



Securing Low-latency Hardware Designs

Adam Sherer, Account Technical Executive, Cadence Design Systems
STAC Spring 2022 Summits

cādence[®]

Cyber Security: Who Cares?

Aero/Def, auto, medical, industrial, comms, IoT, semi, robotics, ...

Financial Technology



Knight Capital Group

SOFTWARE BUGS IN HISTORY

Losing \$460m in 45 minutes

Source: <https://www.bugsnag.com/blog/bug-day-460m-loss>

SECURITY HARDWARE BRANCHSCOPE

BranchScope attack successfully demonstrated on several Intel CPUs

BranchScope exploits another weakness in CPU branch prediction
By William Gayde on Mar 27, 2018, 10:26

TECH CHANGING FACE OF SECURITY

How Qualcomm Flaws Left 900 Million Android Devices Vulnerable to Spies

by David Meyer AUGUST 8, 2016, 9:59 AM EDT



Technology

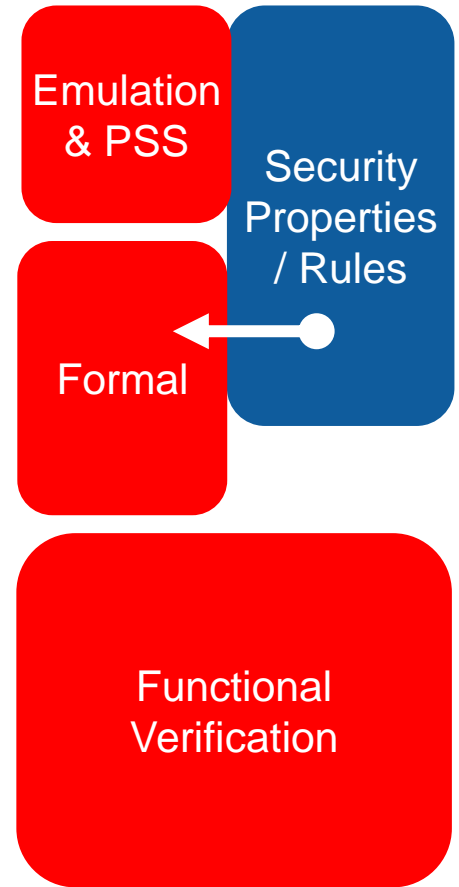
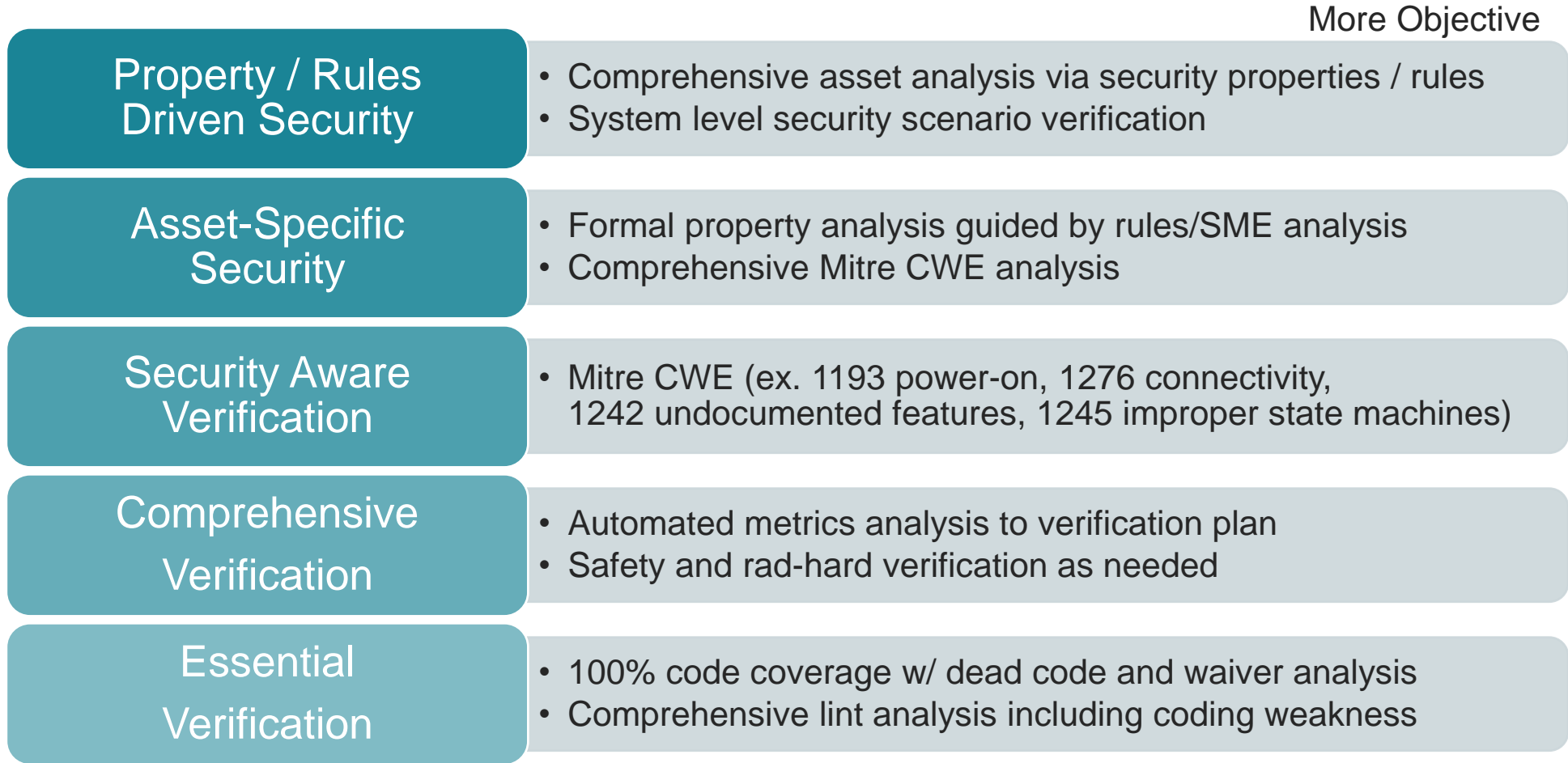
Hardware hack defeats iPhone passcode security

© 19 September 2016 | Technology



- Impact: \$460M loss in 45 minutes
- Multiple factors in loss
 - Improperly set flag put system into test mode
 - Improperly configured production env
 - Dead code in production env (“Power Peg”)
 - Lack of formal QA process
- Bug could have been a security issue
 - “Power Peg” was essentially a trojan

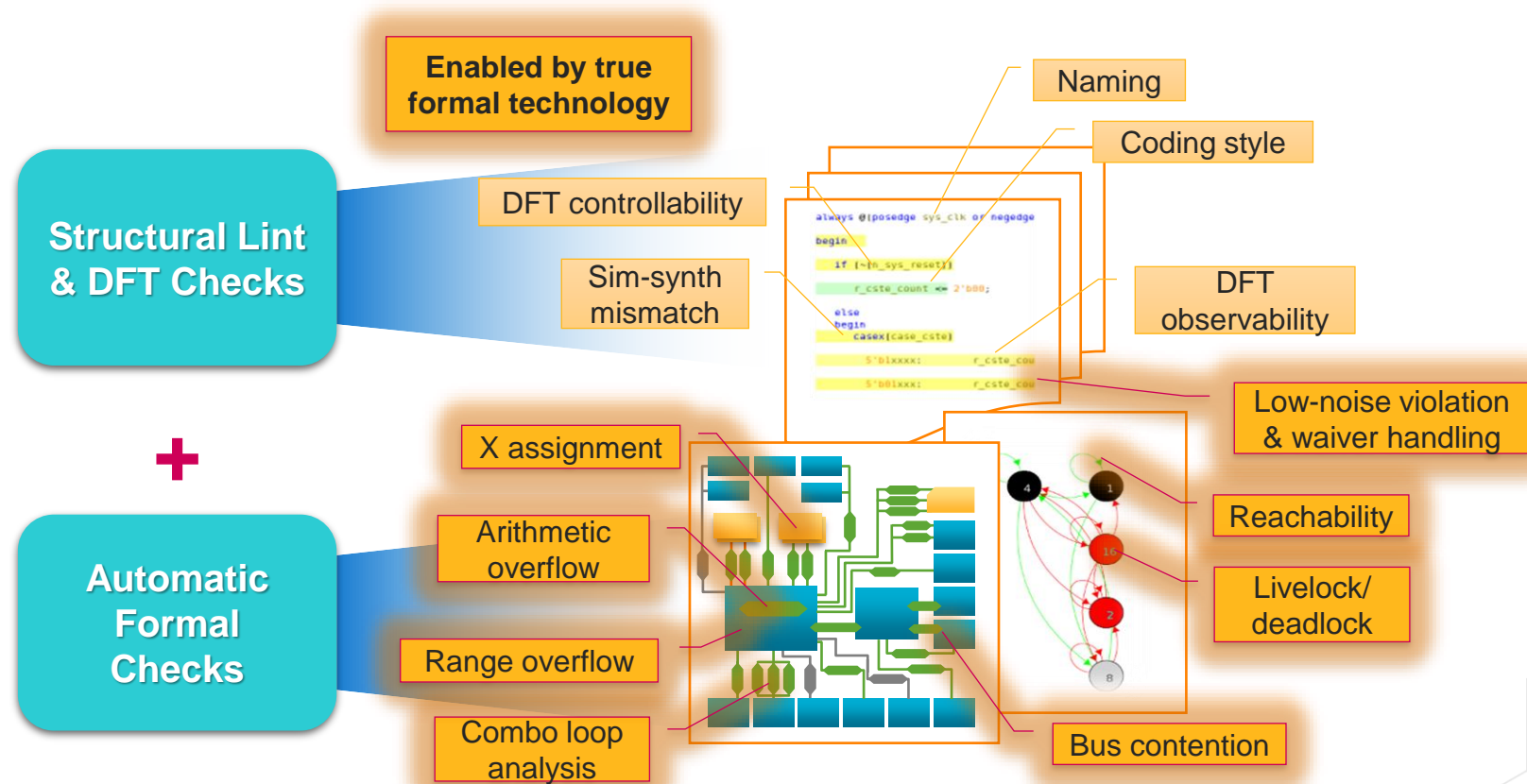
Build To Objective Security Analysis



- Security starts with a foundation of comprehensive verification
- All projects should be security aware, adding technology to support by security requirements

Using Lint Checks to Reduce Potential Attack Surfaces

- Coding style can leave a design exposed
 - Ex: side-channel exploit could be forcing data input that causes a register overflow resulting in a denial of service
 - Ex: trojan hiding in statemachine
- Examples coding issues
 - Undefined states in explicit and implicit statemachines
 - Incomplete if-then-else statements
 - Uninitialized variable/signal states
 - Livelock/deadlock states
 - Unguarded overflow/ underflow registers/ queues/ arrays, etc.
- Dead code elimination
 - Code coverage analysis

















Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

- CWE: list of known weaknesses that a secure system should not have
- Mitre Corp maintains on-line db
- Started with SW CWE, HW added about a year ago

1194 - Hardware Design

- +  Manufacturing and Life Cycle Management Concerns - (1195)
- +  Security Flow Issues - (1196)
- +  Integration Issues - (1197)
- +  Privilege Separation and Access Control Issues - (1198)
- +  General Circuit and Logic Design Concerns - (1199)
- +  Core and Compute Issues - (1201)
- +  Memory and Storage Issues - (1202)
- +  Peripherals, On-chip Fabric, and Interface/IO Problems - (1203)
- +  Security Primitives and Cryptography Issues - (1205)
- +  Power, Clock, and Reset Concerns - (1206)
- +  Debug and Test Problems - (1207)
- +  Cross-Cutting Problems - (1208)

View Metrics			
	CWEs in this view		Total CWEs
Weaknesses	96		out of 922
Categories	12		out of 316
Views	0		out of 44
Total	108		out of 1282

Security Verification Plan – Central Aggregation Point for Security Data

Security Requirements
(from req. mgmt.)

Link to
Security Specs

Security vPlan

Name	Overall Average Grade	Mapped Elements Grade	Planned Elements Accumulated
Mitre_CWE_1194_HW_Weaknesses_vPlan	29.93%	66.67%	24
203 - Observable Discrepancy	75%	100%	2
FRAME_DATA	50%	100%	1
FRAME_MSB_LSB	100%	100%	1
226 - Sensitive Information Uncleared in Resource Before Release fo...	81.7%	100%	1
regs	81.7%	100%	1
276 - Incorrect Default Permissions	16.67%	33.33%	3
dbg	50%	100%	1
Debug Inf Attack Test	0%	0%	1
Debug Inf Security Mechanism Assertion	0%	0%	1
325 - Missing Required Cryptographic Step	0%	0%	1
440 - Expected Behavior Violation	43.52%	100%	3
apb_uart_1stopbit	100%	100%	1
apb_uart_rx_tx	25%	100%	1
uart_apb_incr_data	5.56%	100%	1
1053 - Missing Documentation for Design	n/a	n/a	1
1189 - Improper Isolation of Shared Resources on System-on-Chip (...)	70.83%	100%	2
TRANS_DATA	66.67%	100%	1
TRANS_ADDR_X_TRANS_DIRECTION	75%	100%	1
1190 - DMA Device Enabled Too Early in Boot Phase	0%	0%	1
DMA Boot Attack TC	0%	0%	1
1191 - Exposed Chip Debug Interface With Insufficient Access Control	50%	100%	1
dbg	50%	100%	1
1192 - System-on-Chip (SoC) Using Components without Unique, Im...	0%	0%	1
1193 - Power-On of Untrusted Execution Core Before Enabling Fabri...	0%	0%	1
1209 - Failure to Disable Reserved Bits	0%	0%	1
1220 - Insufficient Granularity of Access Control	0%	0%	1
1221 - Incorrect Register Defaults or Module Parameters	14.58%	100%	3
reg_modeluart_ctrl_rf_ua_dv_latch0.wcov	15.62%	100%	1
reg_modeluart_ctrl_rf_ua_dv_latch0.rcov	26.56%	100%	1
reg_modeluart_ctrl_rf_ua_dv_latch1.wcov	1.56%	100%	1
reg_modeluart_ctrl_rf_ua_dv_latch1.rcov	66.67%	100%	3
1223 - Race Condition for Write-Once Attributes	0%	100%	1
reg_modeluart_ctrl_rf_ua_jer.rcov	0%	100%	1
TRANS_ADDR	100%	100%	1
TRANS_DIRECTION	100%	100%	1

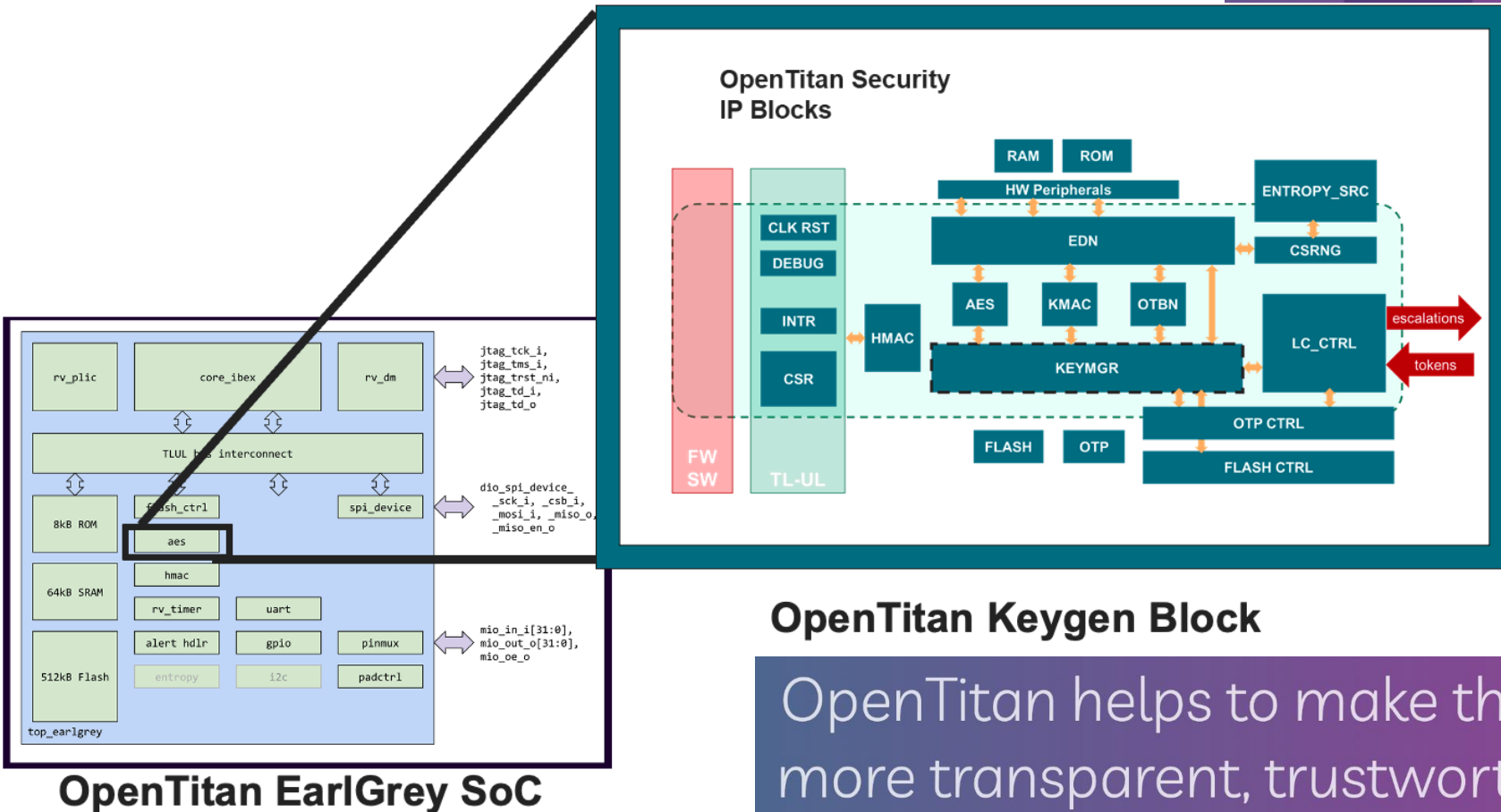
Identify
Security Metrics

Import and Reuse
Security Plans



Collaborate on common
Security Plan

OpenTitan is the first open source project building a transparent, high-quality reference design and integration guidelines for silicon root of trust (RoT) chips.



OpenTitan Keygen Block

OpenTitan helps to make the silicon root of trust (RoT) more transparent, trustworthy, and ultimately, secure.

Improper Finite State Machines in HW Logic with SuperLint

CWE	Description	Formal Application
1245	Improper Finite State Machines (FSMs) in Hardware Logic	SuperLint

- Faulty finite state machines (FSMs) in HW logic allow attacker to put system in undefined state causing denial of service (DoS) or gain privileges to system
- Formal analysis automatically extracts all states and transitions for each FSM in design
- Unreachable states/transitions indicate potential weaknesses where faults can lead the design into unknown behaviors

The screenshot displays the Cadence SuperLint interface. On the left, the 'Automatic Formal Properties' window shows a table of analysis results:

Tag	Type	Name
✓ FSM_NO_RCHB (fsm state cover)	Cover	fsm_state_cover_StOpWait_prop_4555
✓ FSM_NO_RCHB (fsm state cover)	Cover	fsm_state_cover_StClean_prop_4556
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StIdle_StIdle_prop_4557
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StIdle_StTx_prop_4558
✗ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StIdle_StTxLast_prop_4559
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StTx_StTx_prop_4560
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StTx_StTxLast_prop_4561
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StTxLast_StTxLast_prop_4562
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StTxLast_StOpWait_prop_4563
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StOpWait_StOpWait_prop_4564
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StOpWait_StClean_prop_4565
✓ FSM_NO_TRRN (fsm trans cover)	Cover	fsm_trans_cover_StClean_StIdle_prop_4566
✓ FSM_NO_RCHB (fsm state cover)	Cover	fsm state cover StCtrlDataIdle_prop_4568

The 'Analysis Browser' window shows the code for the failing property (FSM_NO_TRRN_4559):

```
154 if (adv_en_i) begin
155   rounds = La
156 end else if (
157   rounds = La
158 end else if (
159   rounds = La
160 end
161
162
163 // we are sen
164 state_d = (ro
165 end
166
167 StTx: begin
168   valid = 'b1;
169   strb = {IfBytes
170
171 // transaction
172 if (kmac_data_1
173   cnt_en = 'b1;
```

The 'Graph.5009' window displays a state transition graph with nodes labeled 'StTx', 'StTxLast', 'StOpWait', and 'StClean'. The graph shows various transitions between these states, with some transitions highlighted in red, indicating the specific failure condition.

Key Overwrite in Wipe Mode Found with Formal Property Verification

CWE	Description	Formal Application
1258	Exposure of Sensitive System Information Due to Uncleared Debug Information	Formal Property Verification

Disabled

Disabled is a state where the key manager is no longer operational. Upon Disabled entry, the working state is updated with KMAC computed random values; however, sideload keys are preserved. This allows the software to keep the last valid sideload keys while preventing the system from further advancing the valid key.

When advance and generate calls are invoked from this state, the outputs and keys are indiscriminately updated with randomly computed values. Key manager enters disabled state based on direct invocation by software:

- Advance from OwnerRootKey
- Disable operation



LFSR must be enabled in key wipe state

```
assert {u_ctrl.state_q == StCtrlWipe |-> (ctrl_lfsr_en && wipe_key)}
```

```
# key values must be filled with random data
```

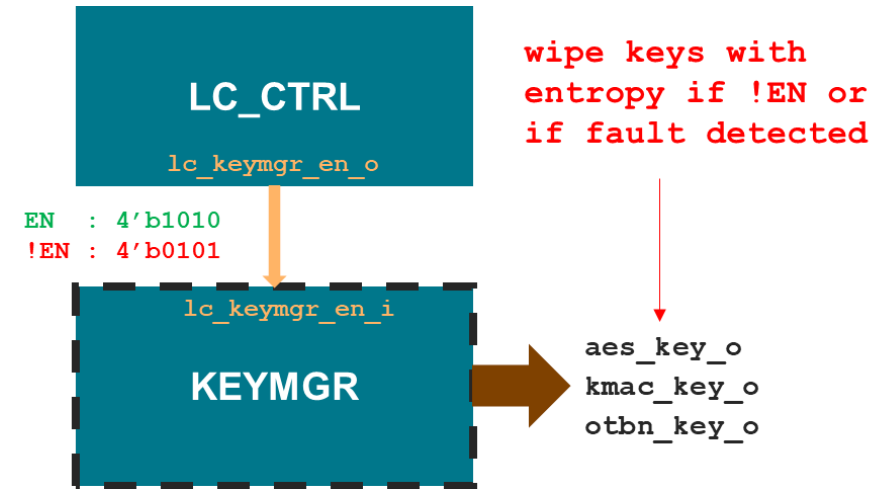
```
assert {u_ctrl.state_q == StCtrlWipe |-> kmac_key.key[0] == {8{ctrl_rand[0]}} }
```

```
assert {u_ctrl.state_q == StCtrlWipe |-> kmac_key.key[1] == {8{ctrl_rand[1]}} }
```

```
# software operations are forbidden when keymanager is disabled during key wipe
```

```
assert {!u_ctrl.en_i |-> u_ctrl.disable_sel && stage_sel == Disable}
```

▽	Type	▽ Name	▽ Engine	▽ Bound	Time	Task
❌!	Assert	in_StCtrlWipe_wipe_lfsr_en	L	336 - 374	436.1	fpv_cwe1258
❌	Cover (relat...	in_StCtrlWipe_wipe_lfsr_en:witness1	N (4)	Infinite	0.0	fpv_cwe1258
✅	Cover (relat...	in_StCtrlWipe_wipe_lfsr_en:precondition1	L	330 - 374	436.1	fpv_cwe1258



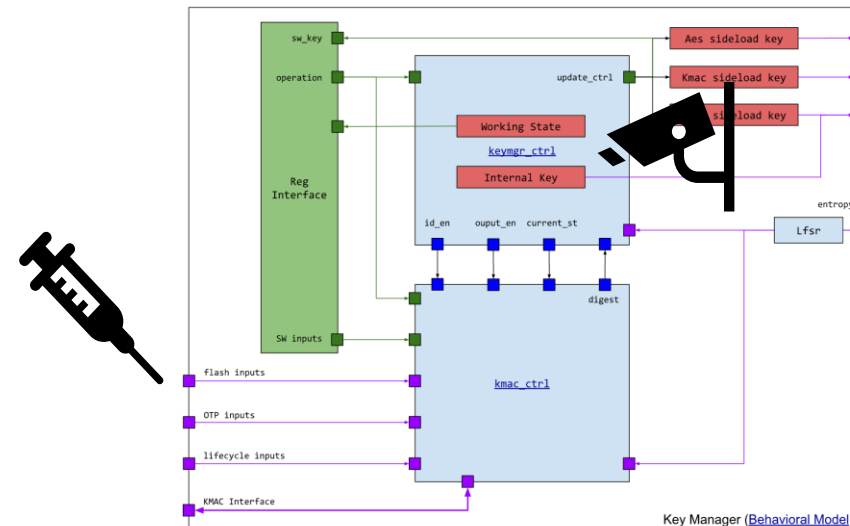
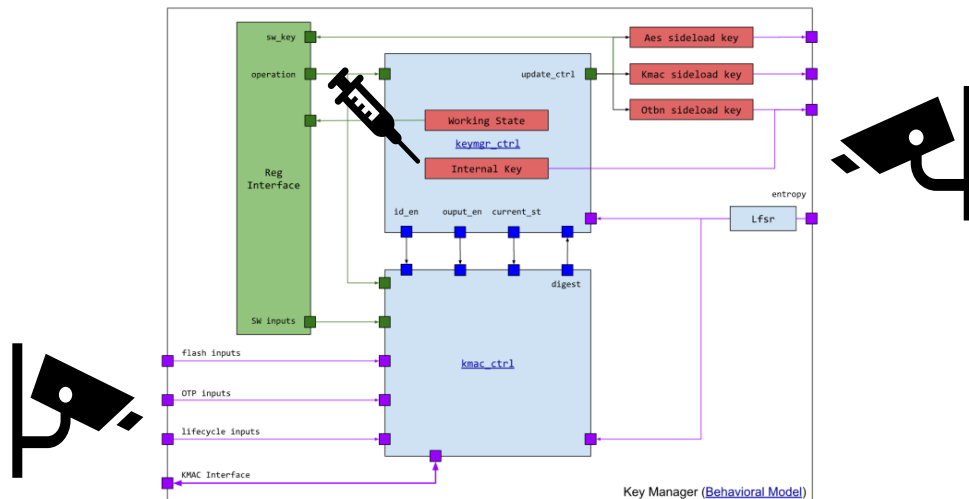
Key Verification with Security Path Verification (SPV)

CWE	Description	Weakness
1263	Improper Physical Access Control	Data Confidentiality and integrity
1282	Assumed-Immutable Data is Stored in Writable Memory	Data integrity
1258	Exposure of Sensitive System Information Due to Uncleared Debug Information	Data confidentiality
1330	Remnant Data Readable after Memory Erase	

- Internal key is maintained inside of the keymgr_ctrl block

Confidentiality (leakage) – inject taint (unique tag) at the key and look for propagation at block outputs

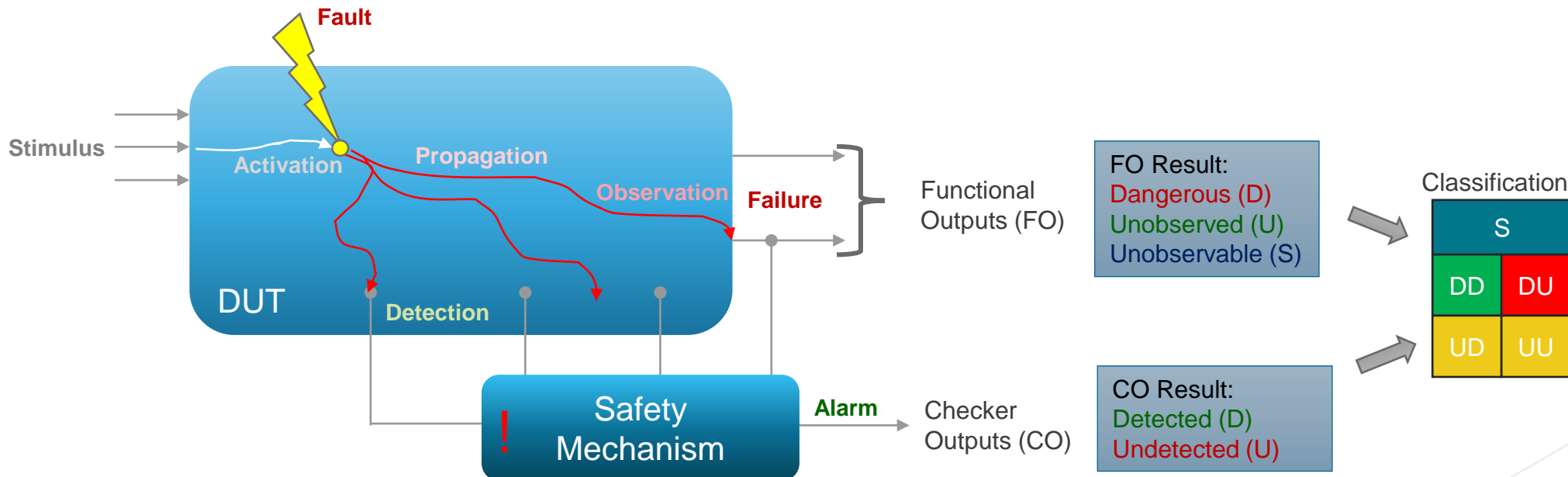
Integrity (corruption) – inject taint at the inputs and look for propagation to key



Formal Safety Verification Exposes Security Vulnerability Analysis

CWE	Description	Formal Application
1261	Improper Handling of Single Even Upsets	Formal Safety Verification (FSV)

- Check ability to detect or eliminate hacker attacks in secure subsystems
 - IP is protected against hacker attacks
 - by sensors and checkers - attack raises alarm
 - by error correction mechanism - attack is eliminated
 - Goal is to detect or correct all attacks



Negative Testing Exposes Illegal Scenarios Using PSS*

PSS

```

action generate_key {
...
share sml_core_r core;
output sml_data_buff token_out;

// API Parameters
rand crypto_algo_e algo;
rand bool extractable;
rand crypto_key_usage_e key_usage;

constraint crypto_mem_blks {
!allow_illegal_mem ->
token_out.mem_seg.mem_block in [MEM0, MEM1];
}

rand bool allow_illegal_mem;

constraint default allow_illegal_mem == false;
}
    
```

LEGAL Spec – Crypto accesses only MEM0 and MEM1

Attribute to allow illegal memories

By default, allow only legal values

Concept is applicable to any SoC resources, state variables, protocol/memory interfaces, etc.

PSS

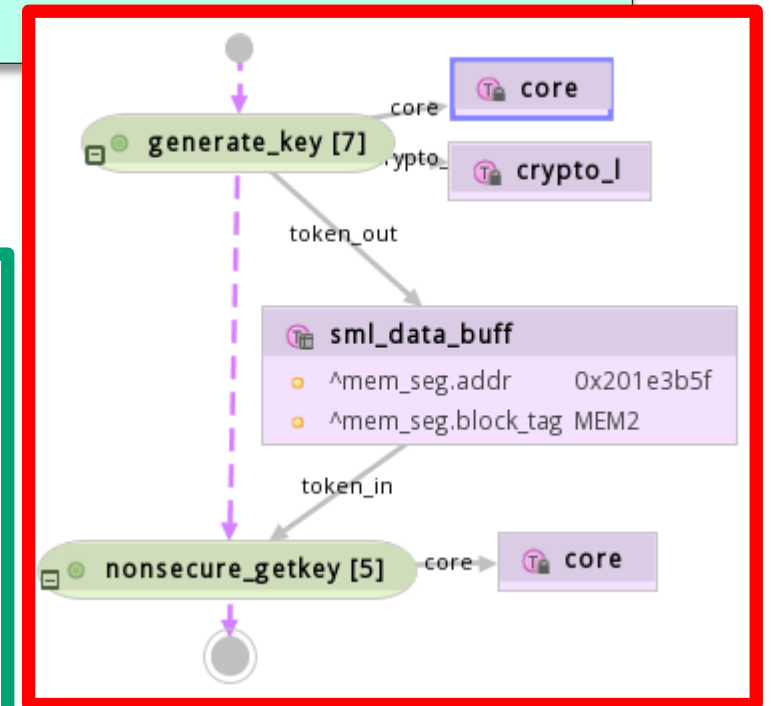
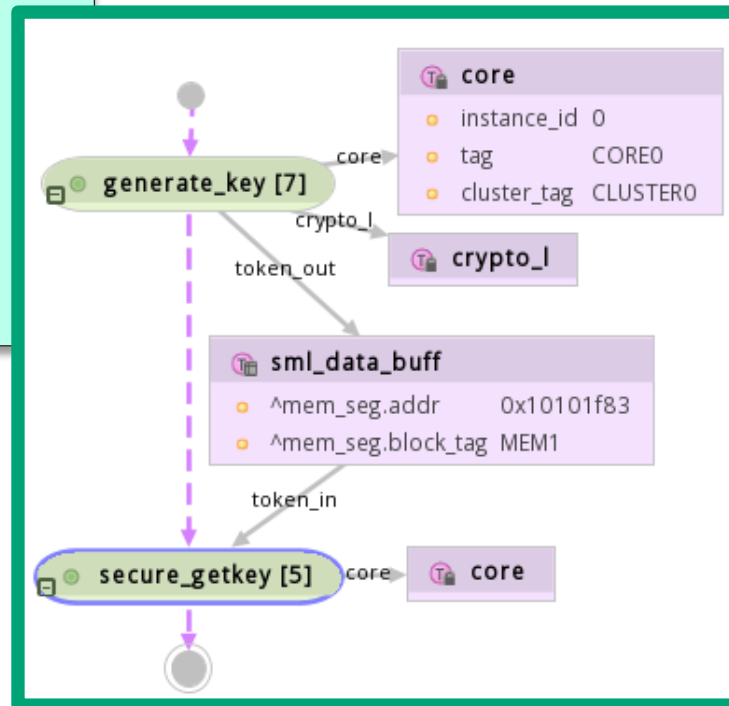
```

action crypto_illegal_mem_test {
activity {
do crypto_c::generate_key with {
default disable allow_illegal_mem;
allow_illegal_mem != false;
token_in.mem_seg.mem_block == MEM2;
}

do security_mod_c::nonsecure_getkey;
}
}
    
```

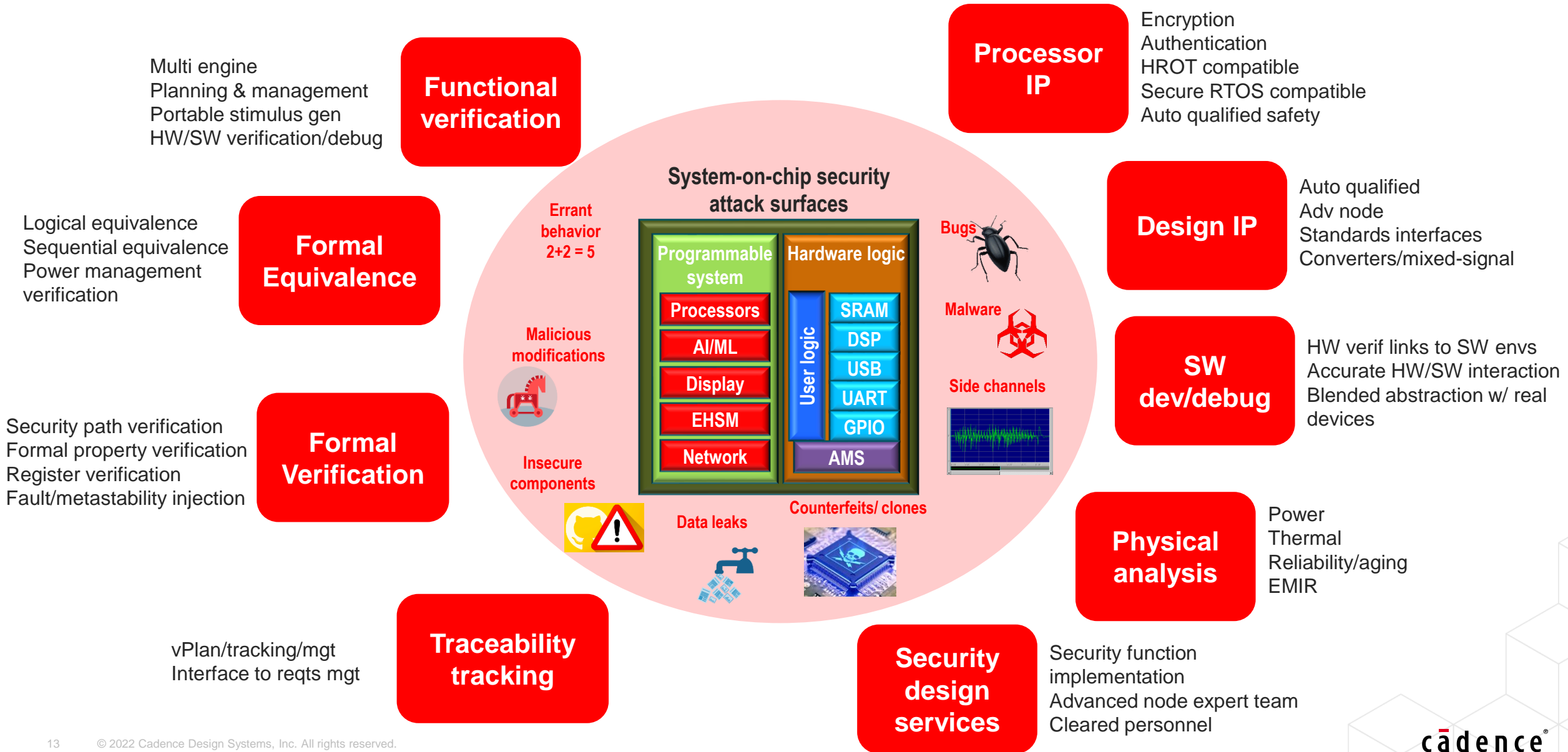
ILLEGAL – Scenario to force crypto to write keys to illegal memory

Disable default (legal) rule



*PSS = Portable Stimulus Standard (Accellera) defines stimulus usable in simulation, emulation, and ASIC/FPGA

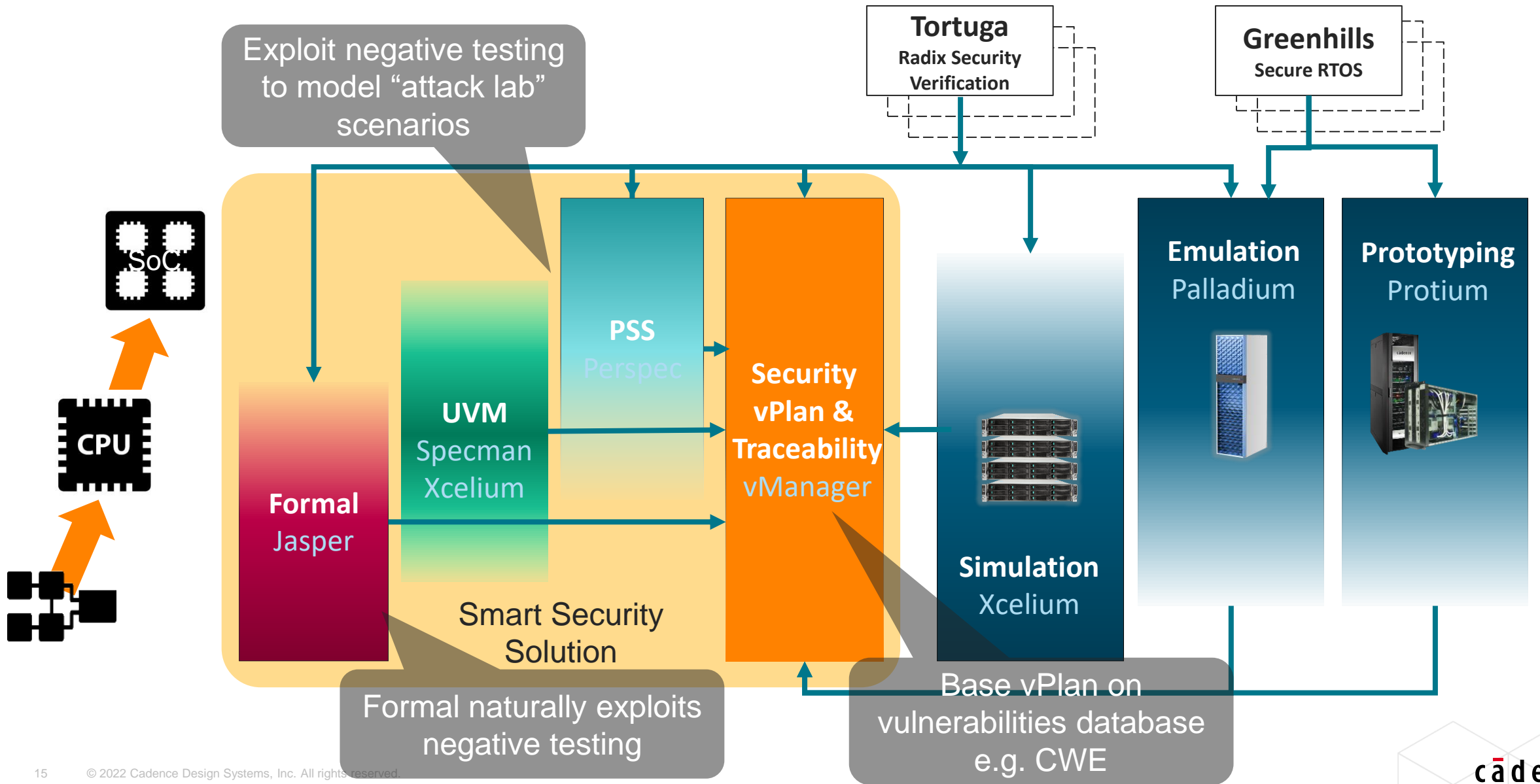
Broad Array of Security Solutions is Needed



Security Call to Action

- Assess current functional and security verification methodology
 - More comprehensive verification reduces security risk
- Discuss how security levels of assurance are applied on projects
 - Set security objectives
- Execute additional security verification
 - Discuss additional security verification goals
 - Identify joint engineering team to bring-up and apply new tools/methodology
 - Measure results from application of new tools/methodology
 - Document results and support materials/training to bring-up new projects

Cadence Security Verification Solution and Partners





cādence®

© 2022 Cadence Design Systems, Inc. All rights reserved worldwide. Cadence, the Cadence logo, and the other Cadence marks found at <https://www.cadence.com/go/trademarks> are trademarks or registered trademarks of Cadence Design Systems, Inc. Accellera and SystemC are trademarks of Accellera Systems Initiative Inc. All Arm products are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All MIPI specifications are registered trademarks or service marks owned by MIPI Alliance. All PCI-SIG specifications are registered trademarks or trademarks of PCI-SIG. All other trademarks are the property of their respective owners.