2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

System-Level Security Verification Starts with the Hardware Root of Trust

Dr. Jason Oberg
Co-founder and CEO
DVCon 2019
jason@tortugalogic.com

accellera SYSTEMS INITIATIVE

Software-only based security is not enough

Connectivity

At Least Three Billion Computer Chips Have the Spectre Security Hole

Companies are rushing out software fixes for Chipmageddon.

THE SPECTRE AND MELTDOWN SERVER TAX BILL

January 8, 2018 Timothy Prickett Morgan

AMD is big winner from chip flaw fiasco as more than $11 billion in Intel stock value is wiped out

Coretek: Hospitals Avoiding Spectre, Meltdown Patches Because of Performance Hit, High Cost Of Adding New Hardware

The top health care solution provider said hospitals are facing multi-million dollar hardware costs and a 40 percent application workload hit if they implement the patches.

By Steven Burke February 07, 2018, 07:00 AM EST

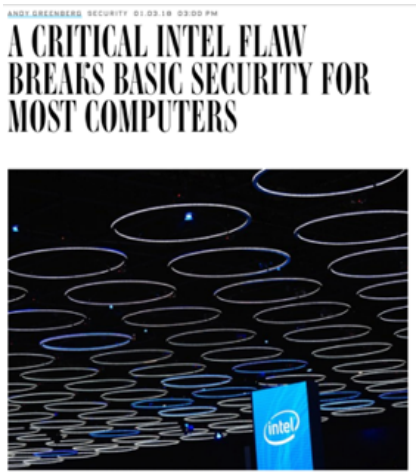
Hardware Vulnerabilities are Widespread, Dangerous, and Costly

Hardware vulnerabilities cause system-level exploits

These hardware vulnerabilities have:

- Exposed intellectual property
- Leaked customer information, violating vendor trust
- Broken system integrity, causing harmful malfunction
- Cost billions in recalls, market cap, and brand damage

July 2017

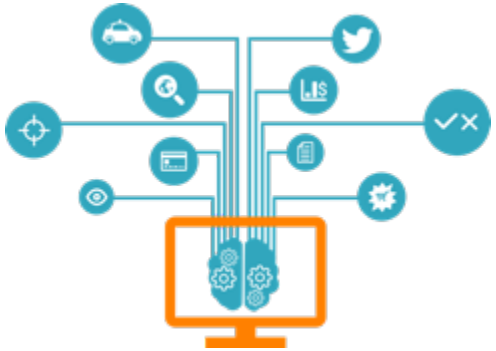


January 2018

April 2018

Hardware Manages Security For Many Markets

AI / Machine learning



Datacenter

Aerospace/Defense

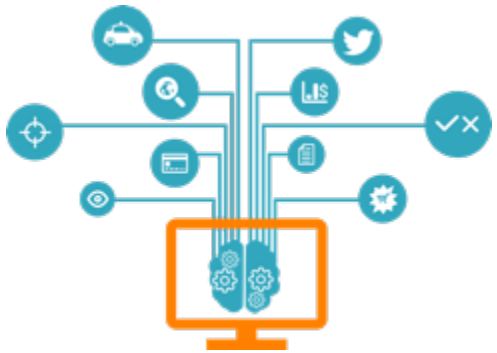
IoT Edge

Automotive

Making security a unique challenge as attacks/threats vary based on the market

Hardware Manages Security For Many Markets

AI / Machine learning



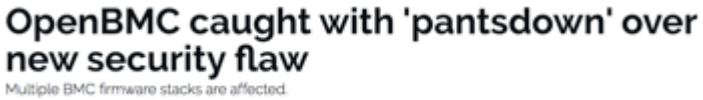
Vulnerabilities:
Changing of weights and other meta data to cause misclassification

Hardware Attacks:
- Firmware extraction and device re-flashing
- External debug access

Datacenter

Vulnerabilities:
Infrastructure vulnerabilities to allow unauthorized access to information

Hardware Attacks:
Baseboard management controller (BMC) compromises allowing malware execution, firmware flashing, or bricking it entirely



Aerospace/Defense

Vulnerabilities:
1. Trusted microelectronics
2. Theft of mission information
3. Mission critical malfunction

Hardware Attacks:
- Hardware trojans
- FIB attacks to connect eFuses
- Fault injection
- Clock glitching
- Differential Power Analysis (DPA)
- Firmware extraction and device re-flashing
- External debug access

Hardware Manages Security For Many Markets

IoT Edge

Vulnerabilities:

1. Unauthorized access to customer data
2. Harmful device malfunction
3. Theft of intellectual property

Hardware Attacks:

- Fault injection
- Clock glitching
- Differential Power Analysis (DPA)
- Firmware extraction and device re-flashing
- External debug access

Zero-day Chip Flaws Expose Devices To Attacks

November 2, 2018 by Stephen Veith

Automotive

Vulnerabilities:

1. Disabling essential driver assist features
2. Fob hacking
3. Remote access to safety features
4. Stolen personal information

Hardware Attacks:

- Fault injection
- Clock glitching
- Differential Power Analysis (DPA)
- Firmware extraction and device re-flashing
- External debug access

Experts say car key fob hacks could become the latest theft tool for cyber-savvy car thieves

Recent reports have raised concerns about ability for a vehicle hacks.

Example Vulnerabilities – AI/ML

Booming area for the semiconductor industry and will introduce new attacks on hardware

- Really good reference here: https://elie.net/blog/ai/attacks-against-machine-learning-an-overview/

1. Adversarial inputs:
   Injecting inputs into the system with intention of them being misclassified

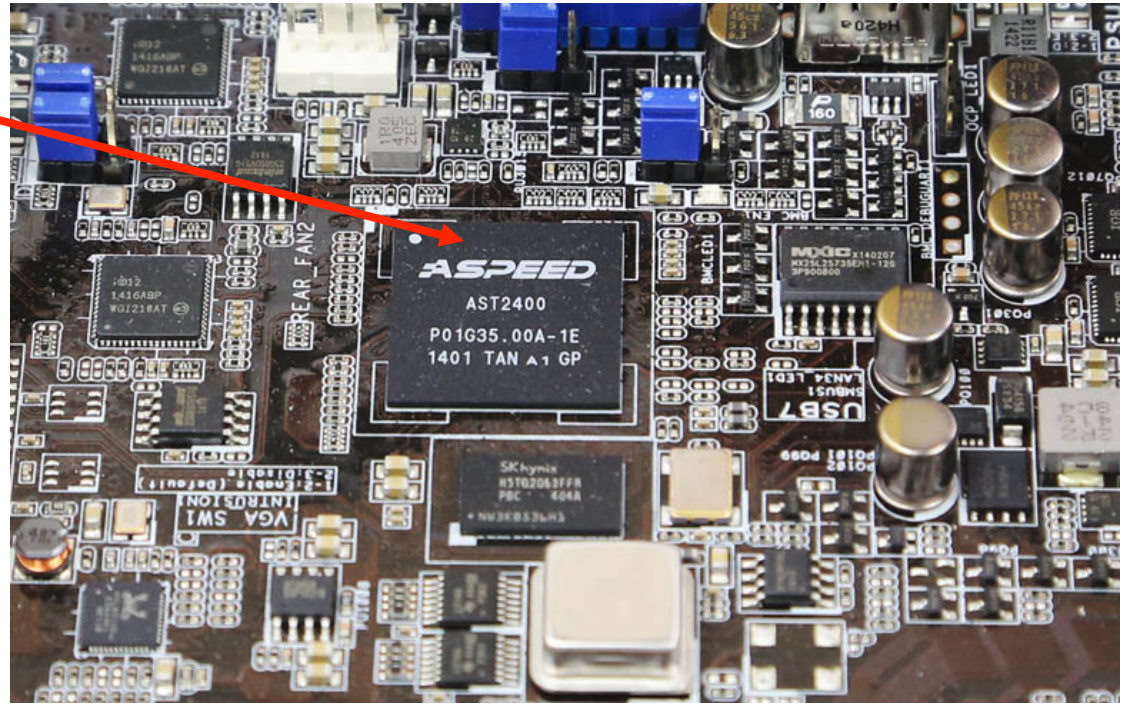
2. Data poisoning:
   Malicious changes to training data to produce invalid and unsafe results

3. Model stealing:
   Copying proprietary training data to get a market advantage

All such assets will be stored on modern hardware and will need to be protected

Example Vulnerabilities - Datacenter

- **Baseboard Management Controller (BMC):** server motherboard component allowing remote monitoring and maintenance of hardware
- BMC can power cycle system, update or re-install the OS etc.
- Enables server admin to perform tasks previously requiring physical access to the server



https://www.servethehome.com/explaining-the-baseboard-management-controller-or-bmc-in-servers/

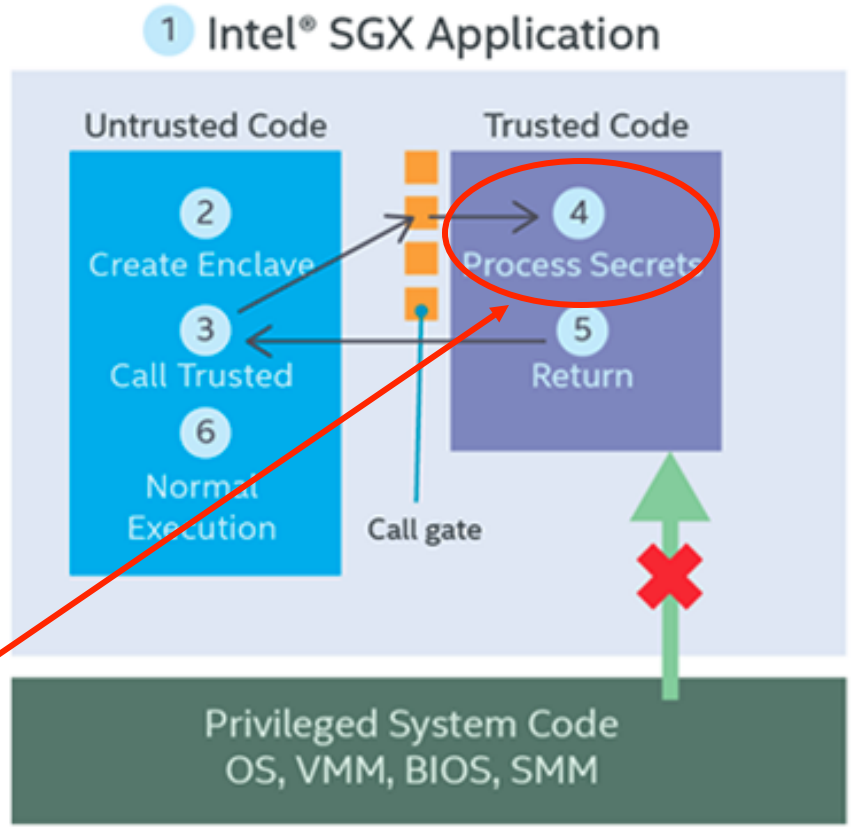
Example Vulnerabilities - Datacenter

- Aspeed ast2400 and ast2500 BMC chips connect to host via Advanced High-performance Bus (AHB) bridges
- Private physical memory and I/O ports exposed directly through AHB interface with **no authentication or access control mechanisms**
- Malware on the host processor with root privileges or rogue server administrator can:
  - Completely replace the BMC firmware with malware
  - Dump BMC firmware to host processor
  - Continuously power cycle the machine or brick the BMC and entire server

https://www.theregister.co.uk/2019/01/24/bmc_pantsdown_bug/
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-6260

Example Vulnerabilities - Datacenter

- Datacenter hardware shared by many customers, OS may not be trusted
- **Secure Enclaves:** execution environments with strong hardware-enforced security guarantees (ex. Intel SGX)
  - Isolation and protection from all other processes, including the operating system
- **Problem:** Secrets processed within SGX enclaves recoverable by the Foreshadow attack



https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

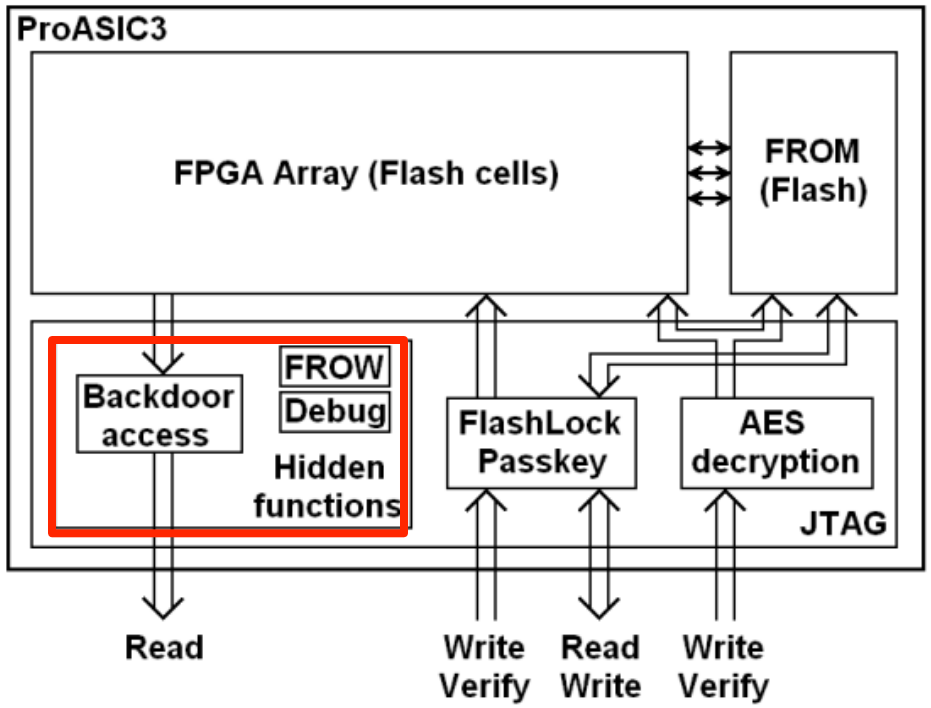
Example Vulnerabilities - Datacenter

- **Foreshadow Attack:** speculative execution attack (similar to Meltdown) targeting enclave memory
  - Malicious process faults when accessing enclave memory but can execute transient instructions before fault is generated (tip race condition in attacker's favor by marking enclave memory page as "not present")
  - Load data into cache locations based on secret enclave data and recover using cache access timing side channel
- Foreshadow used to extract remote attestation keys, effectively allowing arbitrary enclaves to be vetted as trustworthy by the Intel remote attestation ecosystem

Van Bulck, Jo, et al. "Foreshadow: Extracting the Keys to the Intel {SGX} Kingdom with Transient Out-of-Order Execution." USENIX Security. 2018.

Example Vulnerabilities – Aerospace/Defense

- Security and trustworthiness of commercial off the shelf (COTS) digital components integrated in military applications is critical
- Microsemi military grade FPGA bitstream read/overwritten via undocumented JTAG commands
  - Intentional or accidental backdoor?
  - Regardless, protecting HW debug infrastructure critical

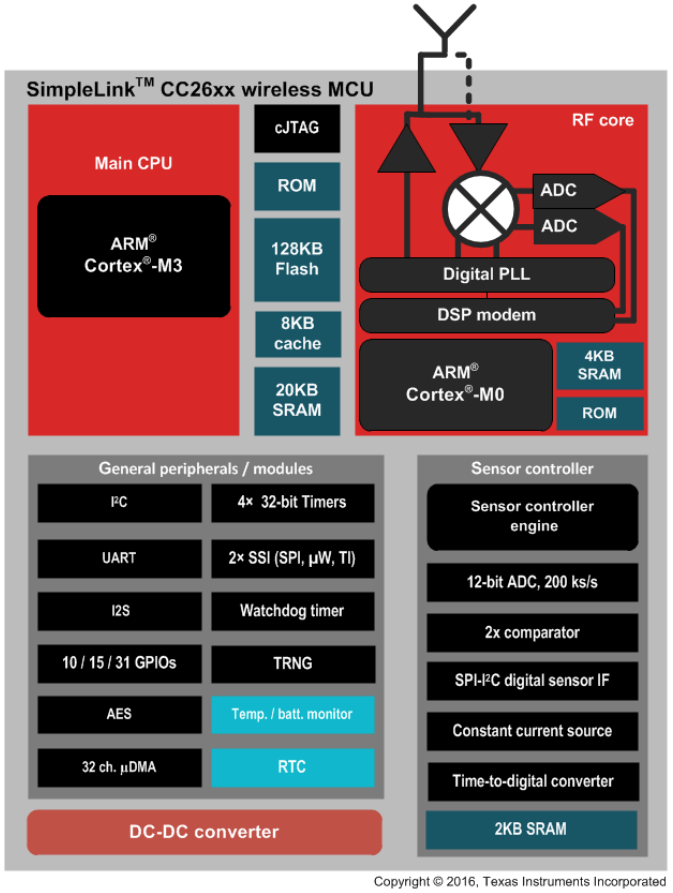


S. Skorobogatov and C. Woods, “Breakthrough Silicon Scanning Discovers Backdoor in Military Chip.” in CHES, LNCS, E. Prouff and P. Schaumont, Eds., vol. 7428. Springer, 2012, pp. 23–40.

Example Vulnerabilities – IoT

- **Bluetooth Low Energy (BLE)** common in medical devices and network access points
- Texas Instruments (TI) multi-protocol wireless SoC chips integrated into Cisco, Meraki, and Aruba wireless access points
- BLE “advertising” packets parsed by TI firmware and reside in chip memory
- BleedingBit Vulnerability: malicious advertising packets can overflow the software stack allowing an attacker to gain control of the chip

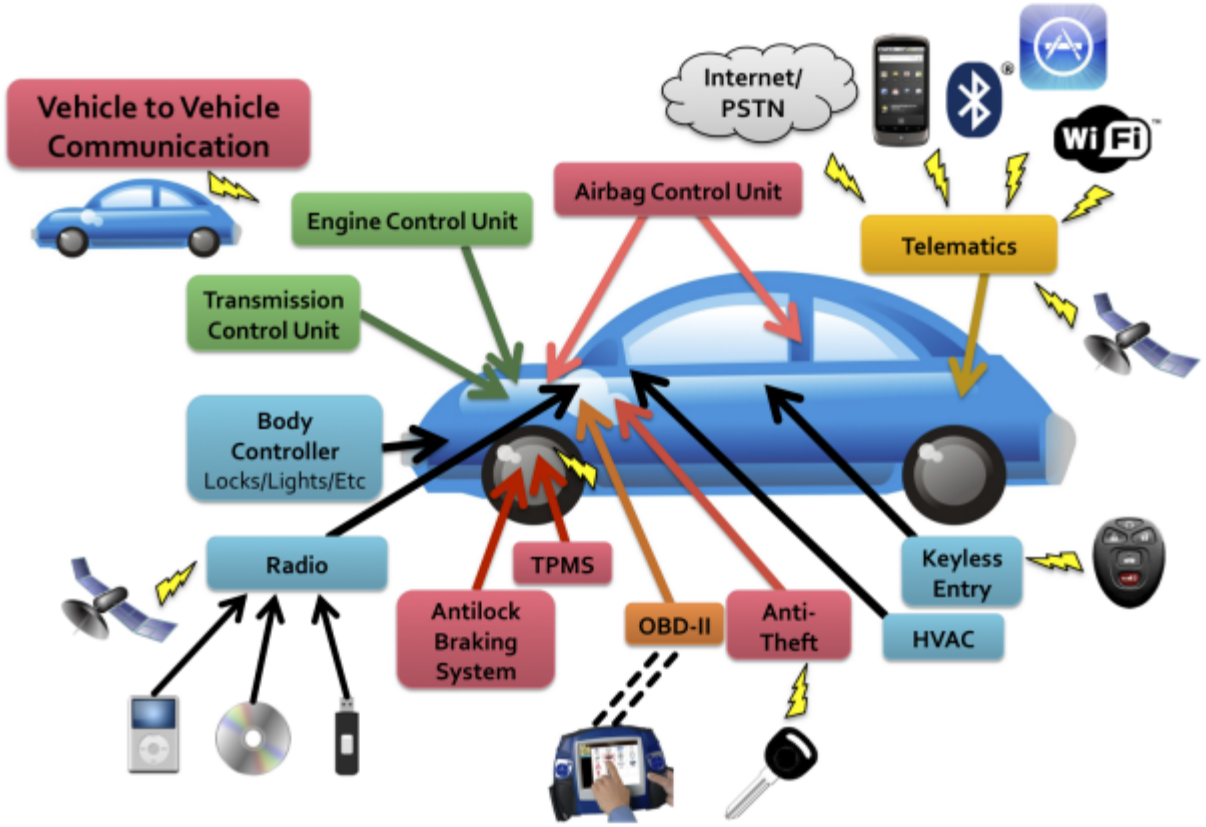
https://go.armis.com/bleedingbit



http://www.ti.com/general/docs/datasheetdiagram.tsp?genericPartNumber=CC2640&diagramId=SWRS176B

Example Vulnerabilities – Automotive

- Modern cars control safety-critical functionality with digital components
- **Problem:** safety-critical components connected via CAN Bus to infotainment and communication systems with internet connectivity; all reliant on their hardware for security



Checkoway, Stephen, et al. "Comprehensive experimental analyses of automotive attack surfaces." USENIX Security Symposium. 2011.

Example Vulnerabilities – Automotive

- In 2014 security researchers **remotely** disable the brakes in a Jeep Cherokee
- Attack first installs malicious firmware for entertainment hardware. **Hardware-assisted secure boot could be used to avoid this.**
- Malicious firmware capable of injecting arbitrary CAN bus commands to transmission and breaking systems

Hardware-based security required to reduce the likelihood of safety-critical vulnerability exploitation. Security begins with hardware.

https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Hardware complexity increasing number of exploitable vulnerabilities

- Higher demands for hardware acceleration increasing complexity and *decreasing* security
- More complexity means more threat vectors and exploitable vulnerabilities
- Makes reasoning about security of the system complex, time consuming, and often overwhelming
- Traditionally security burden did not fall on companies building hardware, now initiatives for hardware to provide the “Root of Trust”

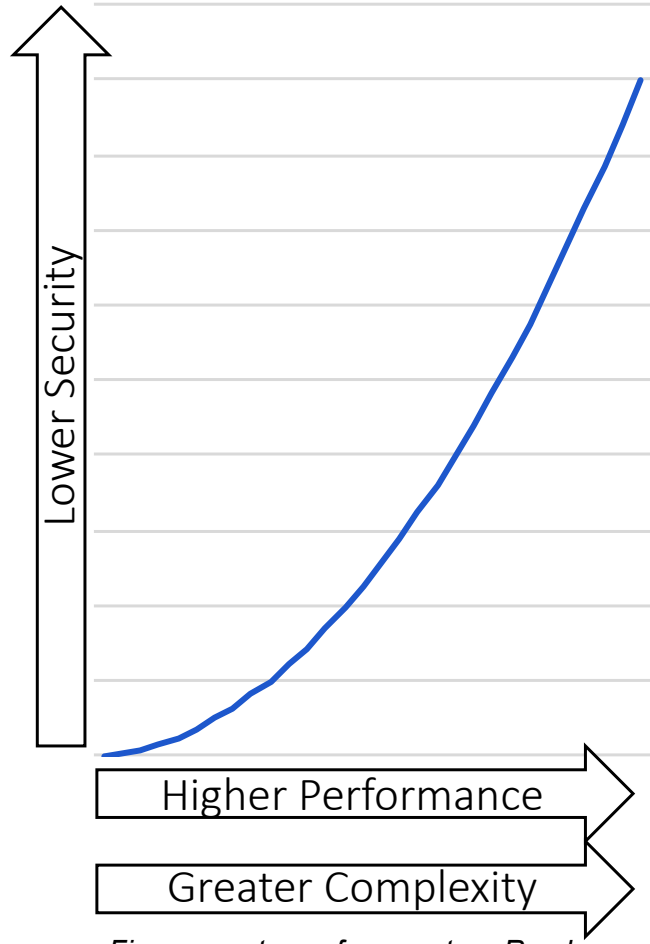


Figure courtesy of our partner Rambus

Hardware Roots of Trust Fundamental for Security

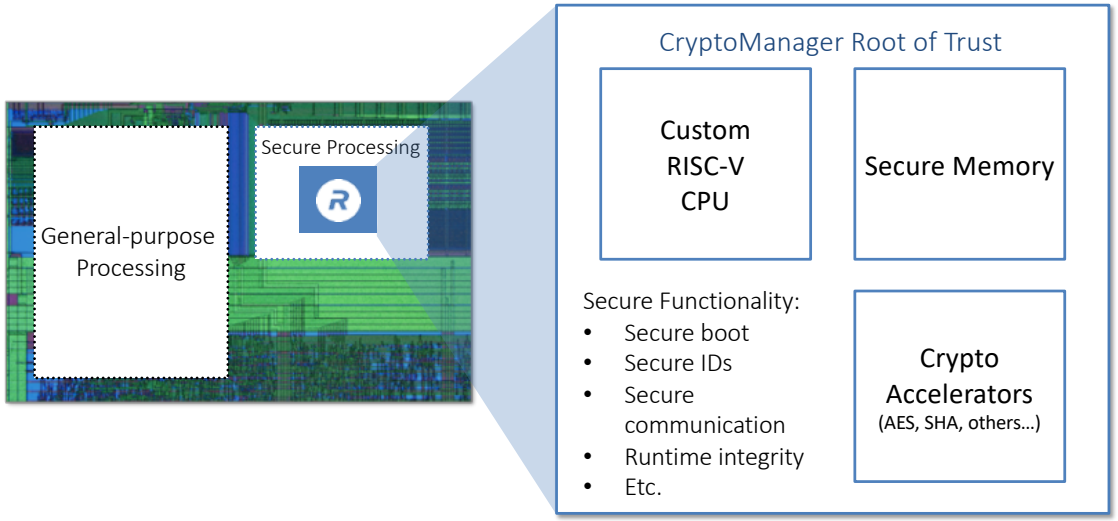
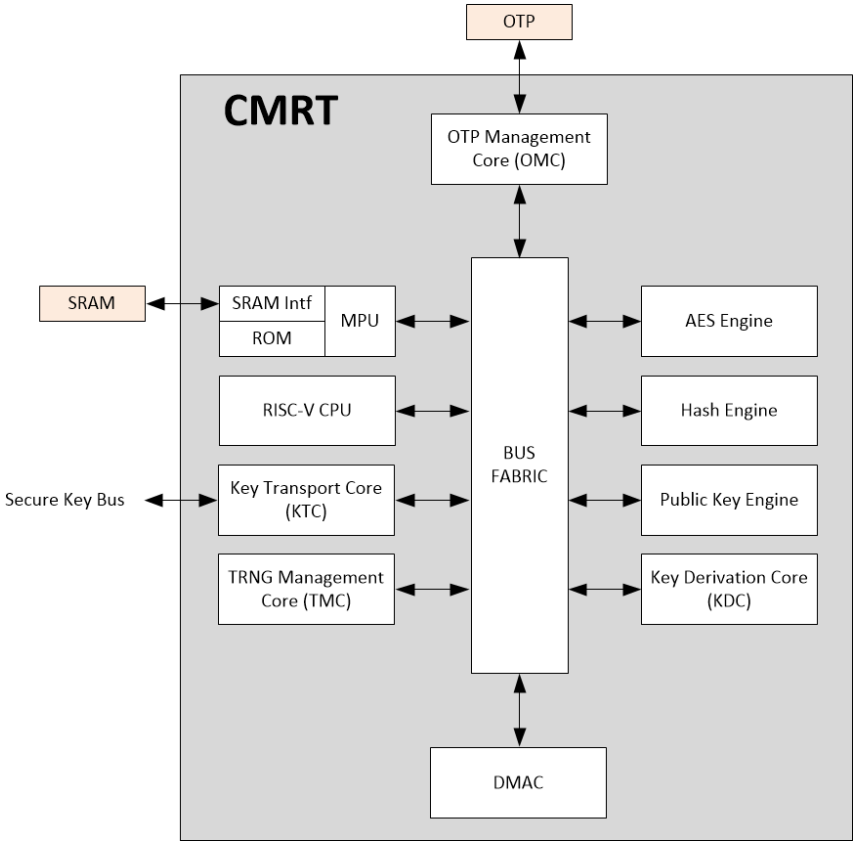


Figure courtesy of our partner Rambus

- Industry shift to utilize Hardware as the Roots of Trust (HRoT)
  - A hardware subsystem focused exclusively on security to reduce complexity and attack surface.

- These systems perform the following:
  1. Facilitate a secure boot
  2. Store secure IDs
  3. Accelerate and isolate cryptographic functions
  4. Protect cryptographic keys
  5. Store and protect customer access
  6. Runtime integrity checks

More Security Features in Hardware Makes Security Validation Challenging



Simplified CMRT RT360 diagram courtesy of Rambus

A HRoT is a powerful *security feature* but must be made a *secure feature*

Many reasons HRoTs can produce system vulnerabilities

- Security of specific states not considered
- Indirect leakages of information
- System architecture and configuration mistakes

<u>Example issues:</u>

- Mismanagement of assets during a secure boot
- Memory protection unit (MPU) misconfiguration
- Improper clearing of keys or configuration registers
- Debug modes not properly configured/disabled
- Crypto key leakage (either directly or side channels)

More Security Features in Hardware Makes Security Validation Challenging

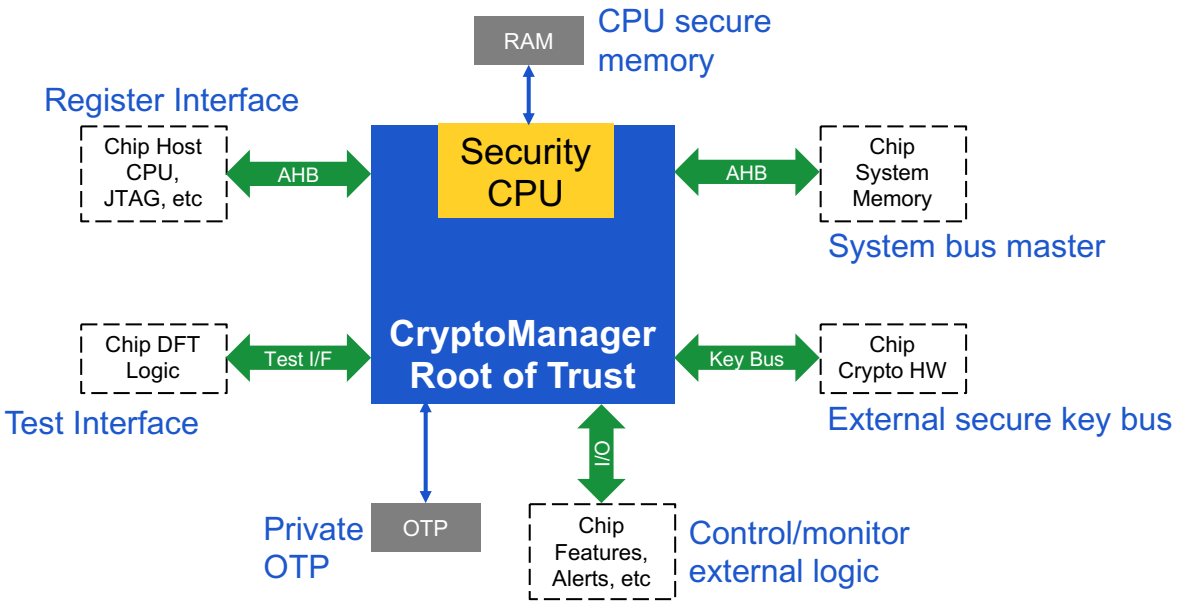


Figure courtesy of our partner Rambus

Beyond just validation of a HRoT itself, integration into larger systems introduces another big challenge

Example issues:

- Proper management of system debug and test interfaces
- Misconfigured access control to system bus master
- External (to the HRoT) NVM key storage
- Software misconfigurations of secure CPU
- Unauthorized system access to OTP
- System software bugs
- Vulnerabilities cross hardware AND software

Existing Security Approaches Don't Solve the Problem

- Architecture design review
  - Manual
  - Rarely looks are real system-level implementation
- Manual Red/Blue teaming
  - Important part of analyzing security of final product
  - Does not find security vulnerabilities pre-silicon
  - Difficult to measure/assess
- Formal verification
  - Good as part of a bigger security strategy
  - Does not scale to large design w/ HW+SW
- Directed functional tests
  - High effort to develop stimulus to target known vulnerabilities
  - Rarely finds new vulnerabilities

Tortuga Logic's products are scalable, low overhead, and finds unknown vulnerabilities while running hardware AND software together

Solution:
Tortuga Logic's Root of Trust Security Verification Solutions

Prevent security vulnerabilities in the pre-silicon design and system integration of Roots of Trust.

Tortuga Logic's products provide value in:

1. Identification of security vulnerabilities in ***HW/SW system architecture***

2. Identification of ***implementation errors*** violating security requirements

3. Tortuga Logic's *Radix-S™* Product
   - Finds unknown vulnerabilities using current functional verification efforts for Roots of Trust
   - Reduces time and effort for current security verification efforts for Roots of Trust
   - Scales to SoC system-level analysis around Roots of Trust
   - Is low effort to deploy

How to use Radix-S™

Step 1 – Define the Threat Model and attack surface

- Understand assets that are to be protected

Step 2 – Hardware design (RTL) and Threat Model are analyzed to produce our Security Model Design (SMD), a synthesizable hardware IP.

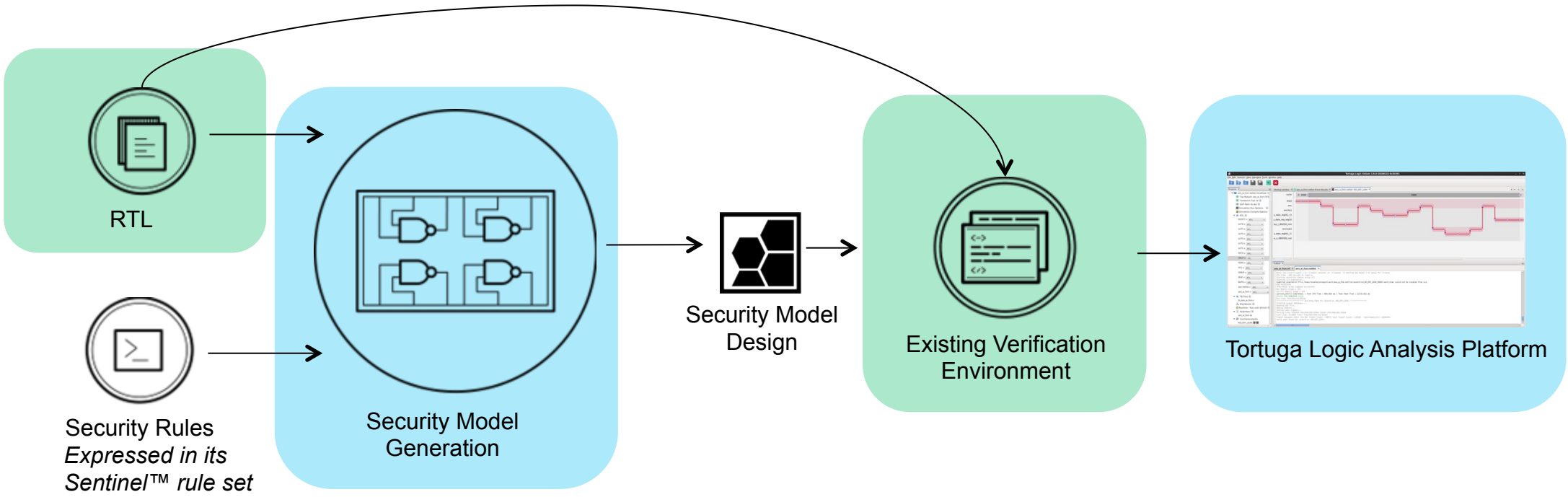
Step 3 – Run SMD in parallel with RTL inside existing functional verification environments

Step 4 – Analyze results and identify security violations

Radix-S™ User Flow

Already Exists at User

Provided by Tortuga Logic



Step 1: Express Threat Models

Step 2: Generate Security Model

Step 3: Insert Security Model into existing regressions/ emulation with original RTL

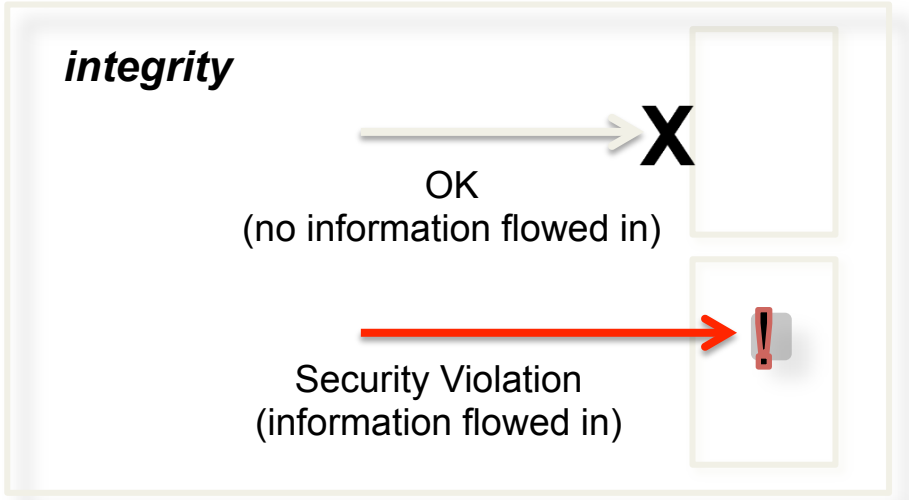
Step 4: Analyze results; identify violations based on Threat Models

Information Flow Security Properties

Ensuring integrity

Integrity is violated when an unauthorized logical (SW) or physical (HW) asset can modify a target register/memory state

```
untrusted_signals    =/=>    trusted_signals
```

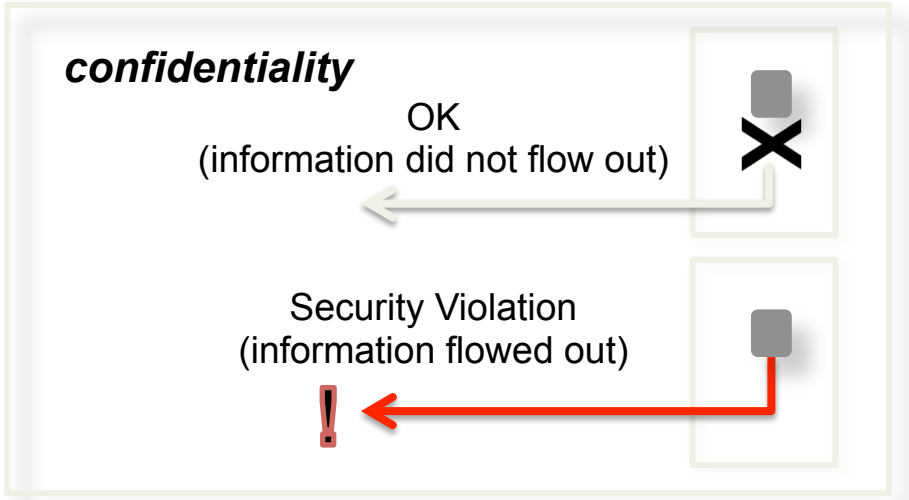


Ensuring confidentiality

Confidentiality is violated when an unauthorized logical (SW) or physical (HW) asset can read the state of a target register/memory

```
secret_signals    =/=>    output_signals
```

"source" *"destination"*

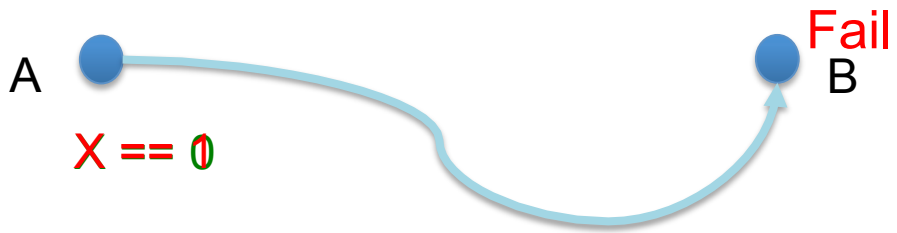


Additional Security Rules

Temporal Rules

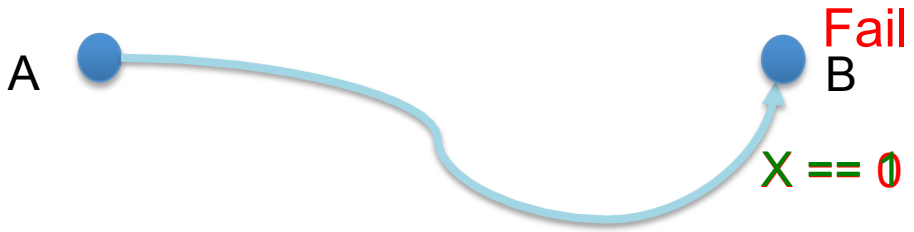
A when X =/=> B

- Starts tracking A when condition X is true and rule will **fail** if A then flows to B.



(A =/=> B) II X

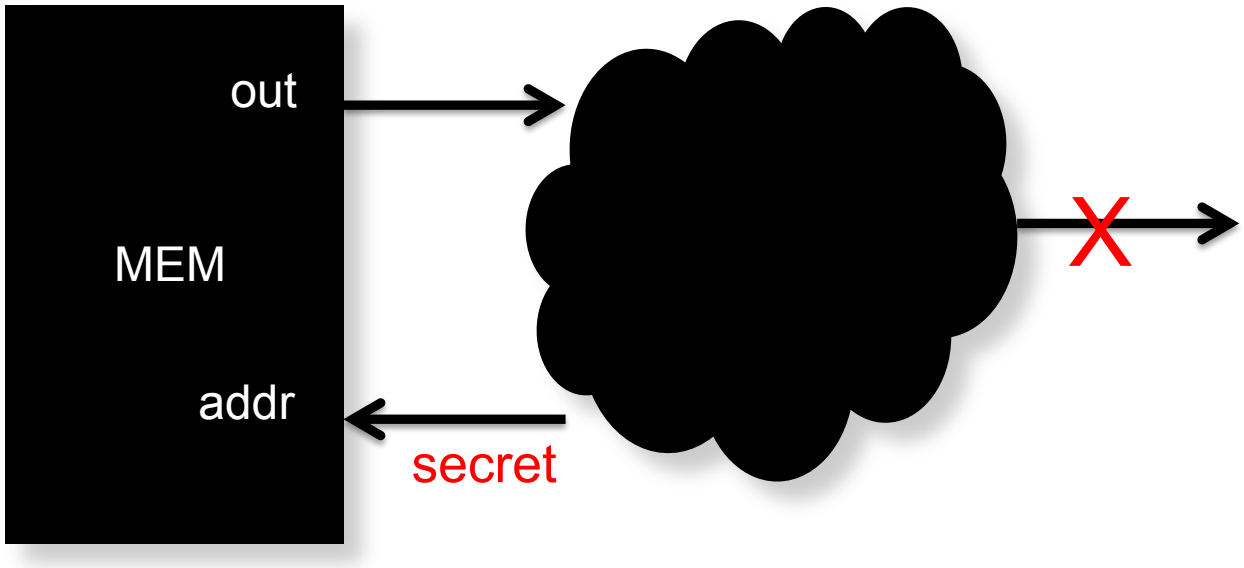
- Starts tracking A at t=0. Rule will **pass** if A doesn't flow to B or X is true.



Memory Read Protection

mem.out when (addr == secret) =/=> $all_outputs

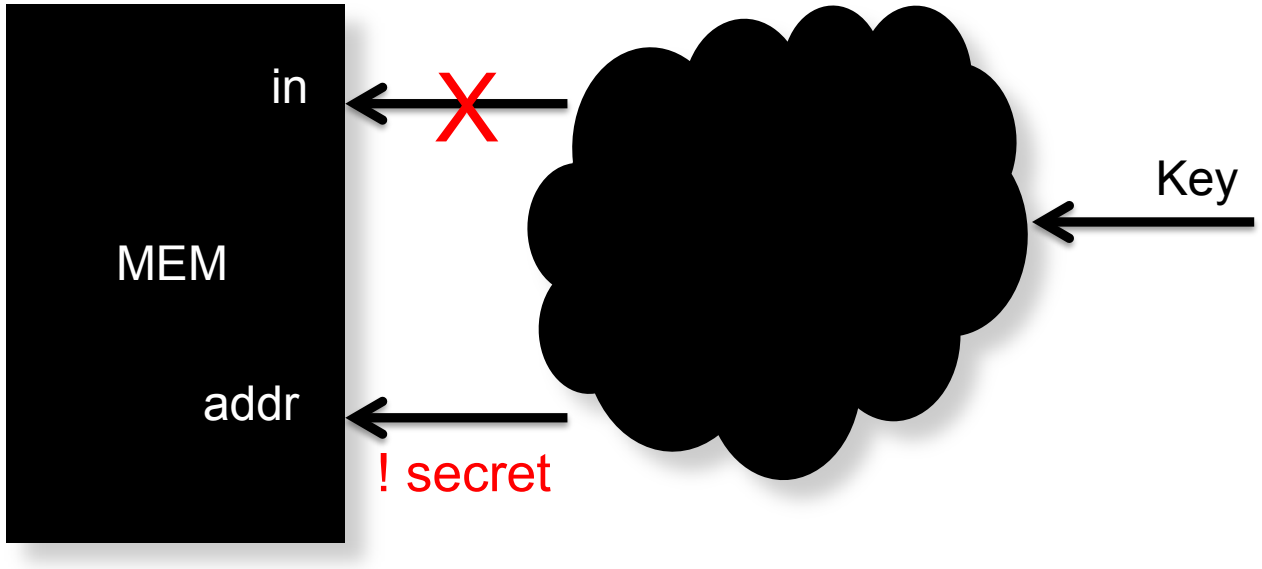
- Secret content being read out of memory should not reach any output



Memory Write Protection

(key =/=> mem.in) || addr == secret

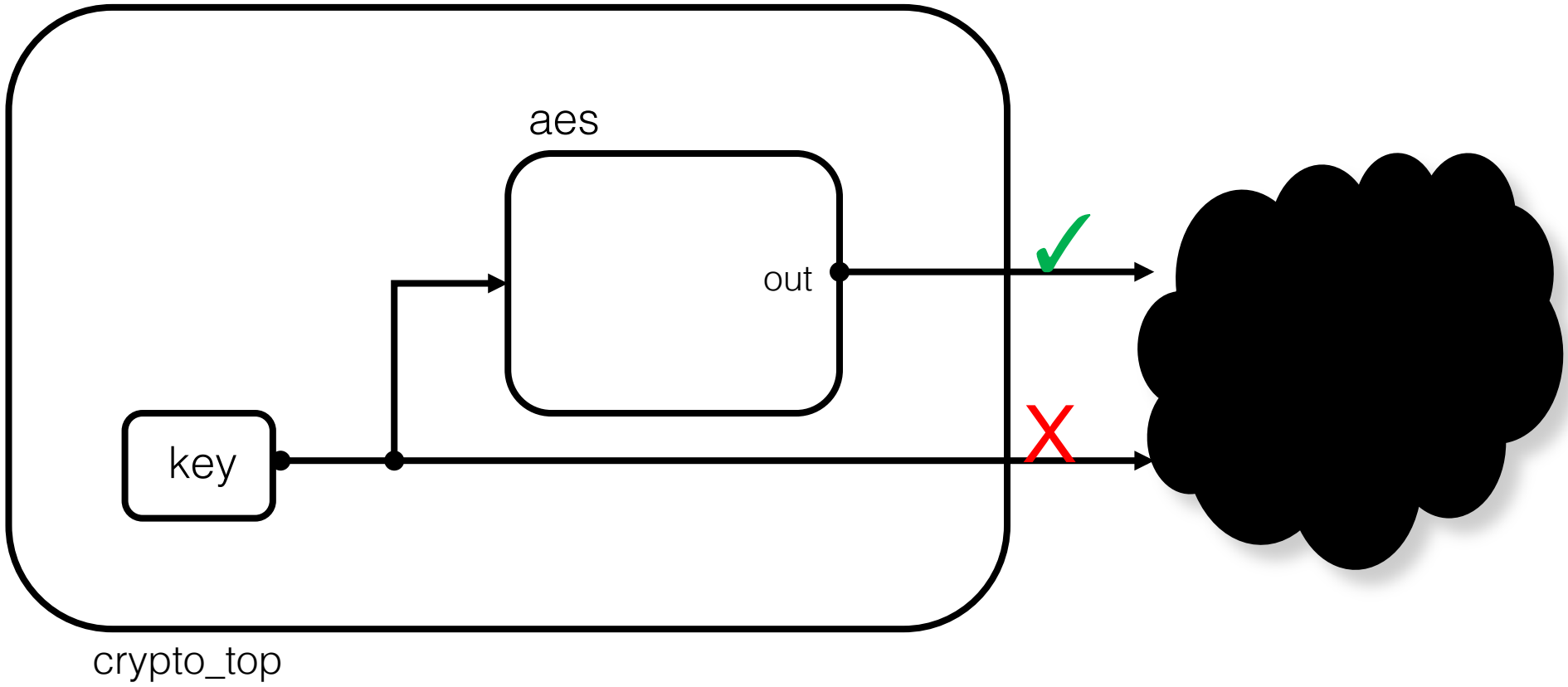
- Secret content (i.e. key) should not be written to non-secure memory region. Flow to mem.in allowed if addr == secret.



Ignoring Condition

key =/=> $all_outputs ignoring aes.out

- Specifies that aes.out is “secure.” Ignores information flow.



Additional Use Cases

- SoC Access control verification
- Secure boot analysis
- Timing side channels
- Encryption key leakage
- Configuration register read/write protection
- Memory Protection Unit (MPU) configuration
- JTAG disablement/analysis
- 3rd party/vendor IPs and interfaces

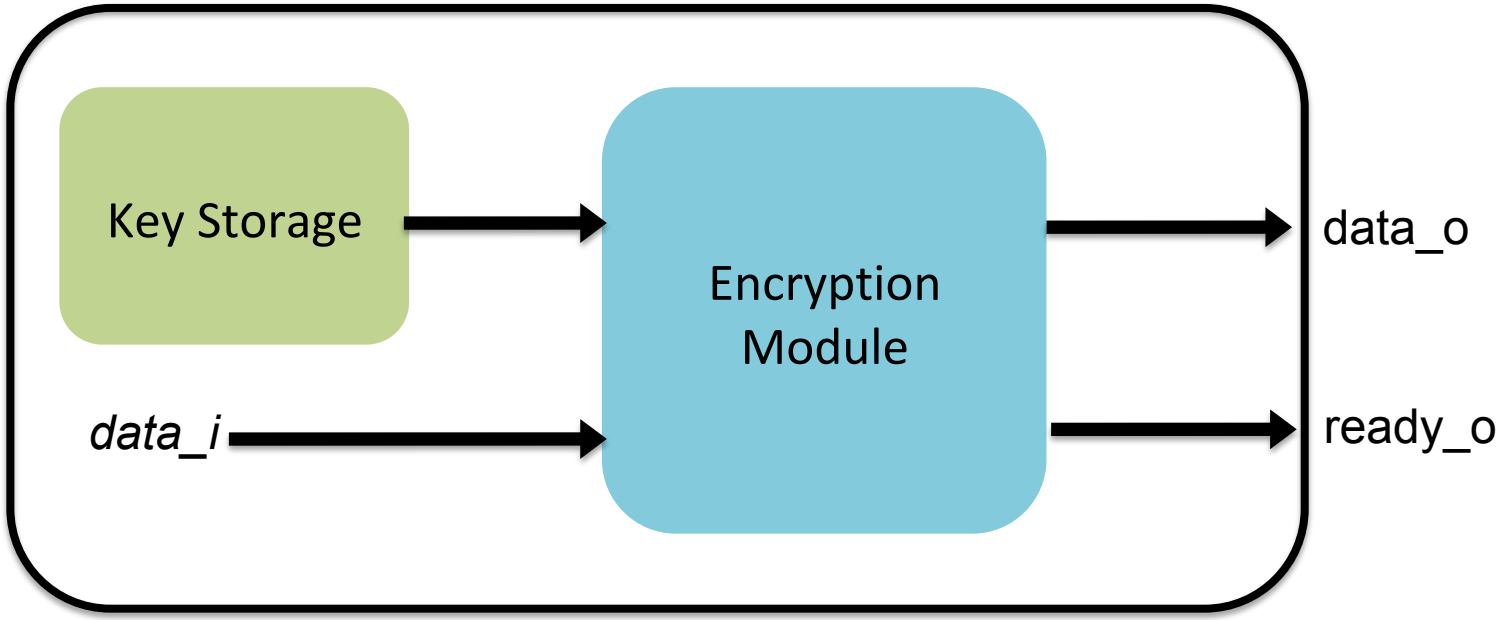
AES Demo: Key Leakage Vulnerability

- AES Encryption block retrieved unmodified from OpenCores.org
  - Fully functional. Correctly performs encryption for thousand of test vectors.
- Threat Model assumes security violated if key or data leaks to the output without being encrypted
- Tortuga Logic's Radix-S™ identifies vulnerability that causes key and data to leak out as plain text

AES Demo: Key Leakage Vulnerability

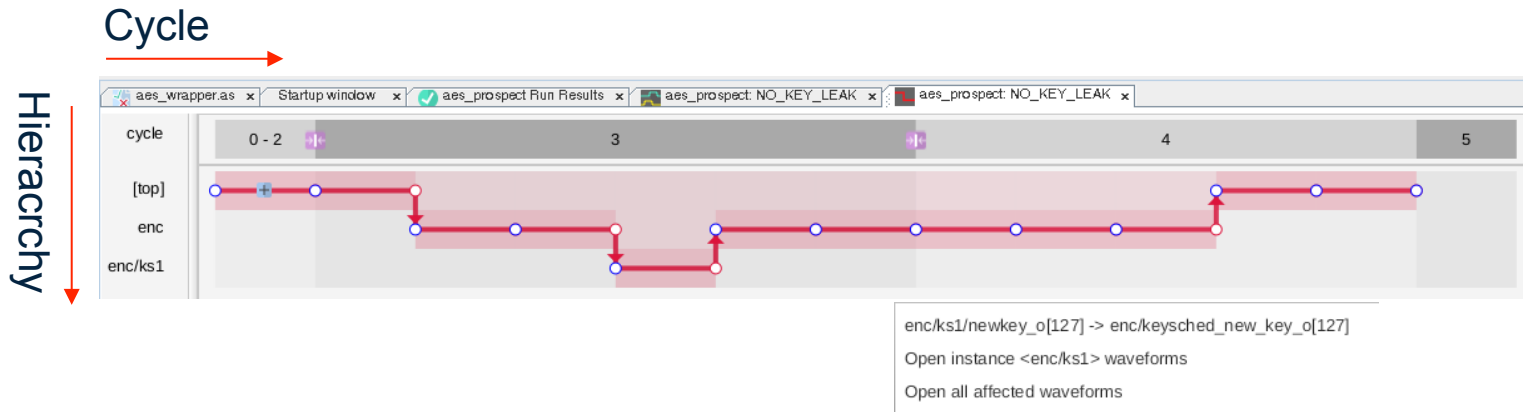
Property:
assert iflow (*key_i* =/=> *data_o*);
"key should not flow to output"

Result (demo):
Rule fails
Key XOR Data flows to data_o pin
Violates threat model



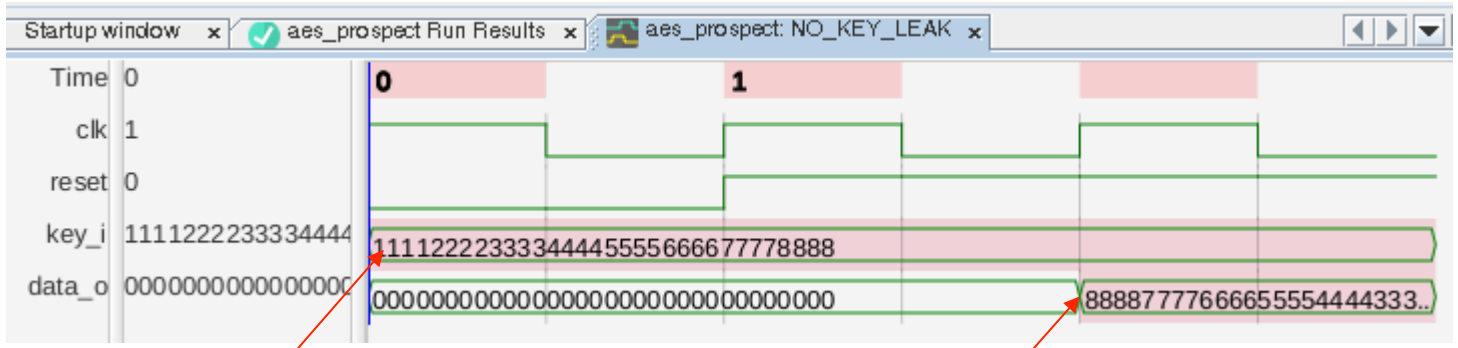
Pathview

- Enumerates a path from source to destination



- Find more details through a mouse hover on any node

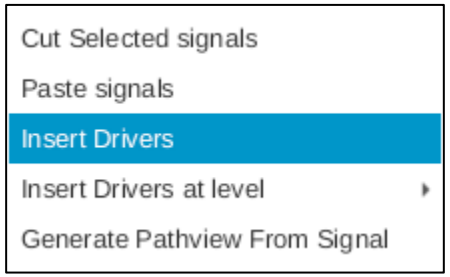
Waveform view



The ‘Red’ shade indicate that the signal is carrying ‘secured’ information

The ‘secured’ information has arrived at the destination

Right Mouse Click on any signal:



- Remove the selected signal from the waveform
- Make a copy of the signal in the display
- Display the immediate drivers of this signal
- Display all the drivers up to the user-specified level
- Display the path from the source up to this signal

Questions?

Jason@tortugalogic.com