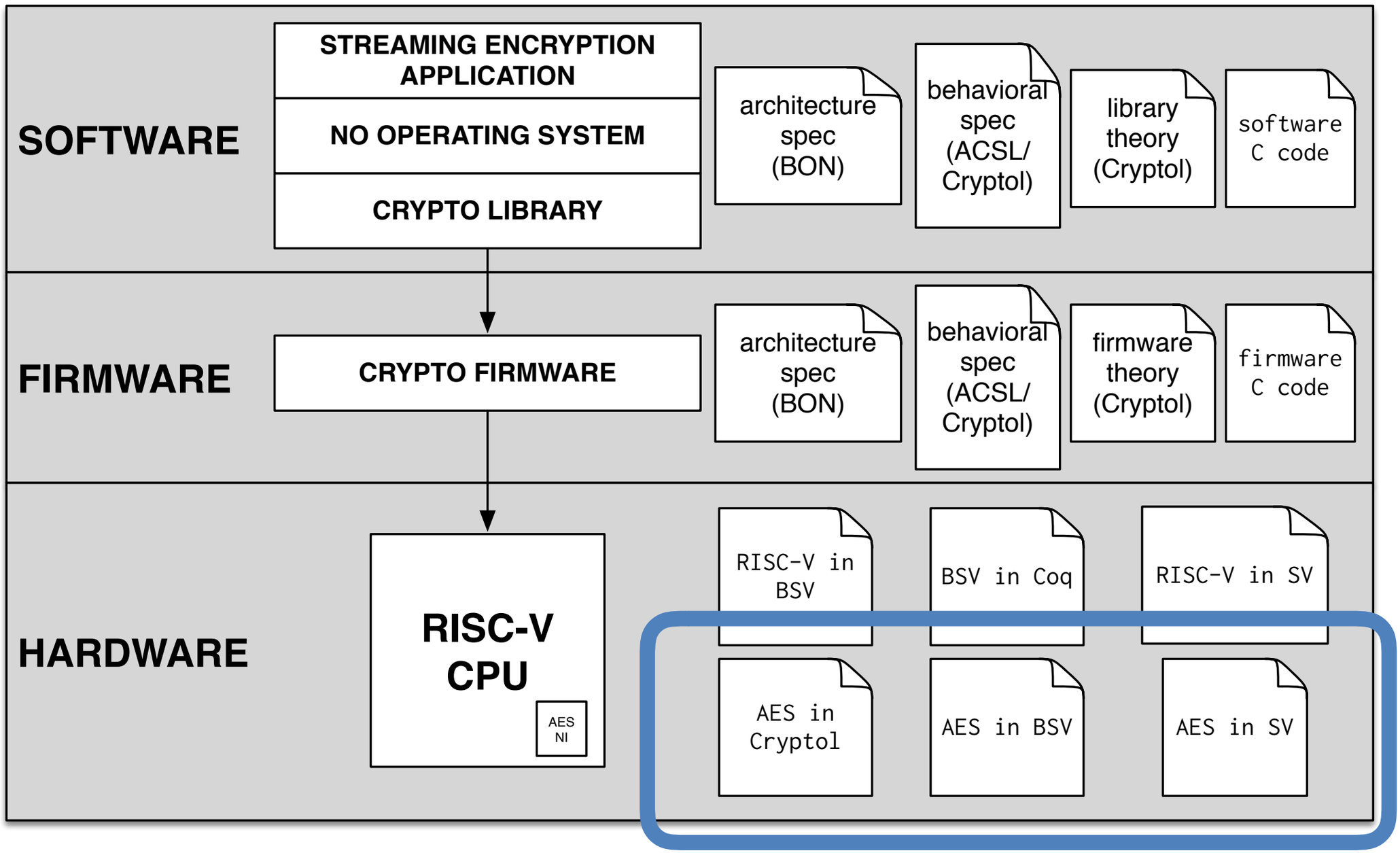
The Near Future—Hardware Assurance

- our perspective: *modern hardware engineering is very much like outdated software engineering*
- how to move hardware assurance forward
  - repurpose/adapt the best R&D in applied formal methods for software
    - choose to use automated synthesis over manually written implementations whenever possible
    - concurrently use a variety of tools and techniques to design, implement, validate, and verify cryptographic modules
- this case study is exactly along these lines

Our Case Study



Our Case Study



AES Extension to RISC-V

- written by hand in BSV, based on one of Galois's existing Cryptol specifications of AES
- not a proper RISC-V ISA extension
- implemented as a coprocessor that uses DMA to read/write key and text blocks from/to main memory, controlled and synchronized via memory-mapped CSRs

Formal Verification of AES Extension

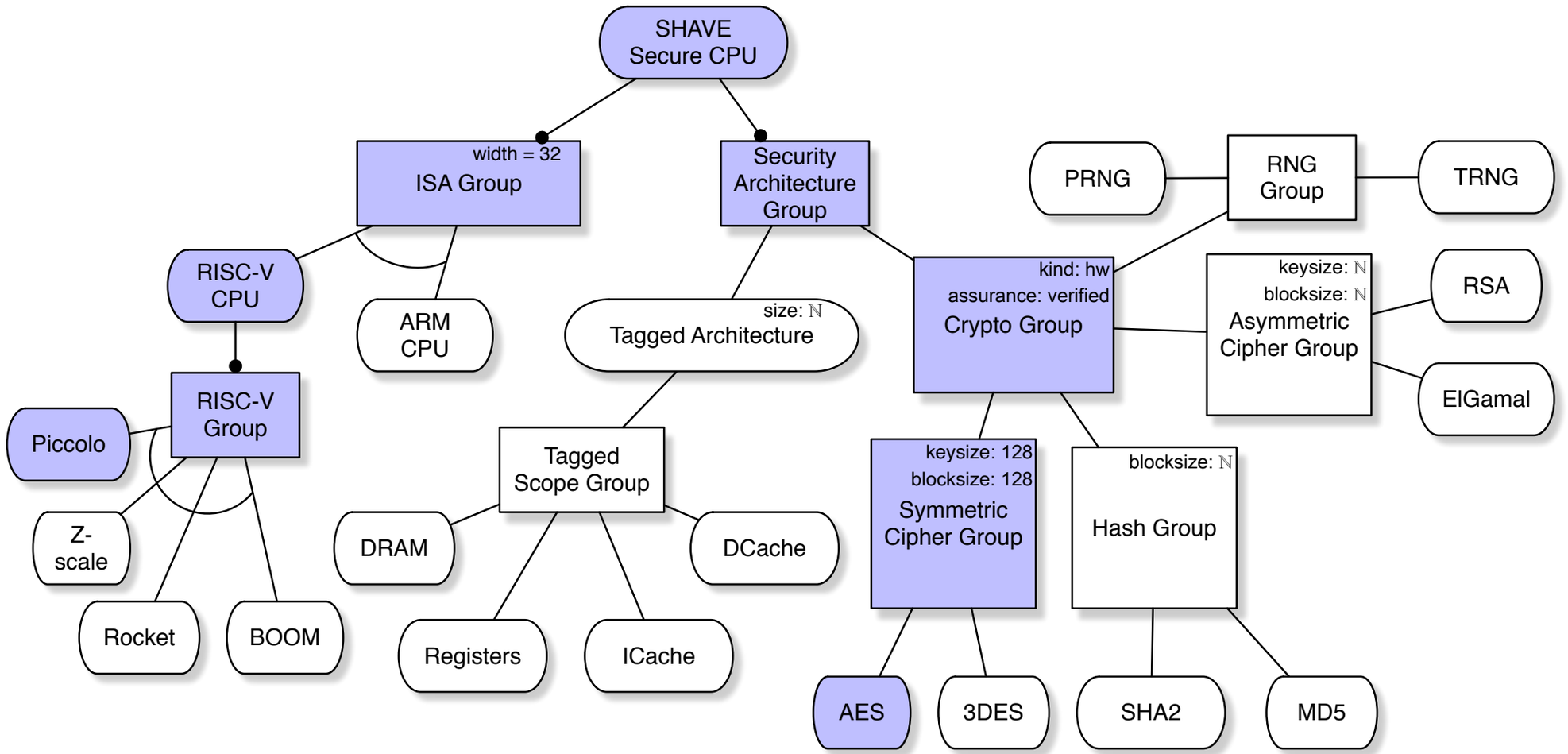
- extended Galois's existing formal methods tools (SAW) to reason about a core subset of BSV
- used multiple independent formal specifications of AES and multiple verified firmware/software implementations
- used Software Analysis Workbench (SAW) to verify mutual equivalence of AES specifications, BSV AES implementation, and firmware/software AES implementations

Rigorous Validation of Secure Systems

- automatically generate test benches from specifications when at all possible
- treat RTL like any other programming language
- hand-write ~1% of test cases that are scenario-based, automatically generate all other tests
- uniformly generate parametrized test cases by translating theorems about formal model to software, firmware, and hardware test benches

Product Line Engineering and Assurance

- treat the entire system as a product line for design, development, and assurance



Product Line Engineering and Assurance

- treat the entire system as a product line for design, development, and assurance
- configure the product by selecting variants and automatically build the same product targeting
  - software simulation of platform on an OS
  - software vs. hardware cryptography
  - software simulation of hardware
  - FPGA simulation of hardware

Ongoing and Future Work

- continue to try to use modern commercial formal verification tools to verify AES SV and understand the edge capabilities of these tools
- develop new HDL verification tools to verify currently-unverifiable properties about hardware architectures' and their implementations' correctness and security properties
- develop a DSL specification language for hardware that permits the specification of system architecture and its correctness and security properties