Invited: Automating Hardware Security Property Generation

Ryan Kastner
Francesco Restuccia
Andres Meza
UC San Diego
La Jolla, CA, USA

Sayak Ray
Jason Fung
Intel
Hillsboro, OR, USA

Cynthia Sturton
University of North Carolina
Chapel Hill, NC, USA

ABSTRACT

Security verification is an important part of the hardware design process. Security verification teams can uncover weaknesses, vulnerabilities, and flaws. Unfortunately, the verification process involves substantial manual analysis to create the threat model, identify important security assets, articulate weaknesses, define security requirements, and specify security properties that formally describe security requirements upon the hardware. This work describes current hardware security verification practices. Many of these rely on manual analysis. We argue that the property generation process is a first step towards scalable and reproducible hardware security verification.

ACM Reference Format:
Ryan Kastner, Francesco Restuccia, Andres Meza, Sayak Ray, Jason Fung, and Cynthia Sturton. 2022. Invited: Automating Hardware Security Property Generation. In *Design Automation Conference, July 10-14, 2022, San Francisco, CA.* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Most semiconductor manufacturers have dedicated security verification teams that assess their hardware, uncover potential weaknesses, and perform redesigns to make it more secure. The teams use a best effort approach to analyze the hardware and assess potential weaknesses and vulnerabilities. They work with system engineers to identify important security-sensitive assets. They use Common Weakness Enumerations (CWEs) [14] to articulate the relevant threats. They refine these threats into requirements and properties. They verify that the hardware adheres to those security properties using a mix of formal methods, simulation, emulation, and manual code review at the pre-silicon phase.

Hardware security verification may uncover sophisticated and intricate weaknesses, vulnerabilities, and flaws. However, its reliance on manual analysis limits its scalability and reproducibility. Hardware attacks are becoming more sophisticated and reach across hardware and software. This requires system-level verification to also cover the complex software and hardware interactions. System-level properties quickly go beyond the verification engineers' ability to easily reason about them. Additionally, verification efforts involving manual code review are often not well-documented or formally described making it costly to assess and challenging to reproduce.

The ideal hardware security verification process is formal and automated. The assets are exactly specified to hardware resources. The threats are well-documented, broadly understandable, and automatically refined into requirements. The requirements are specified in a formal language and then compiled into properties, which are also described formally. And the hardware specification is automatically verified to adhere to those properties or produce counterexamples indicating security violations.

The ability to quickly, easily, and automatically develop security properties based on the threat model, is an important part of formalizing and automating hardware security verification. Translating security requirements into hardware properties is a complex, mostly manual process that is a bottleneck in the hardware security verification process. Techniques that automate property generation are crucial to reduce the manual effort involved. Some tools exist that automate parts of the process. We argue that more is required to advance hardware security analysis.

This paper describes current hardware security verification practices. We use access control verification as an exemplar to highlight key aspects of the verification process. Verification requires a large investment of manual effort in order to translate from a threat model to security properties. We argue that security property generation can and should be automated; this streamlines the hardware verification process making it more scalable and reproducible. We describe some important efforts in automated property generation and discuss opportunities for future research directions.

2 HARDWARE SECURITY VERIFICATION

Hardware security verification involves simulation, emulation, formal verification, code review, post-silicon penetration testing, and a significant amount of manual analysis [15]. Careful and intense verification can uncover critical weaknesses and vulnerabilities. But the heavy reliance on manual analysis limits the ability to scale the verification process. For example, understanding and verifying behaviors within an IP core is feasible. However, it becomes much more challenging to reason about system-on-chip architectures and their interactions with firmware, system, and application software.

Large portions of the verification process are ad hoc and not formally specified, making it challenging to assess its effectiveness and difficult to replicate. For example, when performing code review, a verification engineer proceeds based upon their experience. While the end results may be documented, the verification process itself – the different steps taken to uncover vulnerabilities and the process by which the hardware is examined – is often vaguely described.

DAC '22, July 10-14, 2022, San Francisco, CA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2.1 Verification Process

There are many efforts to describe the steps of the hardware security verification process [2, 4, 11, 13]. In general, this process involves: 1) Creating Threat Model, 2) Identifying Assets, 3) Articulating Common Weaknesses, 4) Defining Security Requirements, 5) Specifying Security Properties, and 6) Verifying Security Properties.

The first step of the security verification process creates the threat model. This involves identifying important assets and defining how they can interact with the rest of the system. Assets may be subject to threats related to confidentiality, integrity, and availability. Adversaries, their intents and starting privileges are also enumerated to help compose the threats. Precisely and accurately describing these threats and how they relate to the assets is a critical component of the hardware security verification process.

Common Weakness Enumerations (CWEs) [14] play an important role in the verification process. CWEs are a category system for security weaknesses and provide understanding around common security concerns. CWEs are a natural first step towards formalizing the hardware verification process. Vulnerabilities, requirements, and properties can be mapped to specific CWEs to enable understanding and reproducibility of the verification process.

After articulating potential weaknesses, each individual threat will be elaborated to one or more security requirements to be imposed upon the hardware. Along with implementation details of the actual design like signal and register names, each security requirement will be translated into one or more security properties. These properties should be described in a formal language that can be verified using automated tools. Translating CWEs into requirements and properties is largely a manual procedure. Thus, it presents a bottleneck in scaling the hardware security verification process.

Trace properties and hyperproperties are two classes of security properties. Trace properties are exhibited in a single trace of execution, and, similarly, falsified by a single trace of execution. Trace properties are commonly used for functional verification and can be formally described, e.g., as SystemVerilog Assertions (SVAs). Many open-source and commercial functional hardware verification tools exist to determine if hardware adheres to trace properties. Hyperproperties model more complex behaviors like nondeterminism and information flow [5]. Hardware information flow tracking (IFT) tools are capable of verifying more complex security behaviors related to confidentiality, integrity, and availability [12].

The final step in the verification process is determining if the hardware adheres to the security properties. Commercial functional verification tools analyze the hardware for security trace properties. Commercial IFT hardware verification tools verify security hyperproperties using formal methods, simulation, and emulation.

Property specification is the bottleneck in the hardware security verification process. Developing tools and methodologies that help automate property generation will make the verification process less reliant on manual analysis, making it more scalable and replicatable. *Automated hardware security property generation is critical for scalable hardware security verification.*

2.2 Access Control Verification

To better articulate the hardware security verification process and its status quo, we walk through an example which performs security verification of a hardware access control mechanism. Access control is critical to help maintain integrity, privacy, and availability. Access control mechanisms help to ensure that only authorized entities can access cryptographic keys, lifecycle state, boot memory, firmware, and other critical system information.

Access control verification is challenging. As evidence, 5 out of 12 of the 2021 CWE Most Important Hardware Weaknesses [1] relate to access control systems. The difficulty stems from the complexity of the on-chip communication networks where tens to hundreds (or more) IPs communicate using a mix of modalities: memory mapped I/O, streaming, packet routing, and direct transactions. Access control mechanisms are often distributed across the chip, adding more complexity to the verification process.

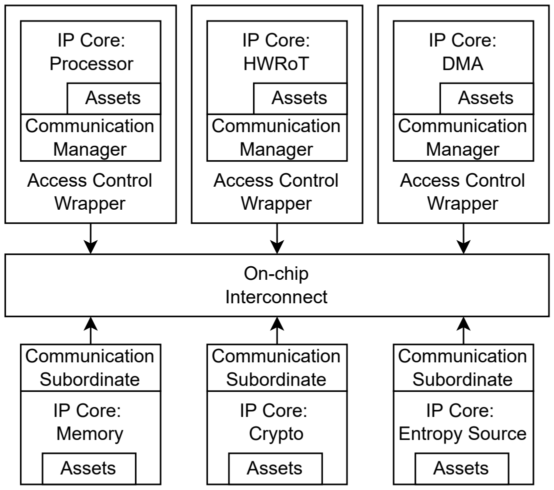

Figure 1: A system-on-chip (SoC) architecture with three IP cores acting communication managers: Processor, hardware root of trust (HWRoT), DMA. Three IP cores act as communication subordinates: Memory, Crypto core, and an Entropy Source. Assets associated with IP cores will have policies about how they can be accessed. Security verification aims to ensure that only the specified communications occur.

Figure 1 provides a simple example of a SoC architecture that requires access control. The three manager IP cores at the top have the ability to make read/write requests to the three subordinate IP cores at the bottom. Example manager IP cores include processors, hardware root of trust (HWRoT), DMA engines, graphics, and other accelerators. Example subordinate IP cores include memories (SRAM, flash, OTP), crypto accelerators, and entropy generator. IP cores have assets that could be listed as part of the threat model, e.g., keys in the crypto core and HWRoT.

Local access control mechanisms arbitrate communication at the IP level. Access to the IP cores inputs, outputs, and control/status registers may be subject to an access control policy, e.g., who can read and write to these internal resources and under what conditions. Local access control policies are relatively easy to reason about and are amenable to formal methods.

Inferring global behaviors becomes challenging as additional IP cores, each with their own local policies, are integrated into the SoC architecture. An example is the confused deputy problem where an intermediary is used to legally access a resource for someone who cannot directly access it [10]. In this case, the local access control policies are not violated, but a series of accesses enable more complex, global information flows. Global access policies quickly become intractable to verify using manual analysis alone.

Access control vulnerabilities are dangerous – potentially exposing keys and other confidential information or interfering with integrity of critical data. Thus, access control systems require a rigorous security verification process. Unfortunately, the complexity of on-chip communication makes it challenging to uncover access control vulnerabilities using manual analysis. The security verification process for access control mechanisms is ripe for automation.

2.3 Aker Access Control Framework

Aker is a design and verification framework for developing hardware access control systems [16]. Aker is built upon two pillars: the Access Control Wrapper (ACW) and a property-driven security verification [11]. Aker uses CWEs to identify potential weaknesses, templates to aid in the property generation process, and functional and IFT verification tools to validate complex global SoC behaviors.

The ACW enforces a local access control policy by monitoring requests from IP core managers. The ACWs are distributed across the SoC architecture; each ACW is programmed with a local access control policy that describe the address regions of subordinates that are legally accessible by this manager. The local access control policy is managed by a hardware root of trust. Figure 1 shows an example SoC using Aker ACWs.

The access control system plays a crucial role in the security and safety of the system. Thus, it requires security verification to ensure safe and secure operation of the SoC. Aker uses the six-step security verification process from Section 2.1. We discuss the IP-level verification in the following. The security verification for the firmware and system behaviors follows a similar procedure [16].

1) Create the Threat Model: A controller C attempts to illegally communicate with a peripheral P in a manner that violates the access control policy.

2) Identify the Assets: The assets are the M AXI signals in the AXI channels connecting the ACW to C and P and the configuration/control signals in the AXI configuration interface.

3) Articulate Common Weaknesses: We identified 17 CWEs relevant to the IP-level verification [16]. The first group relates to the Manager AXI interface. The second group relates to the configuration registers storing the local access control policy, the metadata on illegal requests, and the control logic.

4) Define the Security Requirements: Many CWEs result from a failure to properly initialize, set, and clear of the contents of security-critical registers/signals (e.g., 1258, 1266, 1269, and 1271).

5) Specify the Security Properties: Aker provides a property generation tool that creates properties using templates. For example, the following template generates properties that state no information originating in a manager M should flow to a subordinate S during active reset.

```
`signal_from_M`      //source
 when (ARESETN == 0) //tagging condition
 =/=>                //no-flow operator
`signal_to_S`        //destination
```

This template generates IFT properties that use a no-flow operator `=/=>`. For the security requirements relevant to the config/control group, we generate trace properties that specify what the value of a specific signal/register should be different various conditions. Aker provides 18 property templates. These are matched with the assets provided in Step 2 to generate 316 security properties: 164 IFT properties and 152 trace properties.

6) Verify the Security Properties: We verify the properties using formal methods (Siemens Questa Secure Check) and simulation (Tortuga Logic Radix-S).

3 AUTOMATING PROPERTY GENERATION

Now that we have described the status quo of hardware security verification, we present some ideas for making verification more scalable, reproducible, and efficient. Generating the formal security properties is one major challenge. This is largely a manual process, and thus does not scale. We believe that property generation is ripe for automation. The remainder of this section presents some approaches for automatic hardware security property generation.

3.1 Security Property Languages

The challenging and time consuming aspects of the verification process reside in the steps between creating the threat model, matching it to the relevant CWEs, and refining the CWEs into formally described requirements and properties. Automating this process requires formalizing of the security requirements and properties. *Security property languages are crucial for scalable and reproducible hardware security verification.* The major challenges relate to raising the abstraction from low-level hardware signals and behaviors to system-level, firmware, and software interactions.

The no flow operator `=/=>` offers an example on how a security property language enables the specification of IFT properties. This raises the level of abstraction and loosens the requirements for the verification engineer to understand flow labels and their associated intricacies. The rules include other operators that determine conditions under which to set labels (`when`), check for the occurrence of flows (`unless`), and explicitly downgrade flow (`ignoring`). The rules are an important first step towards abstracting low-level hardware information flows into behaviors on assets. More work is necessary to further abstract IFT and other security concepts to make it easier and more intuitive to specify properties.

Hardware security verification must build upon standards from the functional hardware verification like Property Specification Language (PSL), Universal Verification Methodology (UVM), and SystemVerilog Assertions (SVA). Merging key security language features into these standards is a natural next step. Abstracting even further and developing language features and formalizations closer to CWEs is another compelling direction.

Security properties tend to follow similar syntax with different assets and conditions. Templates are natural way to automate property generation. Section 2.3 provided an example template. Templates exist for other threat models. For example, the leakage of key materials from a cryptographic core often have a pattern where information about the key should not flow except through the output cipher text (`key =/=> all.outputs ignoring cipher`).

3.2 Property Mining

One promising approach to generating security properties is to mine relevant properties from the design itself [18]. Specification mining simulates the design, collecting trace data about the value of each signal of the design at each clock cycle, and mines the trace data for behaviors captured as logical assertions [9, 17]. *Security specification mining* uses the same workflow, but has the added criteria that the mined properties should be critical to the security of the design. This distinction is important. Naive specification mining can produce hundreds of thousands of properties [8]. Prioritizing these properties is an important and necessary procedure.

Isadora is a security specification mining tool that creates an information-flow specification of a hardware design [7]. Automatically generating information-flow properties is a challenge because, unlike standard assertions, hyperproperties are not amenable to trace-based mining techniques. Isadora combines IFT tools with security specification mining to enable security specification mining of IFT properties. The IFT tools provide important meta-data about information flows, i.e., whether a signal is affected, either explicitly or implicitly, by the source signal (asset). Isadora performs specification mining over the functional values and IFT labels. The mined IFT properties are verified with existing hardware IFT tools.

We evaluated Isadora on the Aker ACW [7]. The result is a full specification of how, and under which conditions, information flows through the ACW. The specification consists of 153 properties. Isadora demonstrates the automatic generation of information flow specifications to enable the use of powerful formal verification tools to validate the security of our hardware designs.

3.3 Benchmarks

Benchmarks are crucial for focusing the research community on threat models, designs, and problems that are relevant to industry. Benchmarking is a persistent challenge for the security and hardware communities. Many hardware designs are proprietary. Security weaknesses and vulnerabilities found during the hardware verification process are often not disclosed publicly. Academic benchmarks are often not representative of real-life industrial challenges. System-level security issues are much more challenging and representative of industry concerns.

Open-source hardware movement makes it easier to get larger, more realistic system-level benchmarks. Unfortunately, these do not have well-articulated security weaknesses or vulnerabilities. There needs to be more effort spent in providing security verification specifications, frameworks, and tools for these open-source SoCs.

The Hack@Event provides vulnerabilities inspired by real issues found in complex industrial SoCs [3, 6]. The competitions cover different aspects of hardware security. These provide benchmarks and vulnerabilities that should be used by the research community.

MIT Common Evaluation Platform is a surrogate SoC design for trusted US government hardware. It assesses technologies related to hardware Trojans, reverse engineering, side channels, and supply chain, which are less important for non-government hardware.

TrustHub focuses primarily on IP related issues with a heavy emphasis on lower-level physical hardware security. Trojans, PUFs, and logic locking are well-documented here, but system-level issues lack appropriate representation. The Security Property/Rule Database is a nice initial effort to define security properties.

Aker represents a small but important benchmark for hardware security verification. Aker can be used to create arbitrarily large access control systems. Integrating Aker into larger open-source SoCs would provide more examples of SoC security challenges.

OpenTitan is an open-source HWRoT representative of HWRoTs used in commercial silicon. OpenTitan includes extensive security verification specification and documentation. Further efforts to standardize the security verification process using properties would further enhance the security verification.

4 CONCLUSION

Security verification is an increasingly important part of the hardware design process. Yet, this remains largely a manual effort that limits its scalability and reproducability. Property generation is a key part of formalizing the process. Initial efforts to automate this process have emerged, and more techniques to generate security properties is necessary to advance hardware security analysis.

REFERENCES

[1] 2021. *MITRE CWE Most Important Hardware Weaknesses.* Technical Report.

[2] 2021. *Security Annotation for Electronic Design Integration Standard.* Technical Report. Accellera Systems Initiative.

[3] 2022. Hack@Event. http://hackatevent.org/.

[4] Sohrab Aftabjahani, Ryan Kastner, Mark Tehranipoor, Farimah Farahmandi, Jason Oberg, Anders Nordstrom, Nicole Fern, and Alric Althoff. 2021. Special Session: CAD for Hardware Security-Automation is Key to Adoption of Solutions. In *2021 IEEE 39th VLSI Test Symposium (VTS).* IEEE, 1–10.

[5] Michael R Clarkson and Fred B Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[6] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M. Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. 2019. HardFails: Insights into Software-Exploitable Hardware Bugs. In *28th USENIX Security Symposium (USENIX Security 19).* USENIX Association, 213–230. https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky

[7] Calvin Deutschbein, Andres Meza, Francesco Restuccia, Ryan Kastner, and Cynthia Sturton. 2021. Isadora: Automated Information Flow Property Generation for Hardware Designs. In *Workshop on Attacks and Solutions in Hardware Security.*

[8] Calvin Deutschbein and Cynthia Sturton. 2020. Evaluating security specification mining for a cisc architecture. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* 164–175.

[9] Sudheendra Hangal, Sridhar Narayanan, Naveen Chandra, and Sandeep Chakravorty. 2005. IODINE: A tool to automatically infer dynamic invariants for hardware designs. In *42nd Design Automation Conference (DAC).* IEEE.

[10] Norm Hardy. 1988. The Confused Deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38.

[11] Wei Hu, Alric Althoff, Armaiti Ardeshiricham, and Ryan Kastner. 2016. Towards property driven hardware security. In *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV).* IEEE, 51–56.

[12] Wei Hu, Armaiti Ardeshiricham, and Ryan Kastner. 2021. Hardware information flow tracking. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–39.

[13] Hareesh Khattri, Narasimha Kumar V Mangipudi, and Salvador Mandujano. 2012. Hsdl: A security development lifecycle for hardware technologies. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust.* IEEE, 116–121.

[14] MITRE. 2022. Common Weakness Enumeration. http://cwe.mitre.org/.

[15] Sayak Ray, Nishant Ghosh, Ramya Jayaram Masti, Arun Kanuparthi, and Jason M Fung. 2019. Formal verification of security critical hardware-firmware interactions in commercial SoCs. In *ACM/IEEE Design Automation Conference.*

[16] Francesco Restuccia, Andres Meza, and Ryan Kastner. 2021. Aker: A design and verification framework for safe and secure SoC access control. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD).* IEEE, 1–9.

[17] Shobha Vasudevan, David Sheridan, Sanjay Patel, David Tcheng, Bill Tuohy, and Daniel Johnson. 2010. GoldMine: Automatic Assertion Generation Using Data Mining and Static Analysis. In *Design, Automation and Test in Europe (DATE).*

[18] Rui Zhang, Natalie Stanley, Chris Griggs, Andrew Chi, and Cynthia Sturton. 2017. Identifying Security Critical Properties for the Dynamic Verification of a Processor. In *Architectural Support for Programming Languages and Operating Systems.*