# Poster: Framework for Security Verification of Secure Processor Architectures

Shuwen Deng, Wenjie Xiong, Jakub Szefer
*Department of Electrical Engineering*
*Yale University*
New Haven, USA
{shuwen.deng, wenjie.xiong, jakub.szefer}@yale.edu

Doğuhan Gümüşoğlu, Sercan Sari, Onur Demir
*Department of Computer Engineering*
*Yeditepe University*
Istanbul, Turkey
doguhan.gumusoglu@std.yeditepe.edu.tr,
{ssari, odemir}@cse.yeditepe.edu.tr

***Abstract*—Secure processor architectures usually add new hardware features for protecting code and data running on the system, but today these architectures largely lack formal security verification. To address this issue, this work presents a design-time security verification framework for secure processor architectures. Our new SecChisel framework is built upon the Chisel hardware construction language and tools, and uses information flow analysis to verify the security of a design at compile-time. To enforce information flow security, the framework supports adding security tags to wires, registers, modules and other parts of the design, defining a custom security lattice and custom information flow policies. The framework performs automatic security tag propagation analysis in a parser and information flow checking using Z3 SMT solver. This framework is evaluated on RISC-V Rocket Chip expanded with AES and SHA modules.**

## I. Introduction

Many secure processor architectures have been designed over the last decade, such as XOM [1], AEGIS [2], and HyperWall [3]. They all implement some new security protection mechanisms, such as encryption and hashing, to protect code or data. However, most of these architectures have not been thoroughly and formally verified from the security perspective.

To help perform security verification of such architectures, this work proposes a new methodology and its realization in the new *SecChisel* framework, which incorporates desired security properties directly into hardware description code. To demonstrate how SecChisel can protect secure processor architectures, the framework is validated on the Rocket Chip [4] RISC-V processor by implementing and verifying AES and SHA security modules realized as Rocket Custom Coprocessor Interface (RoCC) accelerators within the RISC-V processor. In addition, synthetic hardware Trojans (HTs) are introduced into the AES and SHA modules to show how the framework can protect against secret information from leaking out if there are hardware bugs or malicious hardware Trojans.

The contributions of this work are:

- The first framework to verify security properties at the higher abstraction level of the Chisel hardware construction language, and using information flow techniques and SMT solver to formally verify an architecture.
- Flexible security verification framework supporting static and dynamic tags, declassification mechanisms, nested modules, and interference tables.
- Evaluation the functionality and performance of the framework with AES and SHA RoCC, showing fast run time and ability to detect information leaks due to hardware Trojans.

## II. Framework

New SecChisel framework extends the existing Chisel [5] language and tools with new security functionality. The SecChisel workflow is shown in Figure 1. A designer-specified lattice indicating the security level of data is written as part of the SecChisel code. SecChisel extends data types within Chisel with security tags showing the security level. This allows designers to annotate the design with the security tags associated with the various wires, registers, or other parts of the design. We include idea of both static and dynamic tags. For dynamic tags, the designer can specify the tag-range functions in SecChisel code as well.

A design in the SecChisel code is first parsed into a modified FIRRTL (Flexible Intermediate Representation for RTL) [6] and then the logical statements that can be used with the Z3 SMT solver [7] are generated. The statements check for information flow violations based on the security tags. The SMT solver is used to assert that there are no disallowed data transfers between registers, wires, etc., which could violate the security policy for enforcing confidentiality: it is not allowed that variables bound to the higher security tags leak their data to the variables with lower security tags. The security verification steps can be done in parallel with compilation and simulation of a Chisel design.

The whole SecChisel workflow consists of:

1) **SecChisel Code** – code that defines a design, including the new security tags, lattice tag description, and dynamic tag-range functions.
2) **SecChisel Parser** – the tool used to generate the modified FIRRTL that contains not only functional description of the circuits, but also information about the security tags.
3) **FIRRTL Code** – FIRRTL code with information of security tags.
4) **SMT Code Generator** – tool for analyzing the information flow and parsing FIRRTL into a FIRRTL statement/expression tree, which is then processed into SMT statements understood by an SMT solver by dealing with nested modules of FIRRTL code.
5) **SMT Code** – SMT code describing the lattice tags, dynamic tag-range functions, data flows, and assertions for information flow checking.
6) **Parallelization** – parallelizing SMT code according to the number of processor cores available of the machine.
7) **Z3 SMT Solver** – the tool which does the actual information flow checking and generates the satisfiable or unsatisfiable result from the SMT code.

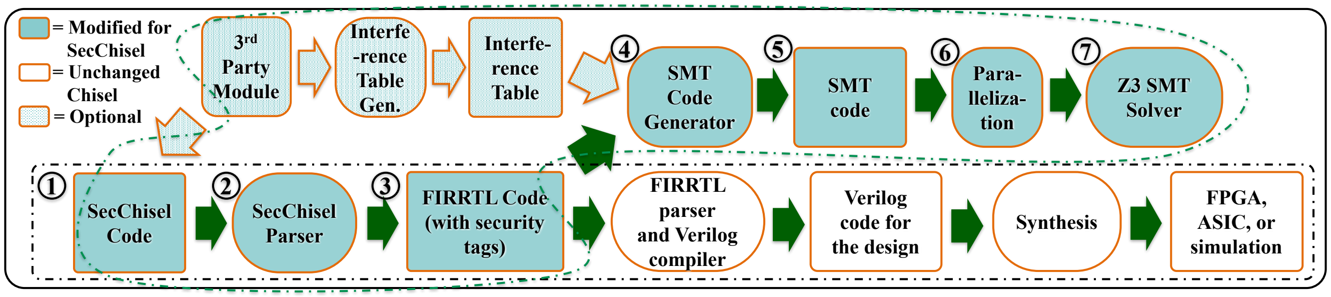

Fig. 1: SecChisel verification workflow. Square boxes represent files or data, ovals represent tools or processes. Black dashed line circles pre-existing baseline Chisel running process. Green dashed line includes whole SecChisel flow. Because of the embedded security tags of the code design, single codebase can be used for both security verification and generating the hardware design. Especially, white part shows that the unmodified Chisel tools can be used to generate the hardware design, while the new SecChisel components perform the security checks. The use of third-party modules and interference table is optional in addition to help support third-party IP.

TABLE I: Effectiveness of AES RoCC and SHA RoCC within Rocket Chip. FP represents False Positive. HT represents Hardware Trojan.

| Module Name | SecChisel Feature Used | | | Formal Verification Result |
|---|---|---|---|---|
| | Static Tag | Dynamic Tag | Declassi-fication | |
| **AES RoCC v1** | ✓ | × | × | found FP |
| **AES RoCC v2** | ✓ | × | ✓ | verified |
| **AES RoCC v2 w/ HT1** | ✓ | × | ✓ | found HT |
| **AES RoCC v2 w/ HT2** | ✓ | × | ✓ | found HT |
| **AES RoCC v2 w/ HT3** | ✓ | × | ✓ | found HT |
| **SHA RoCC v1** | ✓ | × | × | found FP |
| **SHA RoCC v2** | × | ✓ | × | verified |

8) **Interference Table** – Optional step involving third-party modules and interference table.

## III. Evaluation

SecChisel framework is implemented upon Chisel [5]. To evaluate the effectiveness and performance of SecChisel, an AES-128 and a SHA-256 accelerators were implemented as Rocket Custom Coprocessor Interface (RoCC) within the Rocket Chip RISC-V processor. The SecChisel framework can process the whole Rocket Chip as it can handle both SecChisel code and unmodified Chisel code. And in our evaluation, both AES RoCC and SHA RoCC only need tens of lines of SecChisel code to verify the whole circuit. Therefore, the designers' efforts are low. We evaluate RoCC cores within RISC-V cores, which supports high-speed simulation and full synthesis. Our framework works with the whole Rocket Chip and can find improper information flows due to bugs or hardware Trojans as shown in Table I. The framework will detect false positives which are fixed in the designs. Furthermore, three synthetic hardware Trojans are inserted in the AES RoCC, SecChisel is able to detect all of them correctly as it finds information flow of "High" data to "Low" outputs. The security verification of the designs takes an average about 1300s for each accelerator, while compilation of the Rocket Chip with RoCC takes about 1500s. This shows that the verification step can be done in parallel with the compilation of the design and generally does not introduce new overhead.

## IV. Conclusion

SecChisel implements the first security verification framework based on the Chisel hardware construction language. The framework supports both static and dynamic tags, and allows system designers to implement custom security tag lattices. It is also able to handle nested modules and interference tables. For evaluation, SecChisel is tested by implementing and verifying AES-128 and SHA-256 RoCC accelerators within a Rocket Chip RISC-V processor. SecChisel is also able to detect information leaks due to hardware Trojans. Furthermore, SecChisel verification will not cause extra overhead to the original Chisel design because the total verification time in our experiments is always smaller than the compilation plus simulation time and the verification can be done in parallel with compilation and simulation.

## References

[1] D. Lie, J. C. Mitchell, C. A. Thekkath, and M. Horowitz, "Specifying and verifying hardware for tamper-resistant software," in *Proceedings of Symposium on Security and Privacy*, ser. S&P, 2003, pp. 166 – 177.

[2] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the 17th annual International Conference on Supercomputing*, ser. ICS '03, 2003, pp. 160–171. [Online]. Available: http://doi.acm.org/10.1145/782814.782838

[3] J. Szefer and R. B. Lee, "Architectural Support for Hypervisor-Secure Virtualization," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, March 2012, pp. 437–450.

[4] K. Asanovi, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[5] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1216–1225.

[6] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the firrtl language," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-9, Feb 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html

[7] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.