Effective Methodologies to Accelerate Security Verification

Lee Anthony Grajo
Analog Devices
Cavite, Philippines
leeanthony.grajo@analog.com

Ponnambalam Lakshmanan
Analog Devices
Bangalore, India
ponnambalam.lakshmanan@analog.com

Anders Nordstrom
Cycuity, Inc.
San Jose, CA
andersn@cycuity.com

***Abstract*- There is an ever-growing demand for System-on-Chips (SoCs) that contain many variants of hardware and software assets that must be protected from unauthorized access and leakage of sensitive data. It is important to use the right methodologies to accelerate the security verification process and achieve a secure silicon. This paper delves into the process of identifying security requirements from specifications and external references, the use of formal and simulation-based security verification methods and measuring security coverage as part of the sign-off strategy.**

I. INTRODUCTION

Compute systems now are so much a part of everyday life. These systems are used in various applications like automotive, Internet of Things (IoT) and many more. This causes a growing demand to build and verify secure systems that can withstand different threats and attacks and protect sensitive data.

The usage of complex systems today and their exposure to many forms of attacks makes it more important to focus on security verification. Ensuring hardware control and data path security is essential to a comprehensive system security strategy. Designers and security architects can deploy hardware security verification and analysis to proactively identify and eliminate vulnerabilities that could be exploited by attackers, making hardware security sign-off an intrinsic part of their RTL sign-off. Identifying the security assets which are distributed throughout the design and identifying possible design weaknesses is a tedious task, and it becomes an absolute necessity for designers, verification engineers and system architects to work together in identifying and validating them.

In security verification, the verification engineer must make sure that the design can withstand all possible attacks. This is not as straightforward as verifying the functionality of a design and therefore poses some challenges. Security-related design vulnerabilities are not easily identified in design reviews or captured in design specifications or functional verification plans. The attack surface may also increase when designs become larger. Creating manual checkers for specific areas of the design become time-consuming to the point of being impractical. Including third-party IPs in the system design may introduce weaknesses when integrated with other blocks in the system[1].

Effective methodologies for security verification are introduced to help overcome these challenges. It starts with the identification of security requirements wherein secure assets and design weaknesses are considered. Using the advantages of formal and simulation-based verification ensures a more targeted approach to verifying the security requirements. The inclusion of coverage in security verification provides a metric to measure the verification effort and serves as criteria for signoff.

II. BACKGROUND

A. MITRE Common Weakness Enumeration (CWE) based security verification

The MITRE Common Weakness Enumeration (CWE) is a comprehensive, community-driven catalog of software and hardware weaknesses that can lead to security vulnerabilities[2]. It serves as a standard reference for identifying and describing flaws ranging from coding errors like buffer overflows and race conditions to design-level issues such as improper authentication or insecure default configurations. CWE information is used to map applicable weaknesses into verification goals which are documented in verification plans, augmenting the traditional functional verification test items. The security verification test items are implemented into simulation tests, formal properties, static check items, or a combination of these testing methods.

B. Functional Security Verification

Functional security verification is about ensuring that the security features of the design behave according to the specifications. Examples of designs are control mechanisms, privilege level and locking mechanisms, boot sequences

and protection logic. The verification methods are similar to non-security features of the design, wherein checkers, assertions or reference models are used to verify the expected behavior.

C. Information flow analysis-based Security Verification

Security verification based on information flow analysis originated from the need to rigorously ensure confidentiality and integrity in computing systems by analyzing how data propagates through software and hardware components. Its roots trace back to foundational work in the 1970s and 1980s on noninterference—a formal property introduced to prevent high-security-level data from influencing low-security-level outputs. Over time, this evolved into more sophisticated models like information flow tracking (IFT), which assign security labels to data and monitor their movement to detect unauthorized flows[3]. These techniques were initially applied to operating systems and programming languages, and later extended to hardware verification, especially as threats like timing side channels, speculative execution, and hardware Trojans became more prominent. Today, IFT is a cornerstone of hardware security verification, enabling designers to validate that sensitive data remains protected across complex architectures.

III. METHODOLOGIES AND SOLUTIONS

The methodology in this effort is composed of identifying the security requirements and documenting them in a security verification plan, performing verification using formal and simulation with coverage measurement, and analyzing the test and coverage results.

A. Security Requirements

Security requirements can be functional and can be captured in requirements or design specifications. An example of this type of requirement is "Secure data in memory should not be read by non-secure bus master." This requirement contains details such as the location of the secure data, the attack surface, and the security objective (confidentiality or integrity).

There are also weaknesses in the design that are not captured in the specifications. They can be identified through references such as the MITRE Common Weakness Enumeration (CWE) List. An example of CWE weakness is "CWE 1231 - Improper Prevention of Lock Bit Modification." The description does not include a secure data location, but it can include design that implements security-related logic. A requirement can be created based on the CWE description and the affected block in the design. The requirement can state: "*Block A* lock bits should not be modified after boot." This statement can be improved based on additional details in the design specifications. Identifying applicable weaknesses in the design will also help verification engineers develop test cases to target these weaknesses.

B. Formal Verification

In Formal verification, the different formal apps from formal tool vendors can be used depending on the description, example and detection methods of the specifications-based requirement or CWE-based requirement. Performing exhaustive checks in formal can greatly increase confidence in the design since it can cover all possible combinations of inputs and can hit corner-case scenarios. Block or subsystem-level designs with hundreds of cycles of sequential depth are good candidates for formal. Some formal apps are security-specific, targeting confidentiality and integrity issues in the design path by tracking information flow. Other apps are used to verify designs that implement security-specific control logic, register behavior, unwanted X-propagation, or detection of faults in the design. Fig. 1 shows some of the formal apps that can be used in security verification.

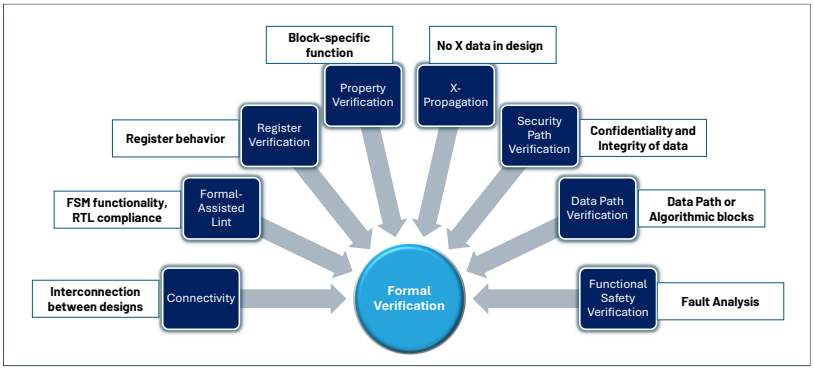

Figure 1. Formal Verification Apps for Security Verification

C. Simulation-based Verification

In simulation-based security verification, an additional testbench component called "security monitor" verifies the confidentiality or integrity requirements that are defined as "rules." The security monitor is a tool-generated SystemVerilog module that tracks information flow. This is summarized in four steps as shown in Fig. 2.

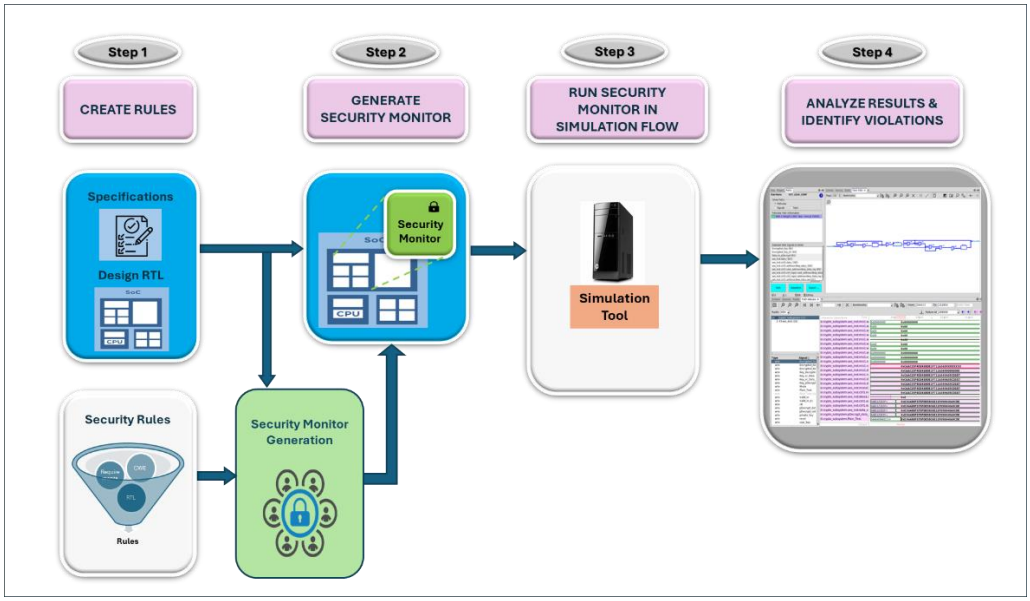

Figure 2. Simulation-based Security Verification Flow

Step 1: Creation of Security Rules

A security rule is composed of the following:

- Asset - something that is to be protected.
- Objective - that must hold with the asset ("Confidentiality" or "Integrity").
- Protection - feature that protects the asset under the objective.
- Boundary - Perimeter in the design, inside which the asset should be protected.

The security rules are created using a simple syntax shown below:

assert iflow ({ Source Signal Set } =/=> { Destination Signal Set });

Step 2: Generation of Security Monitor

The Security Monitor is a testbench component generated by a security verification EDA tool. It tracks information flows in the design as described in the rules. A script containing tool-specific commands is used to generate the monitor.

Step 3: Run Simulation

The security monitor is bound to the design using SystemVerilog bind. The selected test (or tests) which were previously created for functional verification are reused for security verification. The security verification tool injects "taint" data into the source and the monitor checks whether it reaches the destination. The rule is violated when taint reaches the destination. A passing rule indicates that the protection logic of the design has prevented any taint propagating beyond the protection boundary. The simulation log messages from the generated security monitor indicate if the rules are passed or violated.

Step 4: Analyze Results

Failing rules mean there is information flow from the source to the destination. This information flow is visualized by highlighting signals involved in the flow through the security verification tool's waveform view and path view.

D. *Security Coverage*

Security verification completeness is measured by security coverage metrics. This is different from functional coverage wherein the goal is to achieve coverage for the whole design. In security coverage, only portions of the design that are related to security are included in the coverage metrics. Currently security coverage is not yet standardized, but there are different tools from vendors for measuring security coverage. In formal, the code coverage of the security-related logic can be measured when using property verification apps. A complete verification is achieved when the code coverage requirements for the security logic is met. In simulation-based verification, the security verification tool measures coverage of the tainted signals from the source signal to the protection boundary of the rule[4]. Complete verification of a rule is achieved when the taint has propagated to all signals from the source to the protection boundary of the rule. Once all security coverage measurements are completed, the signoff is decided. The measured coverage by the simulation-based security verification tool is shown in Fig. 3.

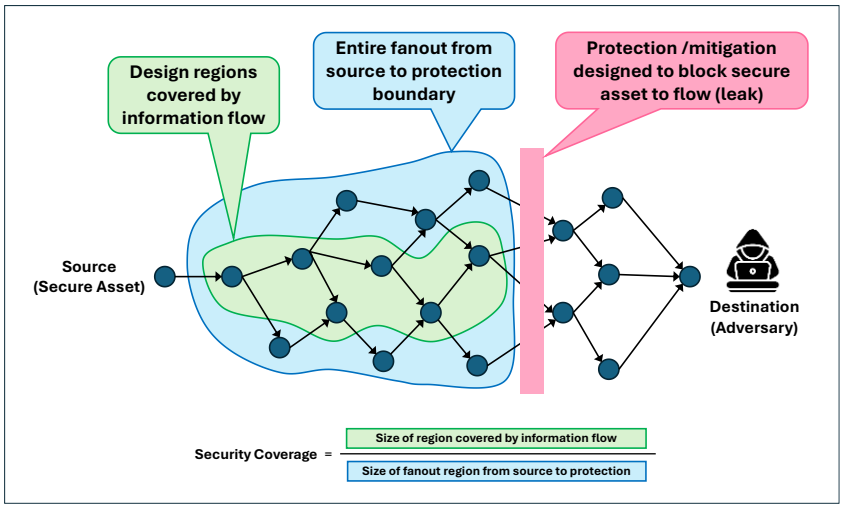

Figure 3. Simulation-based coverage measurement by the security verification tool

IV. SECURITY SYSTEM VERIFICATION

Fig. 4 shows a product containing a security system that provides security solutions such as isolation from the product's host system, perform secure cryptographic operations, and other key operations. The processor (CPU) and direct memory access (DMA) are the masters (managers) of the security system. The external communication interface is the communication channel between the product and the security system. The privilege controller sets the privilege level and acts as memory protection to manage access to memory and peripherals.

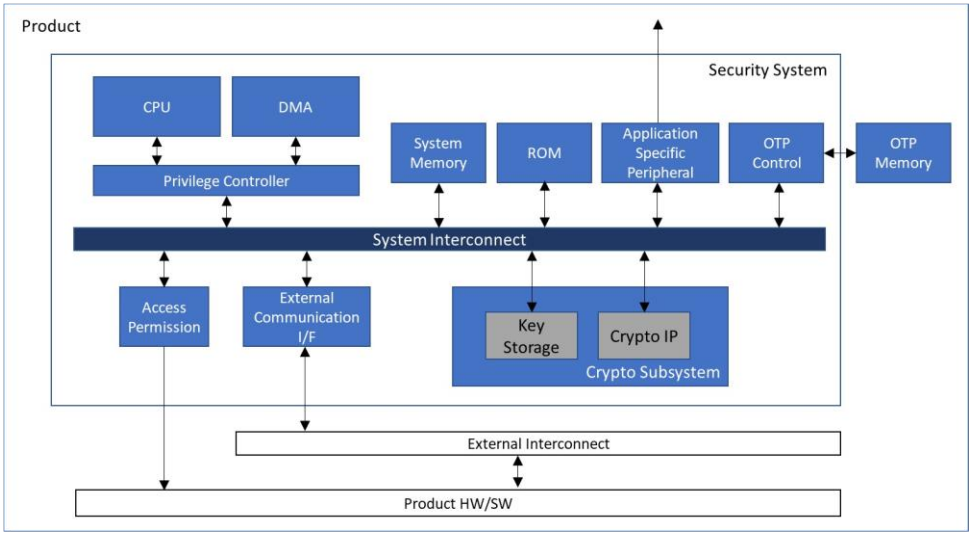

Figure 4. Security System Block Diagram

This security system was functionally verified using a combination of C-code, UVM testbench, directed tests and SystemVerilog Assertions (SVA). These verification methods are not optimized to find security weaknesses related to control logic design or confidentiality and integrity issues from information flow, therefore a security verification project was undertaken.

The security requirements are based on specifications and from checking possible design weaknesses by referring to the CWE list. Simulation was used to verify the specification-related requirements, while formal was used to verify the CWE-based requirements.

For the formal verification of the Security System design, applicable CWE-based requirements were identified and those were verified using Formal apps. Three simple steps were used to determine what formal app to use in verifying the design. The first step is to review the CWE description, examples and detection methods that can be referenced in the MITRE CWE site. The second step is to determine the design component that needs to be verified, which can be found in the design specifications or in the RTL code. The final step is to determine the formal app that matches the information in the first two steps. It can be possible that multiple formal apps can be used to test a particular requirement. Table I shows the CWE-based security requirements applicable to the security system, the design (block or top) to be verified, and the formal app used for verification.

TABLE I
Security System Requirements Verified using Formal

Requirement	CWE	CWE Description	Block/Top	Formal Tool
1. Reserved register bits should not be usable	1209	Failure to Disable Reserved Bits	Privilege Controller	Register Verification
				Property Verification
2. Register default value should be correct according to specifications	1221	Incorrect Register Defaults or Module Parameters	Privilege Controller	Register Verification
			Security System Top	Property Verification
3. Lock bits from Privilege Controller should not be modified after boot	1231	Improper Prevention of Lock Bit Modification	Privilege Controller	Property Verification
4. Privilege Controller registers should be initialized.	1271	Uninitialized Value on Reset for Registers Holding Security Settings	Privilege Controller	X-propagation Verification
				Register Verification
5. Privilege level settings should not be vulnerable to unwanted changes	1299	Missing Protection Mechanism for Alternate Hardware Interface	Privilege Controller	Security Path Verification
6. Register settings should be restricted to within the valid range	1314	Missing Write Protection for Parametric Data Values	Security System Top	Property Verification
			Privilege Controller Access Permission	Formal-assisted Lint
7. Check for paths along fabric bridge for any malfunction in access control checks, that causes key data to be leaked to unsecure manager	1317	Improper Access Control in Fabric Bridge	Security System Top	Security Path Verification

Fig. 5 shows the location in the design where formal verification is performed. The numbered items correspond to the requirements listed in Table I.

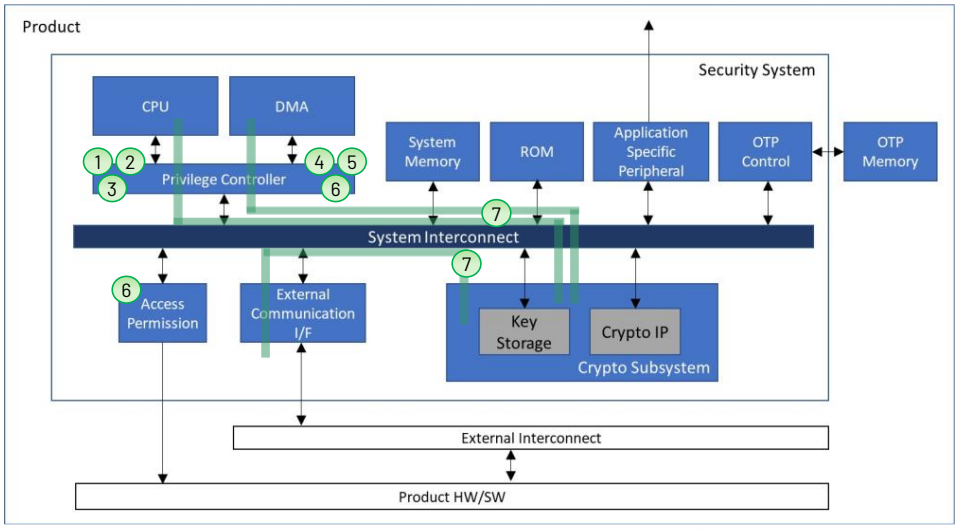

Figure 5. Security System Block Diagram with the Identified Design Areas Verified using Formal Verification

The simulation-based security verification is done using information flow checking. The secure assets and attack surfaces in the security system are identified, and corresponding security requirements are created. After creating all requirements, the security monitor is created using tool-provided script. The monitor was added to the testbench and the simulation tests were reused to verify the design. This method does not add much time to testbench development or test coding due to automation and reusability of tests.

Table II shows the security requirements for simulation-based verification. Each requirement has a secure asset and a security objective.

TABLE II
Simulation-based Verification Security Requirements, Secure Asset and Objective

Requirement	Secure Asset	Objective
Device keys and other assets stored in One-time Programmable (OTP) regions must not flow to any manager except Key DMA channel	1. Device Keys stored in OTP memory	Confidentiality
Crypto IP key must not leak outside the block	2. Crypto IP key	Confidentiality
Keys and other assets stored in Secure System memory region must not flow to any manager except Key DMA channel	3. Runtime Keys stored in System Memory	Confidentiality Integrity
Crypto IP key must not flow on the External Communication I/F which is accessible by the host processor	4.External Communication I/F and IP Key	Confidentiality
The privilege level must not be influenced by the External Communication I/F	5. Privilege Control settings	Integrity
The value of Access Permission bits (0-3, 22-23) must not be influenced by software	6. Access Permission Bits	Integrity

Fig. 6 shows the areas of the design where simulation-based security verification was given focus. Each numbered block contains the secure asset listed in Table II.

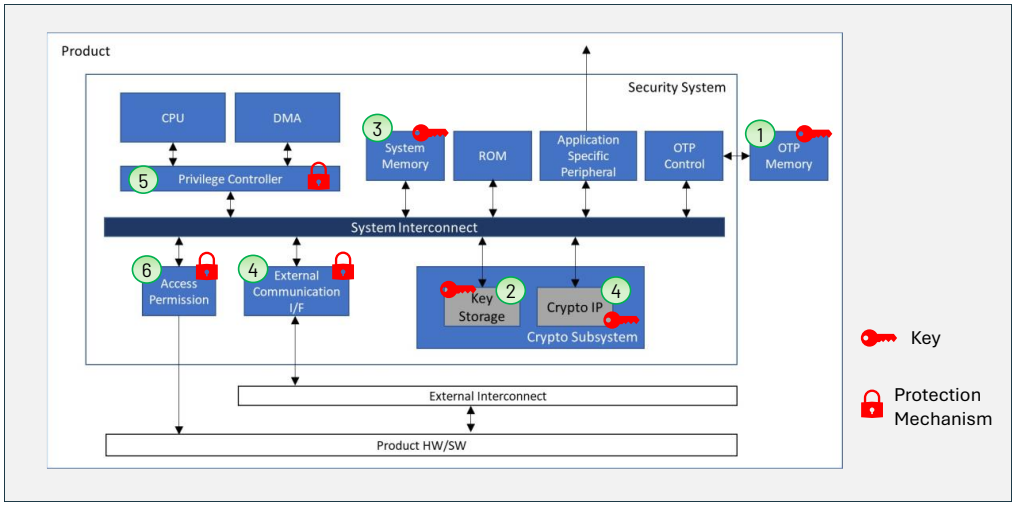

Figure 6. Security System Block Diagram with the Identified Security Assets for Simulation-based Verification

V. RESULTS

Table III shows the formal verification results. Some of the properties are automatically generated, such as those for formal-assisted lint check, register verification and X-propagation verification, while the properties used in property verification are manually generated. Most of the tests have passed, but there were few issues found. There was an out-of-bounds issue in the Access Permission block when the register setting is outside the valid range (CWE 1314). It was also determined that a non-secure CPU interface can read key information (CWE 1317).

TABLE III
Formal Verification Results

Requirement	CWE	Block	Formal Tool	Property Count	Result
1. Reserved register bits should not be usable	1209	Privilege Controller	Register Verification	1176	Pass
			Property Verification	1	Pass
2. Register default value should be correct according to specifications	1221	Privilege Controller	Register Verification	3797	Pass
		Security System Top	Property Verification	13	Pass
3. Lock bits from Privilege Controller should not be modified after boot	1231	Privilege Controller	Property Verification	132	Pass
4. Privilege Controller registers should be initialized.	1271	Privilege Controller	X-propagation Verification	2600	Pass
			Register Verification	843	Pass
5. Privilege level settings should not be vulnerable to unwanted changes	1299	Privilege Controller	Security Path Verification	22	Pass
6. Register settings should be restricted to within the valid range	1314	Security System Top	Property Verification	1	Pass
		Privilege Controller Access Permission	Formal-assisted Lint	1116	Pass/Fail
7. Check for paths along fabric bridge for any malfunction in access control checks, that causes key data to be leaked to unsecure manager	1317	Security System Top	Security Path Verification	6	Pass/Fail/ Undetermined

Using the exhaustiveness of formal helps find issues even in an already stable design. This shows that using formal for verifying critical control logic provides huge benefit in ensuring correct implementation of a secure system. However, formal does have its limitations as it can only verify hardware and convergence issues are encountered in a large and complex design. There was one property that did not achieve convergence (Undetermined result). This is where simulation-based methodologies become more effective.

Table IV shows the simulation-based verification results. Some of the reused tests initially failed since it did not include the settings to enable the protection mechanism of the design, causing the security monitor to flag an information flow violation. After test modification, the security monitor did not flag any violations and the test passed. In addition, since security verification was done when the design was already stable, there were no new errors found in the design. This result provided increased confidence that the design will not have any security-related issues.

TABLE IV
Simulation-based Security Verification Results

Requirement	Secure Asset	Objective	Result	Notes
Device keys and other assets stored in OTP regions must not flow to any manager except Key DMA channel	1. Device Keys stored in OTP memory	Confidentiality	Passed	Security verification failed when protection mechanism is not set in the test
Crypto IP key must not leak outside the block	2. Crypto IP key	Confidentiality	Passed	
Keys and other assets stored in Secure System memory region must not flow to any manager except Key DMA channel	3. Runtime Keys stored in System Memory	Confidentiality	Passed	Security verification failed when protection mechanism is not set in the test
		Integrity		
Crypto IP key must not flow on the External Communication I/F which is accessible by the host processor	4. External Communication I/F and IP Key	Confidentiality	Passed	
The privilege level must not be influenced by the External Communication I/F	5. Privilege Control settings	Integrity	Passed	
The value of Access Permission bits (0-3, 22-23) must not be influenced by software	6. Access Permission Bits	Integrity	Passed	

Table V shows the security coverage results for the formal verification effort. Code coverage was collected for the blocks but analysis was only focused on the parts of the design that contain the security control logic. Total coverage is less than 100%, but security-related logic is hit by the assertions.

TABLE V
Security Coverage Result (Formal Verification)

Block	Formal Coverage	Details
Privilege Controller	75.9%	Registers, privilege control logic and lock control logic are hit.
Access Permission	49.4%	Registers and the access control bit settings are hit.
Security System Top	12.1%	Top level assertions mainly include access permission signals, therefore only a small percentage of the top-level design was hit.

Table VI shows some of the security coverage results for the simulation-based verification effort. Coverage is measured for each rule. The coverage rate increases when protection boundary signals are added since the fanout from protection boundary to destination is excluded in the coverage metrics. The coverage analysis and addition of protection boundary signals are ongoing for some of the coverage requirements.

TABLE VI
Security Coverage Result (Simulation-based Verification)

Requirement	Security Rule Name	Security Coverage (Initial Result)	Security Coverage (After adding Protection boundary)
Device keys and other assets stored in OTP regions must not flow to any manager except Key DMA channel	OTP2Master	0.6%	> 40% (Ongoing analysis)
Crypto IP key must not leak outside the block	Crypto_KeyLeakageCore	27.6%	77.1% (Unhit areas can be ignored)
Keys and other assets stored in Secure System memory region must not flow to any manager except Key DMA channel	Mem_Conf_Secure	91.8%	>93% (Ongoing analysis)
	Mem_Integ_Secure	92%	>93% (Ongoing analysis)
Crypto IP key must not flow on the External Communication I/F which is accessible by the host processor	CryptoKey2Host	91.6%	>92% (Ongoing analysis)
The privilege level must not be influenced by the External Communication I/F	Host2Priv	91.9%	93% (Unhit areas can be ignored)
The value of Access Permission bits (0-3, 22-23) must not be influenced by software	APBits_IntgRule	92%	92% (Unhit areas can be ignored)

VI. CONCLUSION

This paper shows effective methodologies for security verification. Identification of security requirements sets the stage for determining the right verification methods to use, and using CWE ensures that requirements are not only based on functionality, but also on design weaknesses, thereby creating a more complete set of requirements. Formal verification exhaustively verified the important blocks that implement the protection mechanisms and some information flows in the design. The simulation-based security verification method also checks information flow, clearly identifies the secure assets in the design, and reuses the tests used in functional verification. The use of formal and simulation can complement in meeting the scope of security verification from block to system-level. Measuring the security verification effort through security coverage provides a solid criterion to determine signoff. This is one area of security verification that is still evolving, but early adoption of security coverage produces a more solid verification result.

VII. REFERENCES

[1] Farimah Farahmandi, Yuanwen Huang, Prabhat Mishra, “System-on-Chip Security Validation and Verification,” Springer Nature Switzerland, 2020.
[2] MITRE CWE. [Online]. Available: https://cwe.mitre.org/.
[3] Anupam Chattopadhyay, “Handbook of Computer Architecture,” Springer Nature Singapore Pte Ltd., 2025.
[4] SemiWiki, “How Cycuity Enables Comprehensive Security Coverage”. [Online]. https://semiwiki.com/podcast/video-ep9-how-cycuity-enables-comprehensive-security-coverage-with-john-elliott/.