2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

Effective Methodologies to Accelerate Security Verification

Lee Anthony Grajo
Analog Devices

Outline

- Problem Statement
- Security Verification Methodologies
- Common Weakness Enumeration
- ADI Design (Security System)
- Formal Verification (CWE-based)
  - Introduction
  - Mapping of CWE to Formal apps
- Simulation-based Verification
  - Radix overview
  - Security Verification flow
- Security Coverage
- Results
  - Formal Verification results
  - Simulation results
  - Coverage results
- Conclusion
- Acknowledgements
- References

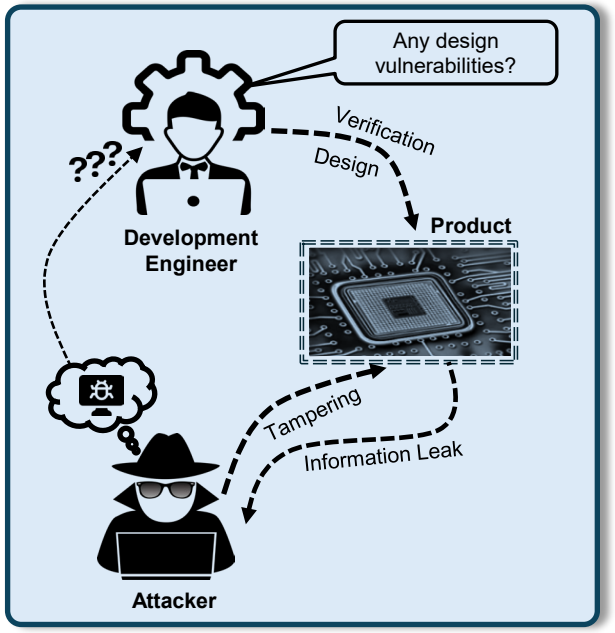
Problem Statement

Compute systems used in various applications must be secure to withstand various attacks and protect sensitive data.

- **Security Verification** becomes more important!

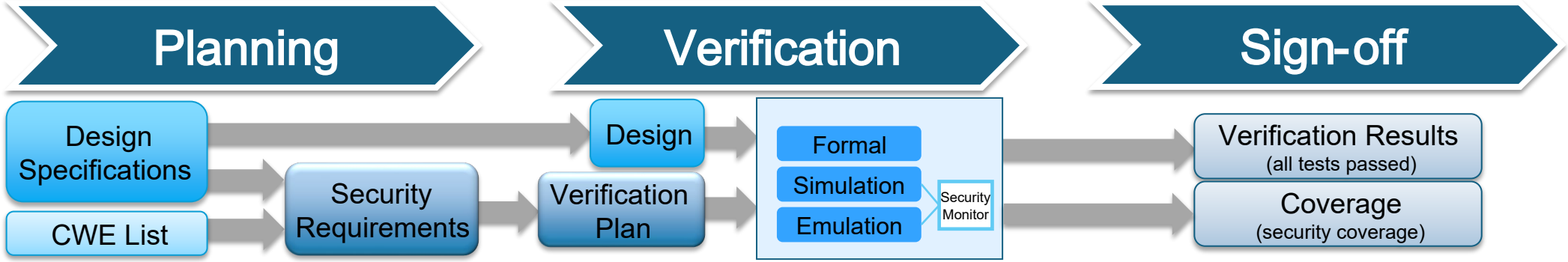
Challenges in Security Verification:

- Security vulnerabilities are not documented
  - These are often not captured in design specifications
- Larger attack surface as designs scale up
  - Attack vectors can be anything under the sun
  - Creating security checks manually is impractical
- Integrating third-party IPs may introduce design weaknesses
  - May be functionally verified but may have some security issues



Security Verification Methodologies

- We can utilize the existing methodologies used in functional verification along with tools for security verification



Identify Security Requirements or possible vulnerabilities using **CWE** as reference

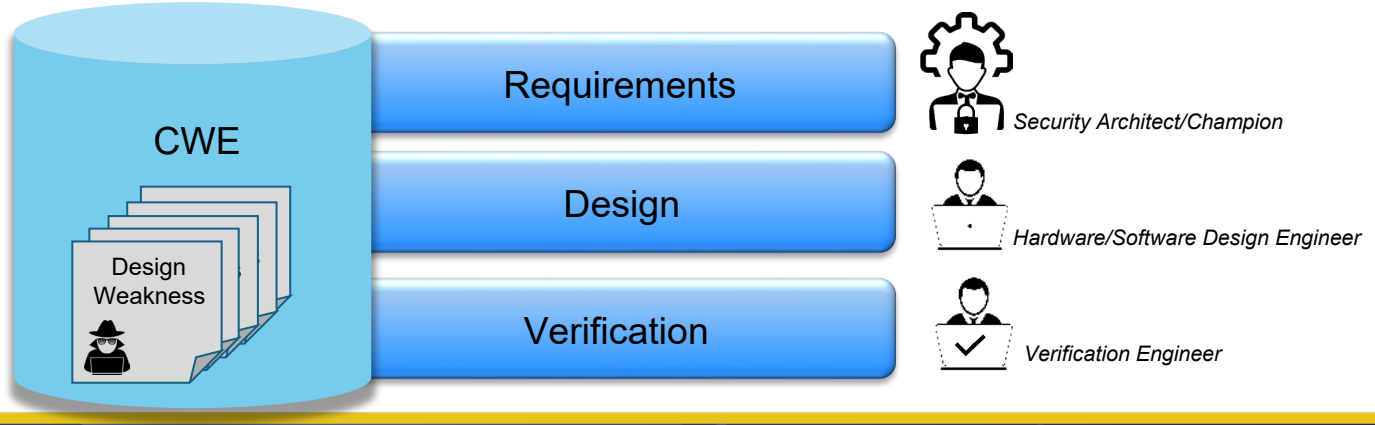
Select whether Formal, Simulation or Emulation is suited to test the security requirement

Formal – use applicable **formal apps**
Simulation/Emulation - additional **security monitor** is added for security verification

Sign off based on tangible evidence showing requirements and verified and security coverage results meet the target.

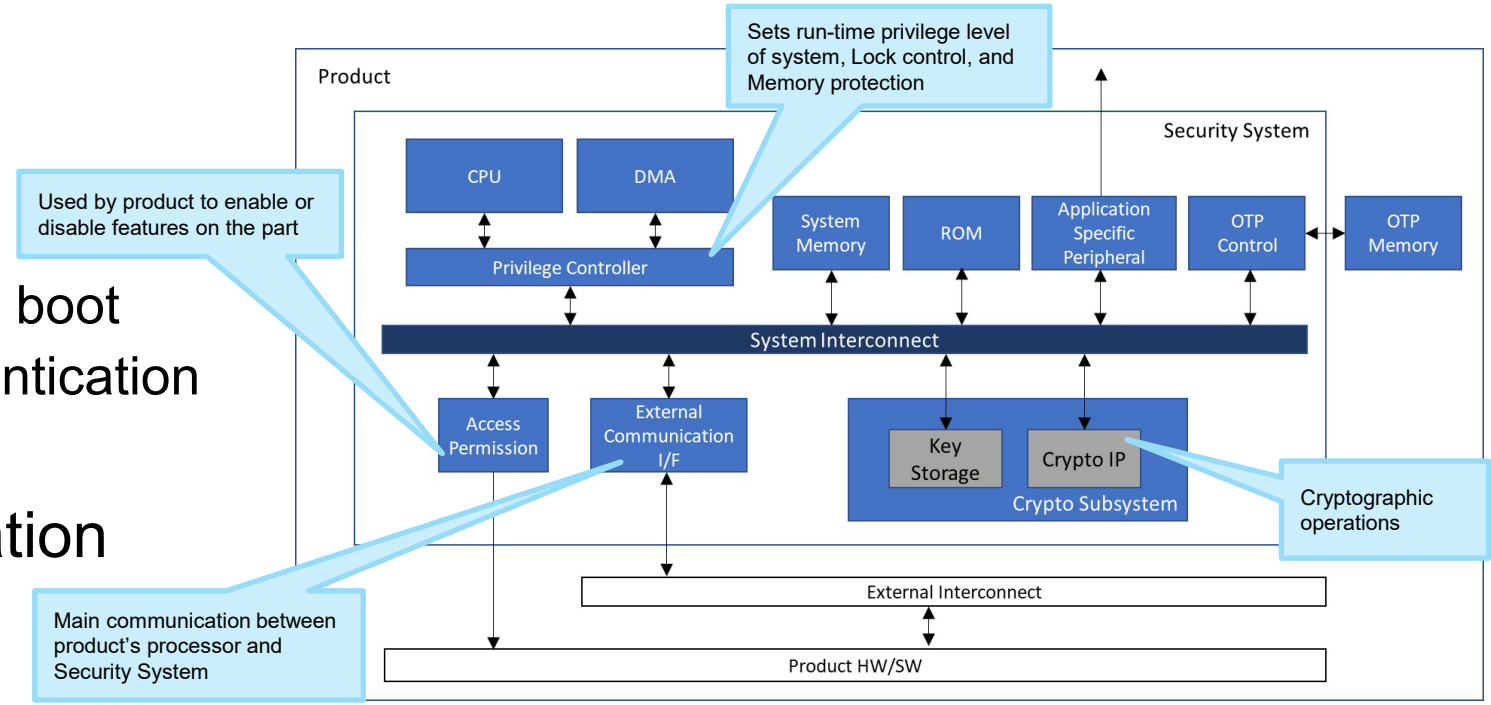
Common Weakness Enumeration

- The MITRE Common Weakness Enumeration (CWE) is a list of hardware and software weakness types that could have security ramifications.
- It is important to identify and detect weaknesses early in the development cycle.



Security System

- Reusable security solution that provides isolation to processing sensitive data
- Features
  - Secure Boot
    - Security system secure boot
    - Host secure boot authentication
  - Feature Enablement
  - Identity and Authentication
  - Key management

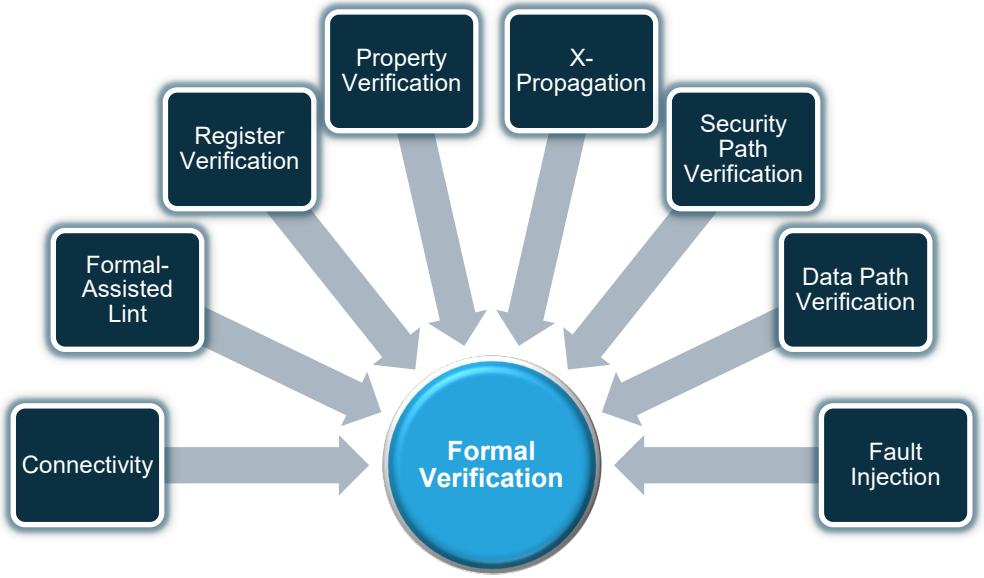


Formal Verification

Formal verification uses mathematical methods to prove that a design meets its specifications

Why Formal?

- Exhaustive in Nature
- Easy to Setup
- Can target security-related design issues



Mapping of CWE to Formal Apps

STEP 1
Review CWE Description, Examples, Detection Methods

CWE 1299

Description

The lack of protections on alternate paths to access control-protected assets (such as unprotected shadow registers and other external facing unguarded interfaces) allows an attacker to bypass existing protections to the asset that are only performed against the primary path.

Demonstrative Examples

Example 1

Register SECURE_ME is located at address 0xF00. A mirror of this register called COPY_OF_SECURE_ME is at location 0x800F00. The register SECURE_ME is protected from malicious agents and only allows access to select, while COPY_OF_SECURE_ME is not.

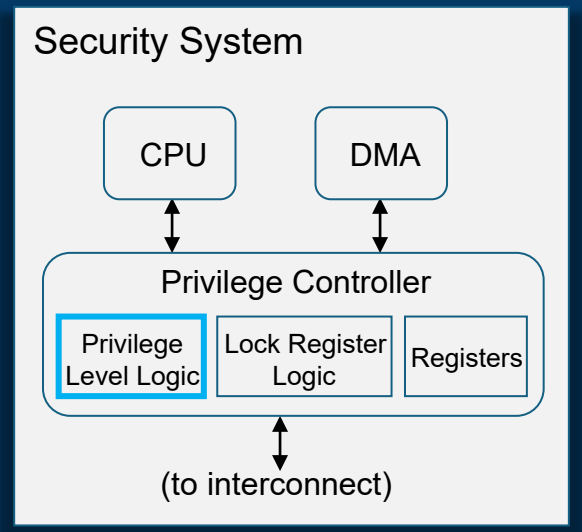
Access control is implemented using an allowlist (as indicated by acl_oh_allowlist). The identity of the initiator of the transaction is indicated by the one hot input, incoming_id. This is checked against the acl_oh_allowlist (which contains a list of initiators that are allowed to access the asset).

Though this example is shown in Verilog, it will apply to VHDL as well.

Example Language: **Verilog** *(informative)*

```
module foo_bar(data_out, data_in, incoming_id, address, clk, rst_n);
output [31:0] data_out;
input [31:0] data_in, incoming_id, address;
input clk, rst_n;
wire write_auth, addr_auth;
reg [31:0] data_out, acl_oh_allowlist, q;
assign write_auth = | (incoming_id & acl_oh_allowlist) ? 1 : 0;
always @*
  acl_oh_allowlist <= 32'h8312;
assign addr_auth = (address == 32'hF00) ? 1: 0;
always @ (posedge clk or negedge rst_n)
  if (!rst_n)
    begin
      q <= 32'h0;
      data_out <= 32'h0;
    end
  else
    begin
      q <= (addr_auth & write_auth) ? data_in: q;
      data_out <= q;
    end
  end
endmodule
```

STEP 2
Determine Design Component to Verify



STEP 3
Determine Formal App

CWE	Component	Formal Tool
1299	Privilege Level Logic (Block-specific)	Property Verification
1231	Lock Register Logic (Block-specific)	Property Verification
1209 1221 1271	Registers	Register Verification X-Propagation Verification

Formal Verification Details

1) CWE 1209 Failure to Disable Reserved bits

Reserved register bits should not be usable

2) CWE 1221 Incorrect Register Defaults or Module Parameters

Register default value should be correct according to specifications

3) CWE 1231 Improper Prevention of Lock Bit Modification

Lock bits from Privilege Controller should not be modified after boot

4) CWE 1271 Uninitialized Value on Reset for Registers Holding Security Settings

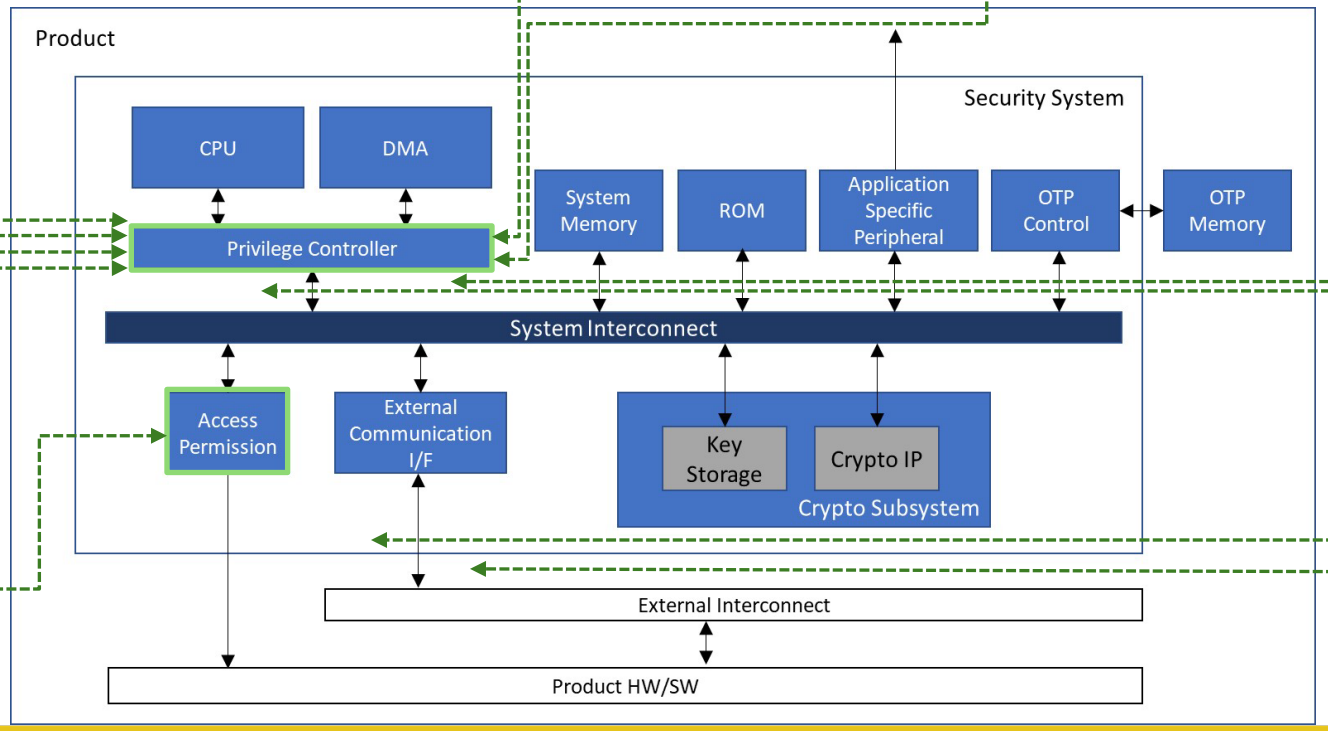
Privilege Controller registers should be initialized

5) CWE 1299 Missing Protection Mechanism for Alternate Hardware Interface

Privilege level settings should not be vulnerable to unwanted changes

6) CWE 1314 Missing Write Protection for Parametric Data Values

Register settings should be restricted to within the valid range

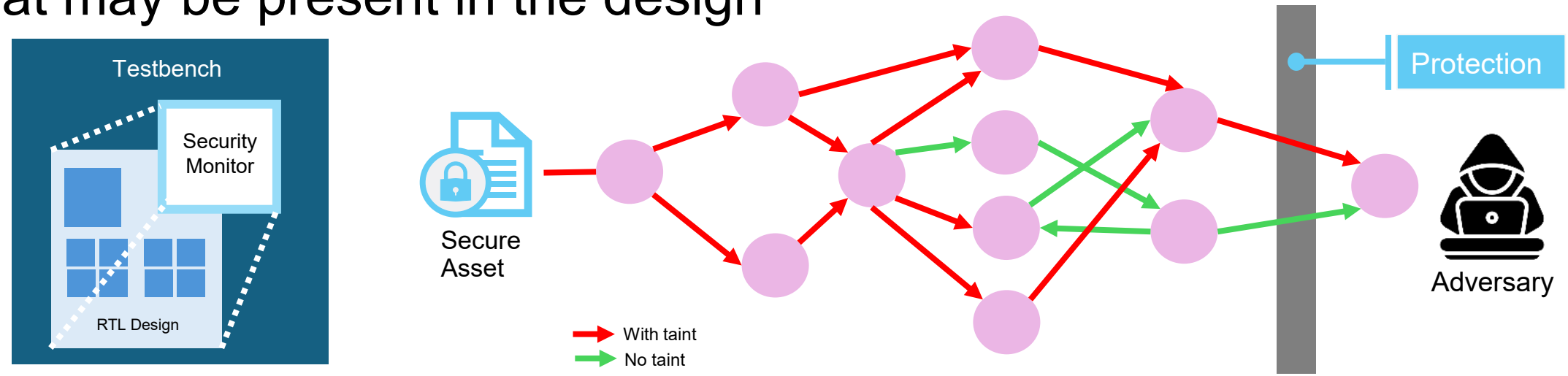


7) CWE 1317 Improper Access Control in Fabric Bridge

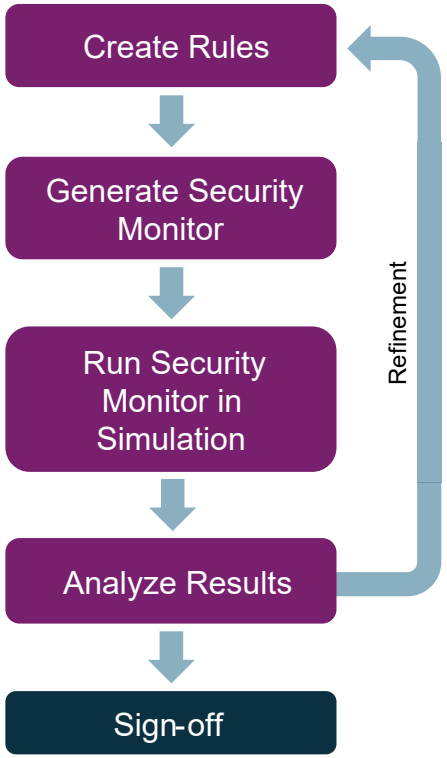
Check for paths along fabric bridge for any malfunction in access control checks, that causes key data to be leaked to unsecure manager

Simulation (with Radix)

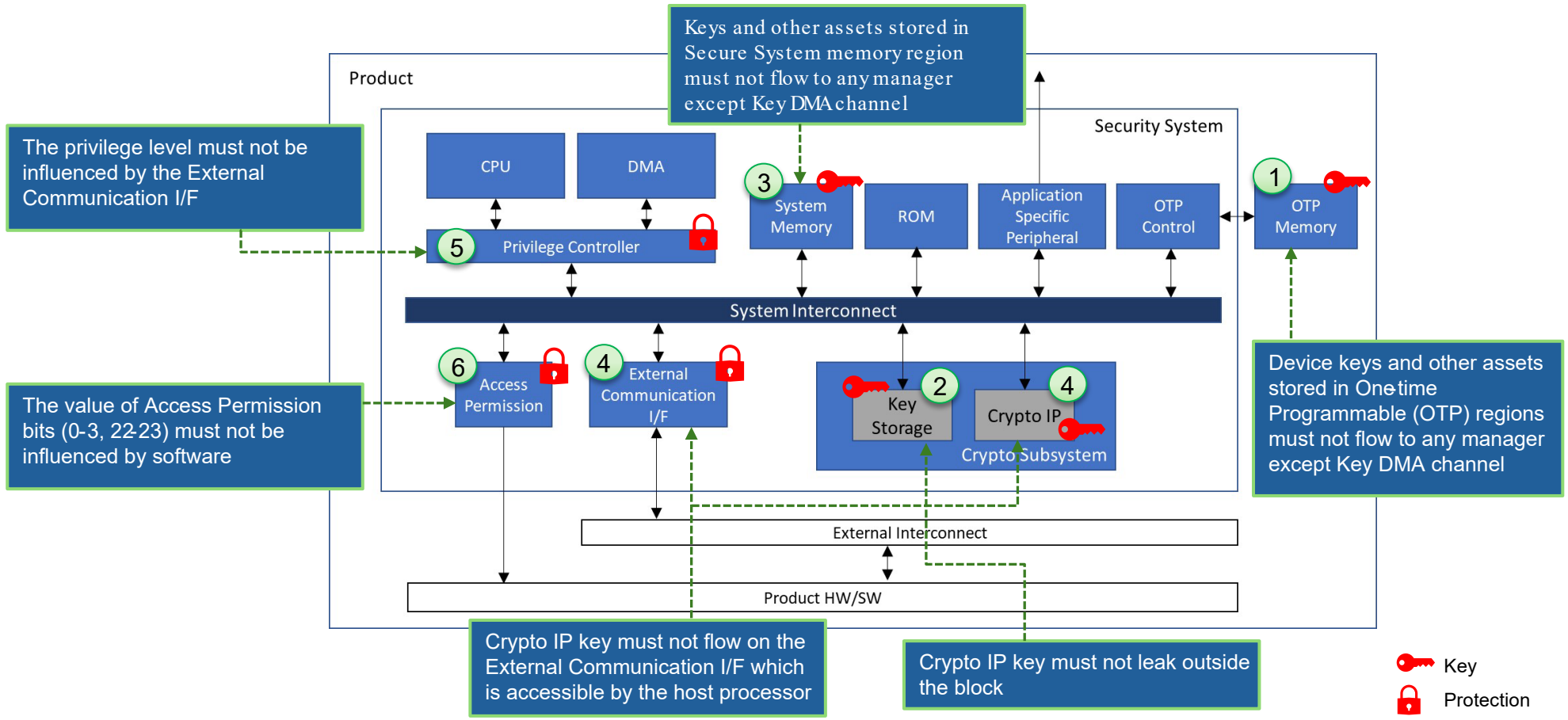
Radix is a security verification tool that automatically creates security monitors based on the input security rule. The security monitor tracks information flow to identify security vulnerabilities that may be present in the design



Security Verification Flow using Radix

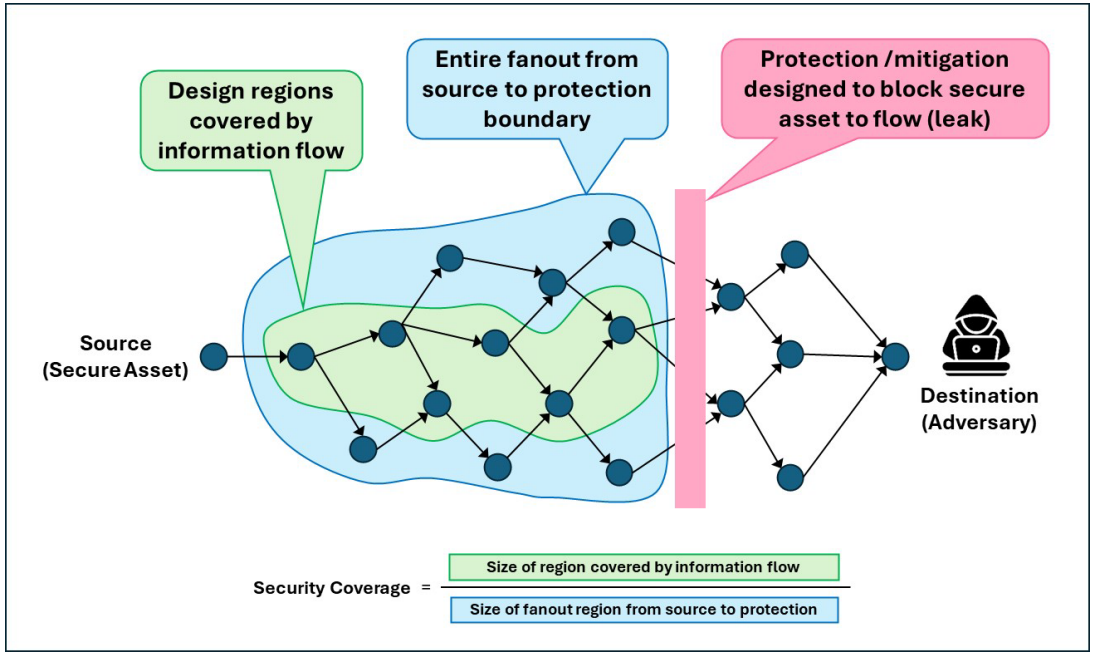


Simulation-based Security Verification Flow



Security Coverage

- Portions of the design related to security are measured for coverage
- In Simulation, this is measured by the security verification tool (Radix)
  - From source to protection boundary



Radix coverage concept

Formal Results

- Over 9000 properties
- Block-level properties proven faster compared to running simulation
- Shorter setup time

Requirement	CWE	Formal Tool	Property Count	Result	Block
1. Reserved register bits should not be usable	1209	Register Verification	1176	PASS	Privilege Controller
		Property Verification	1		
2. Register default value should be correct according to specifications	1221	Register Verification	3797	PASS	Privilege Controller
		Property Verification	13		Security System Top
3. Lock bits from Privilege Controller should not be modified after boot	1231	Property Verification	132	PASS	Privilege Controller
4. Privilege Controller registers should be initialized.	1271	X-Propagation Verification	2600	PASS	Privilege Controller
		Register Verification	843		
5. Privilege level settings should not be vulnerable to unwanted changes	1299	Security Path Verification	22	PASS	Privilege Controller
6. Register settings should be restricted to within the valid range	1314	Property Verification	1	PASS	Security System Top
		Formal-assisted Lint	1116	PASS FAIL	Privilege Controller Access Permission
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 Path Verification	6	FAIL PASS Undetermined	Security System Top

Simulation Results

- Verification was done when the design was already stable
- Result increased confidence in the design

Requirement	Secure Asset	Objective	Result
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	PASS*
Crypto IP key must not leak outside the block	2. Crypto IP key	Confidentiality	PASS
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	PASS*
		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	PASS
The privilege level must not be influenced by the External Communication I/F	5. Privilege Control settings	Integrity	PASS
The value of Access Permission bits (0-3, 22-23) must not be influenced by software	6. Access Permission Bits	Integrity	PASS

**Security verification failed when protection mechanism is not set in the test*

Security Coverage Results (Formal)

- Code coverage was collected
- Analysis was focused on parts of the design with security control logic

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

Security Coverage Results (Simulation)

- Coverage is measured for each rule

Requirement	Security Rule Name	Security Coverage (Initial)	Security Coverage (with Protection boundary)
Device keys and other assets stored in OTP regions must not flow to any manager except Key DMA channel	OTP2Master	0.6%	(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%	(Ongoing analysis)
	Mem_Integ_Secure	92%	(Ongoing analysis)
Crypto IP key must not flow on the External Communication I/F which is accessible by the host processor	CryptoKey2Host	91.6%	(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)

Conclusion

1. Identification of security requirements sets the stage for determining the right verification methods to use.
2. Using CWE ensures that requirements are not only based on functionality, but also on design weaknesses.
3. The use of formal and simulation can complement in meeting the scope of security verification from block to system-level.
4. Measuring the security verification effort through security coverage provides a solid criterion to determine signoff.

Acknowledgements

- ADI: **Albert Landicho, Larry Getzin, Nimay Shah**
- Cadence: **Tom Weiss, Franco De Seta**
- Cycuity (Arteris): **John Elliott, John Branigan**

Thank You!

References

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

Thank You!

Any Questions?