



Formal Analysis of Security Data Paths in RTL Design

Ziyad Hanna, PhD, Chief Architect, VP of Research

Jamil R. Mazzawi, Consulting Services Manager

HVC 2012 - Haifa Verification Conference, November 8, 2012

Introduction

[For Business](#)[For Home](#)[Products](#)[Support](#)[About Intel](#)[IT Center](#)[Developer Center](#)[Partners](#)[Technology](#)[Communities](#)[Change Location](#)

The latest
security information
on Intel® products.



[Home](#) > [Security Center](#) >

SINIT Buffer Overflow Vulnerability

Intel ID:	INTEL-SA-00030
Product family:	Intel® Trusted Execution Technology
Impact of vulnerability:	Elevation of Privilege
Severity rating:	Important
Original release:	Dec 05, 2011
Last revised:	Dec 06, 2011

Summary:

When Intel® Trusted Execution Technology SINIT Authenticated Code Modules (ACMs) are susceptible to a buffer overflow issue. Intel is providing updated SINIT ACMs to mitigate this issue and microcode updates to revoke vulnerable SINIT ACMs.

Description:

When Intel® Trusted Execution Technology measured launch is invoked using affected SINIT Authenticated Code Modules (ACMs), the platform is susceptible to an OS-level exploit, which can bypass Intel® TXT compromising certain SINIT ACM functionality, including launch control policy and additionally lead to compromise of System Management Mode (SMM). To mitigate this issue, Intel is releasing updated SINIT ACMs. Additionally, microcode-based revocation of vulnerable SINIT ACMs is being made available for all affected processors.

RUB

The diagram illustrates a hardware security architecture for a network device, showing the flow of data and the internal components of the PCB board.

Top Components: A network switch and a hard drive are shown at the top, representing the physical hardware.

PCB board: The central component is a blue box labeled "PCB board". Inside, there is a grey box labeled "SRAM FPGA" and a red box labeled "E2PROM".

SRAM FPGA: Inside the SRAM FPGA, there is a 3DES-1 block. A key icon is shown next to it, indicating a key is used in the encryption process. A green arrow labeled "Power-up" points from the E2PROM to the 3DES-1 block.

E2PROM: The E2PROM is a non-volatile memory that stores the encryption key. A green arrow labeled "Power-up" points from it to the 3DES-1 block.

Attacker: An attacker is shown on the left, with a question mark and an equals sign, indicating a security vulnerability or attack scenario.

Factory Internet Firmware Update: A green arrow labeled "Factory Internet Firmware Update" points from the E2PROM to the 3DES-1 block.

Secret Keys Proprietary Algorithms IP Cores: A blue box at the top right contains the text "Secret Keys Proprietary Algorithms IP Cores".

Bitstream: A green box labeled "Bitstream" is shown below the top box. A blue arrow points from the top box to the Bitstream box.

3DES: A blue box labeled "3DES" is shown below the Bitstream box. A blue arrow points from the Bitstream box to the 3DES box. A key icon is shown next to the 3DES box.

Encrypted Bitstream: A red box labeled "Encrypted Bitstream" is shown below the 3DES box. A blue arrow points from the 3DES box to the Encrypted Bitstream box. A yellow arrow labeled "CBC" points from the Encrypted Bitstream box to the 3DES box.



JASPER
design automation

Introduction (Cont.)

Securing the System with TrustZone® Ready Program Securing *your* Digital World



Secure Services Division

The Architecture for the Digital World®

ARM®

TrustZone in 3 Steps

1. Define secure hardware architecture

- Two separate domains: normal and secure
- Extends across system
 - Processor, display, keypad, memory, clock, radios

2. Implement in silicon system on chip (SoC)

- Enforcing secure/normal separation physically

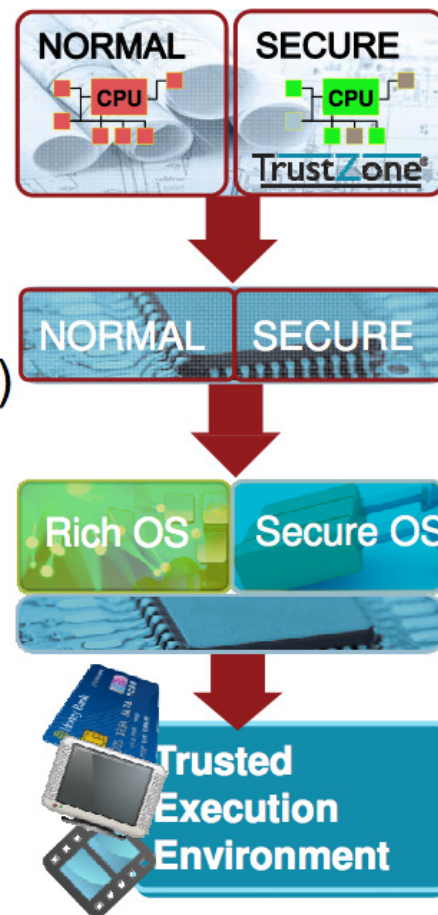
3. Combine SoC with Trusted OS

- Separate but connected to main operating system

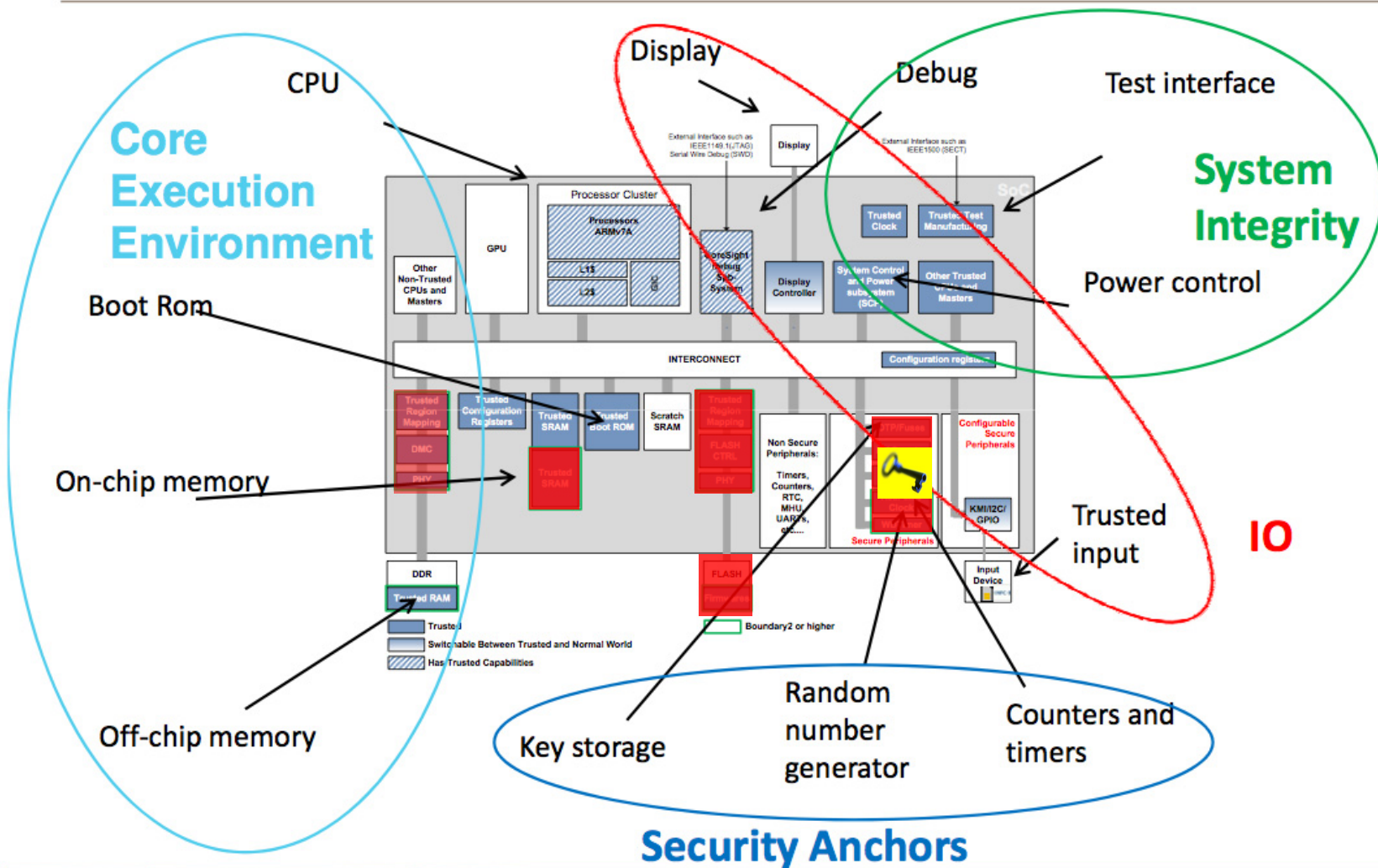
Result:

A Trusted Execution Environment (TEE)

- Ready to develop and deploy trusted services



ARM Trusted Base System Architecture



Agenda

- Why security path verification
- Security path verification overview
- Usage examples
- Conclusion

Why Security Path Verification

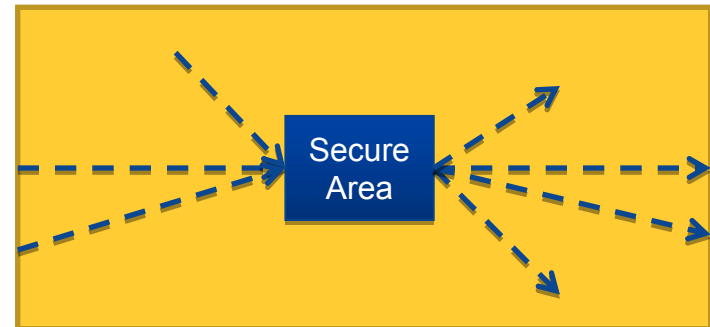
- Why do people care
 - Many systems (cell phones, game console etc.) contains secure information
 - Vulnerability will likely lead to unauthorized access of secure data
 - The potential loss (direct or indirect) is very large
- Where is the vulnerability
 - Software encryption algorithm
 - Hardware where secured areas can be accessed without going through the proper encryption paths
- Why is there hardware vulnerability
 - Integrating IP's create an unexpected paths to access secure areas
 - Addition of test-logic creates paths to the outputs

What Do People Do Today

- System Architecture
 - Moving secured areas in to isolation
 - Analyzing data-flow on the whole system
 - Manually identify potential structural paths and insert blocking conditions
- System integration
 - Investigate (often with a structural analysis tool) each structural path to ensure that they are false path
- Very tedious and ad hoc
 - Only be able to look at a small subset of trace
 - Rather subjective and no clear checking mechanism
 - Difficult to get real sense of completeness

Security Path Verification overview

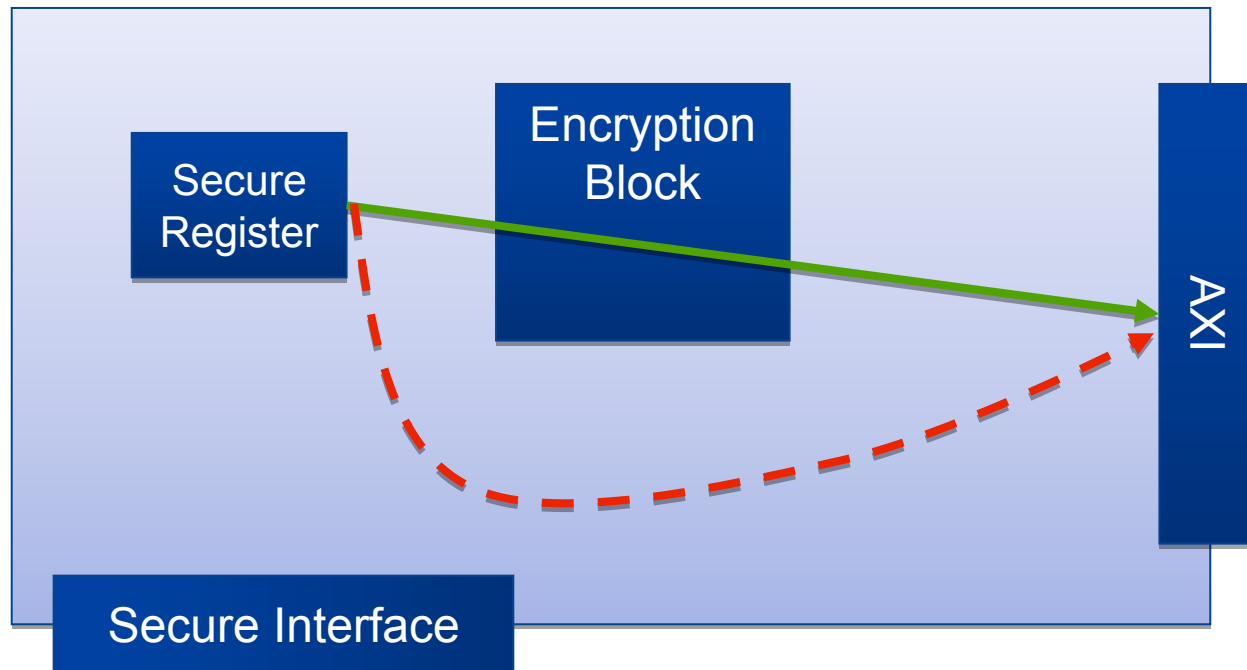
- Identify any unintentional path to/from secured areas
 - Use of Path sensitization technology (see next)
 - User specifies secure area and illegal sources and destinations for the data inside this area
 - Optional: User can specify blocks through which data is allowed to propagate
 - Waveforms show illegal propagations found



- Potential Savings
 - Time savings – weeks vs months
 - Verify all paths and able to understand progress
 - Completeness, not just subjective determination of correctness

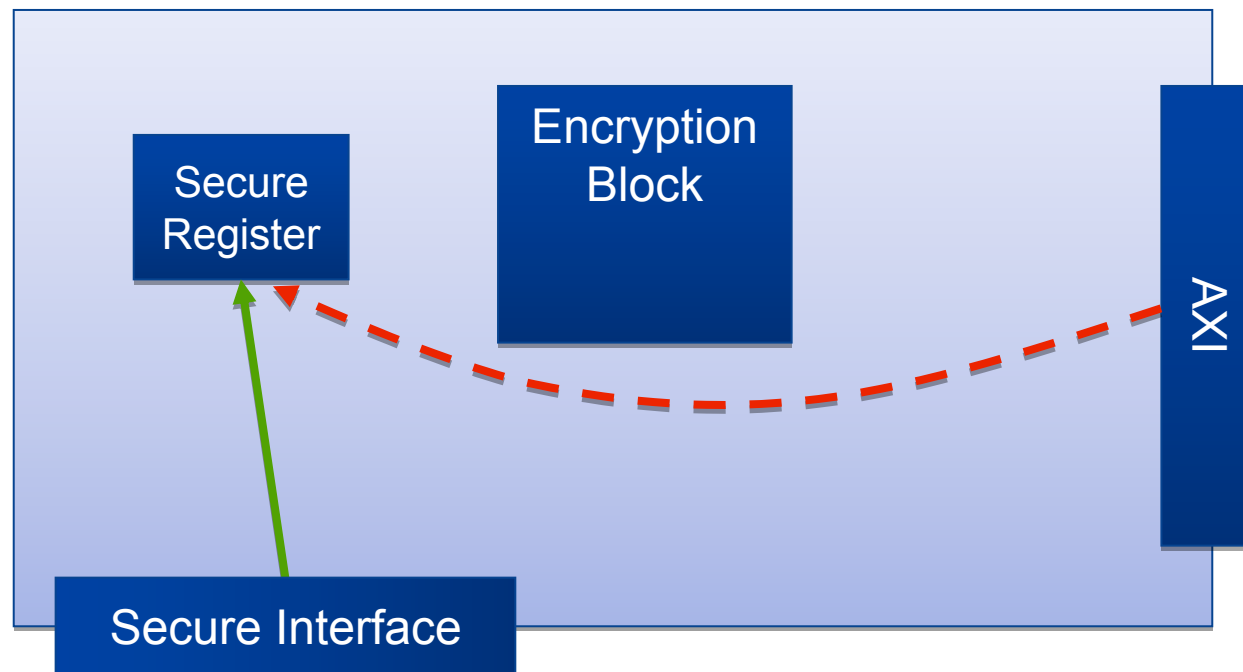
Usage example 1

- Verify that data in secure register does not propagate to AXI interface without going through encryption block



Usage example 1 (cont'd)

- Verify that data in secure register does not get overwritten by any data on AXI interface



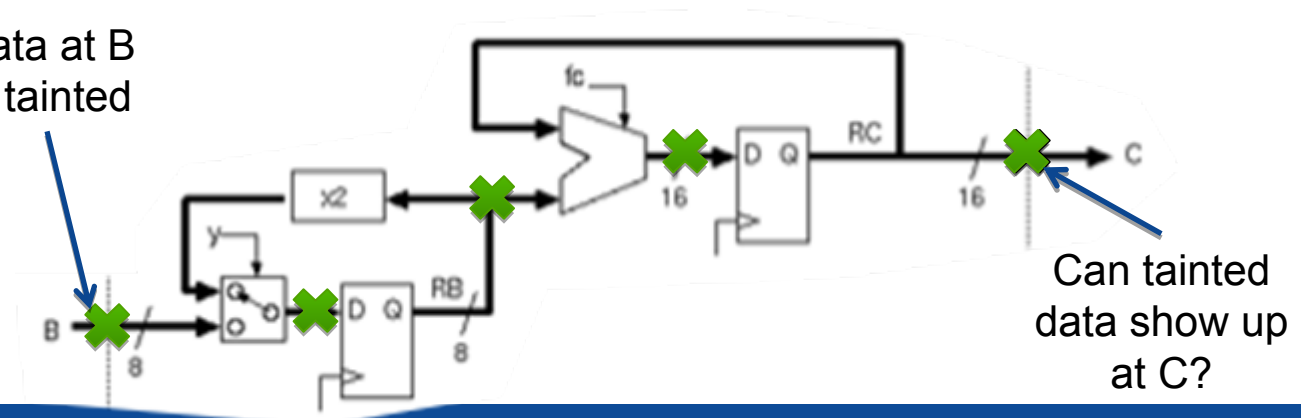
Path sensitization technology

- Path sensitization technology:
 - This technology introduces a new type of property: path cover, which has a source signal and a destination signal
 - No SVA equivalent for this property!
 - By proving this property, **data at the source of the path is tainted**. Then, Jasper formally verifies if it is possible to **cover tainted data at the destination**
 - When the property is **covered**, a waveform displays *how* data can propagate from source to destination
 - The property can also be determined to be **unreachable**, which means that it is not possible for data to propagate from source to destination

Example:

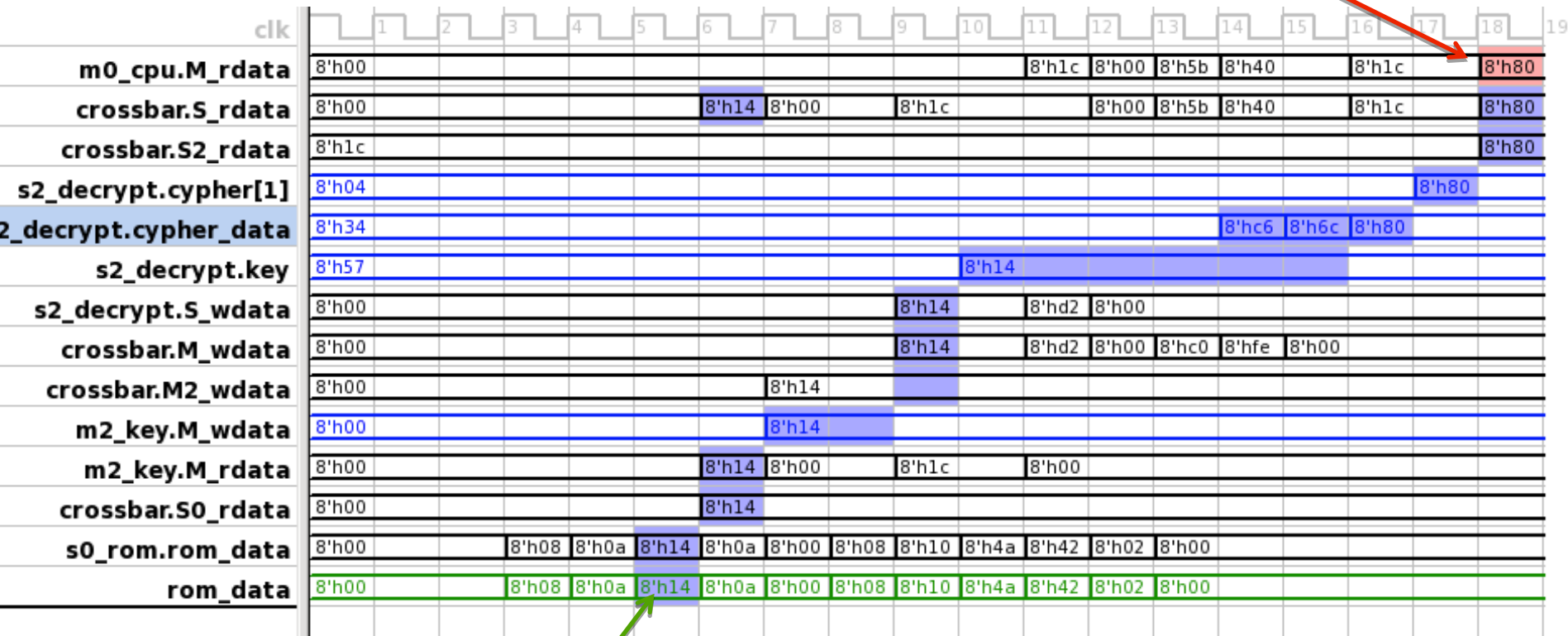
Path cover
from B to C

Data at B
is tainted



Waveforms

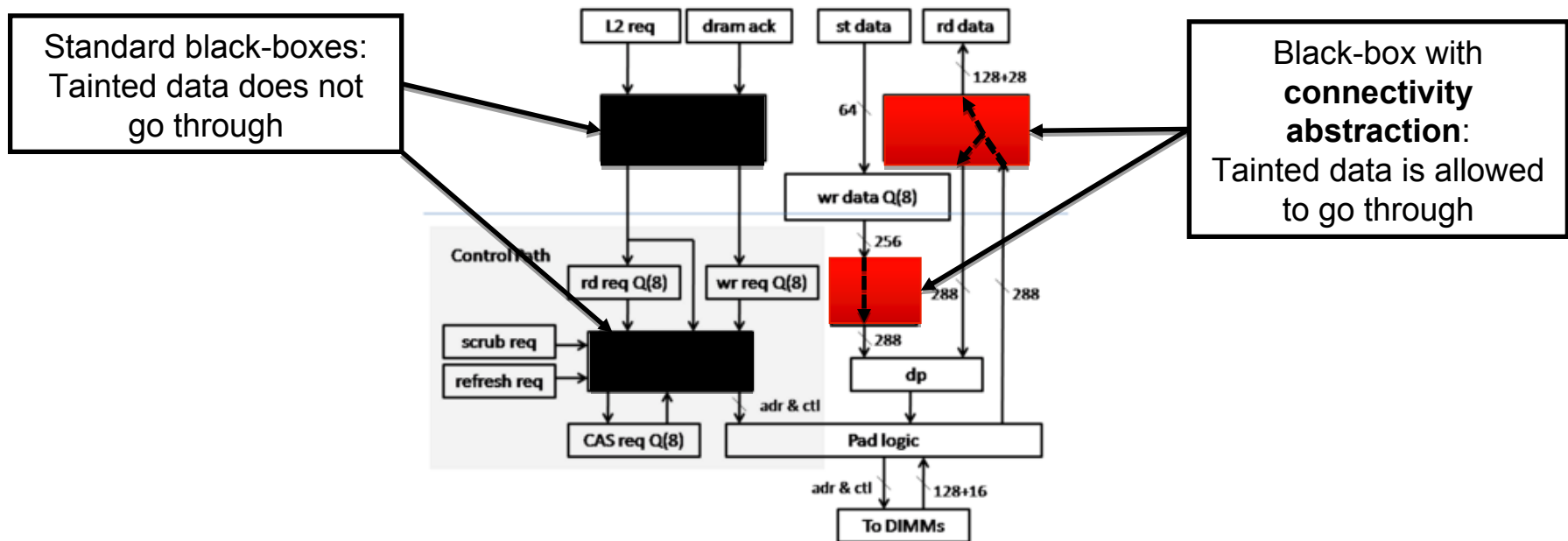
Red highlighting:
Data reached
destination



Taint generated at
the source

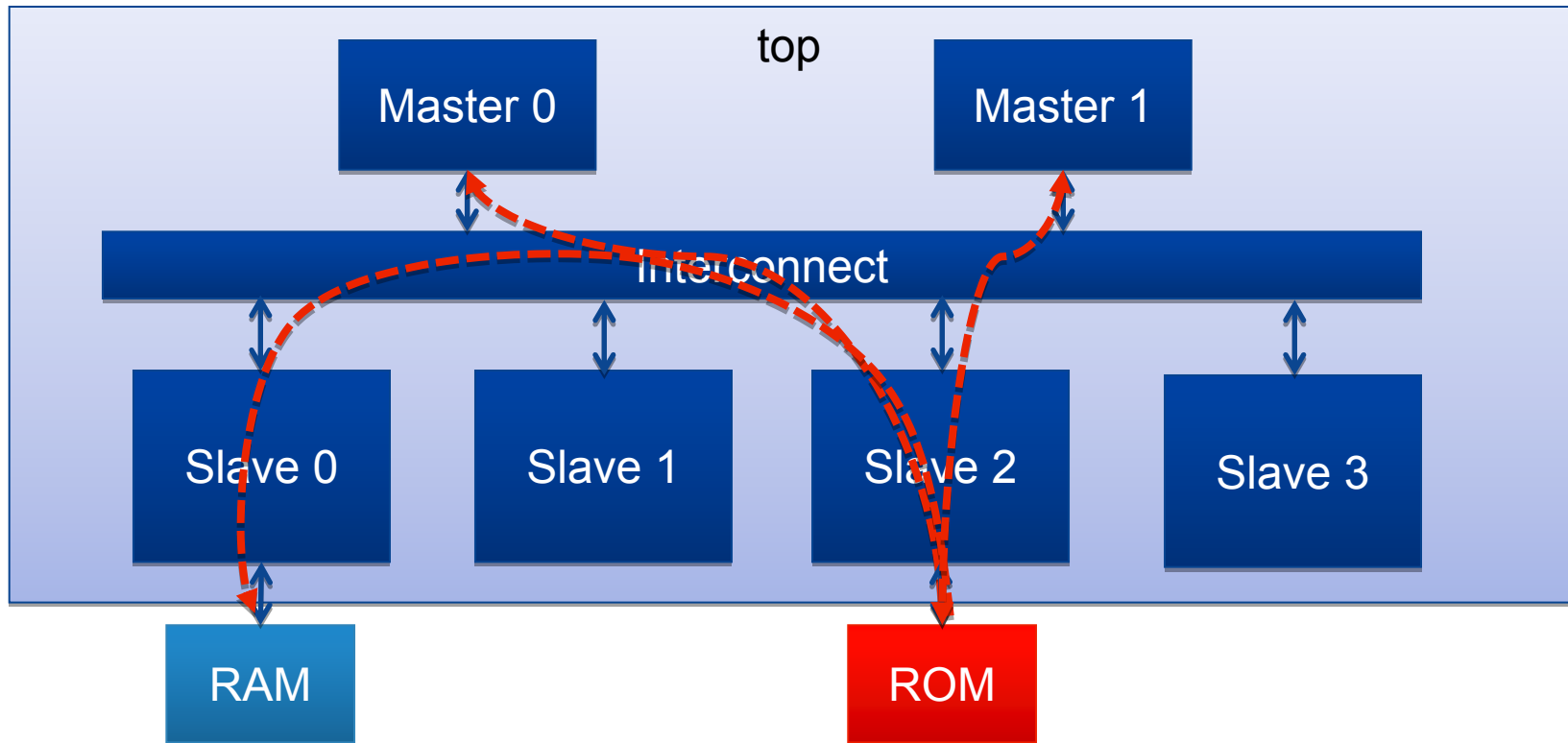
Black-Box with Connectivity abstraction

- During Security Verification, blocks can be black-boxed with a special qualification about whether tainted data is allowed to go through the black-box or not
- This allows the user to selective black-box modules and tune the verification so only meaningful paths are found
- This feature is important to guarantee the scalability of this verification while still keeping results sound



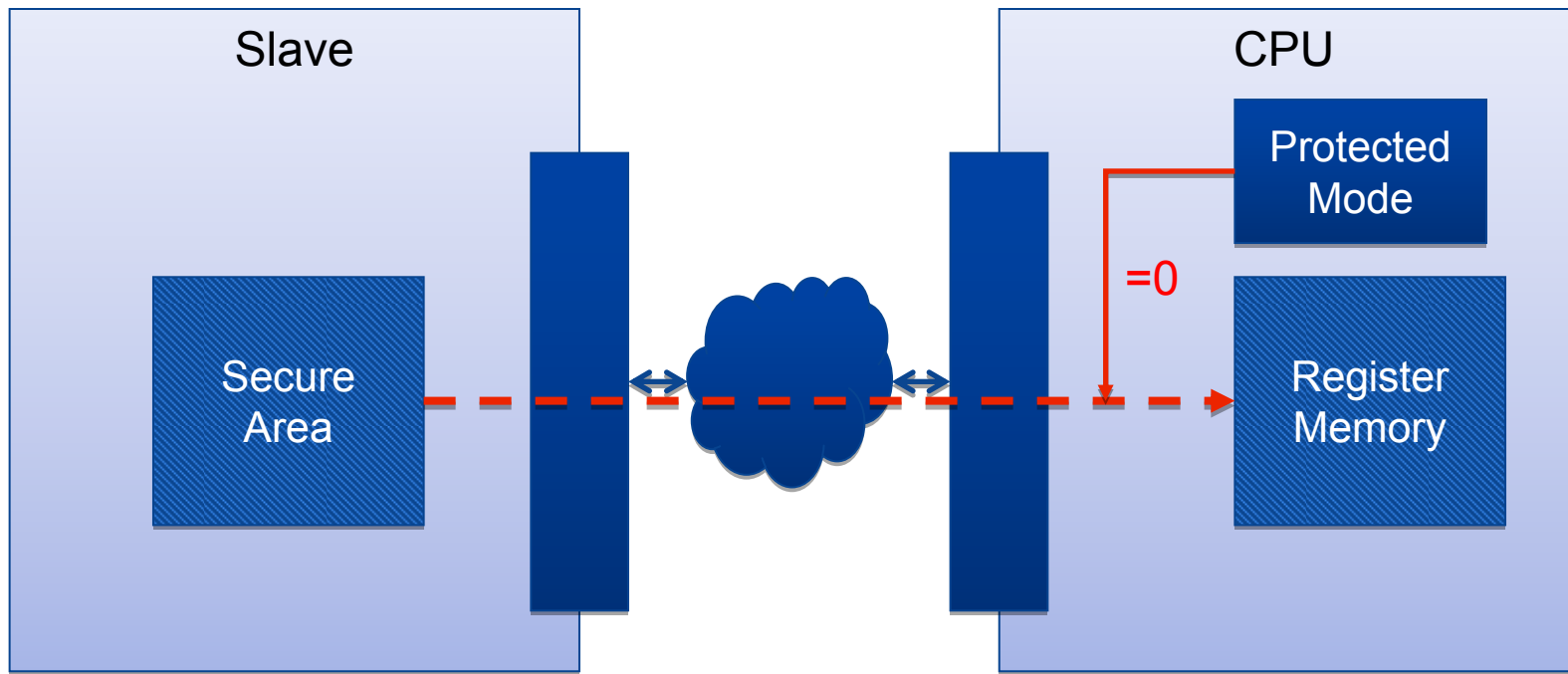
Usage example 2

- Verify that data read from ROM does not propagate to some key points of the design



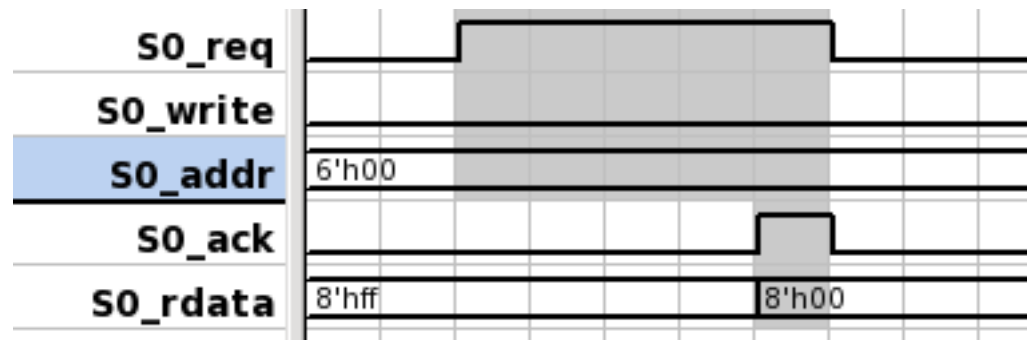
Usage example 3

- Verify that data from secure area never propagates to CPU register memory if “protected mode” is set to 0

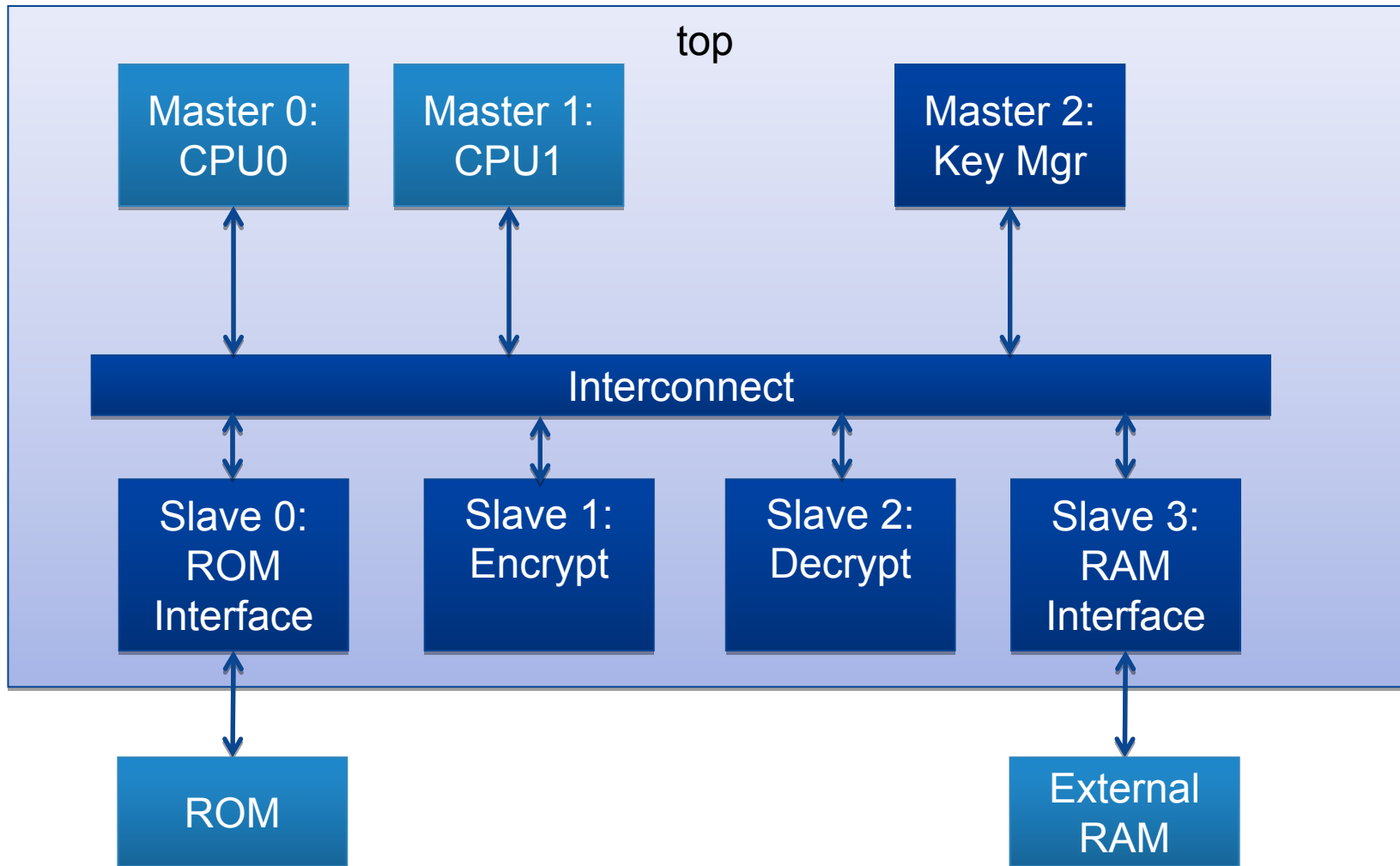


Usage example 4

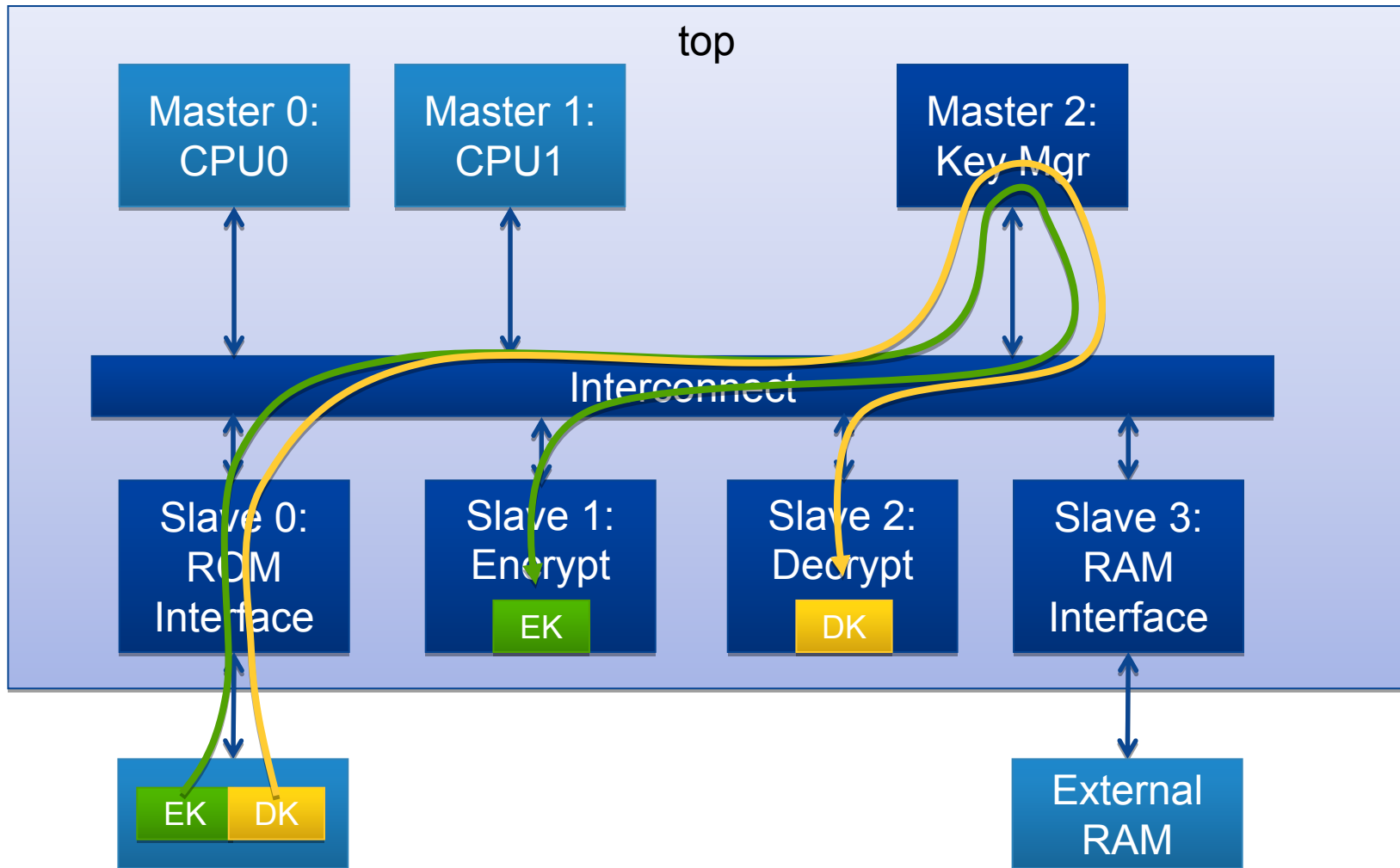
- The design is a small network, consisting of external memory, external secure ROM, two CPUs and some internal elements.
- The Encrypt/Decrypt slaves receive cryptography keys from the Key Manager (a master) and encrypts/decrypts data written to some of its memory locations
- The Key Manager reads the keys from the ROM and transfers it to the slaves after reset
- The goal is to verify that there is no undesired leak of this key to other parts of the system, such as CPU and external memory
- Bus protocol is simple request/acknowledge



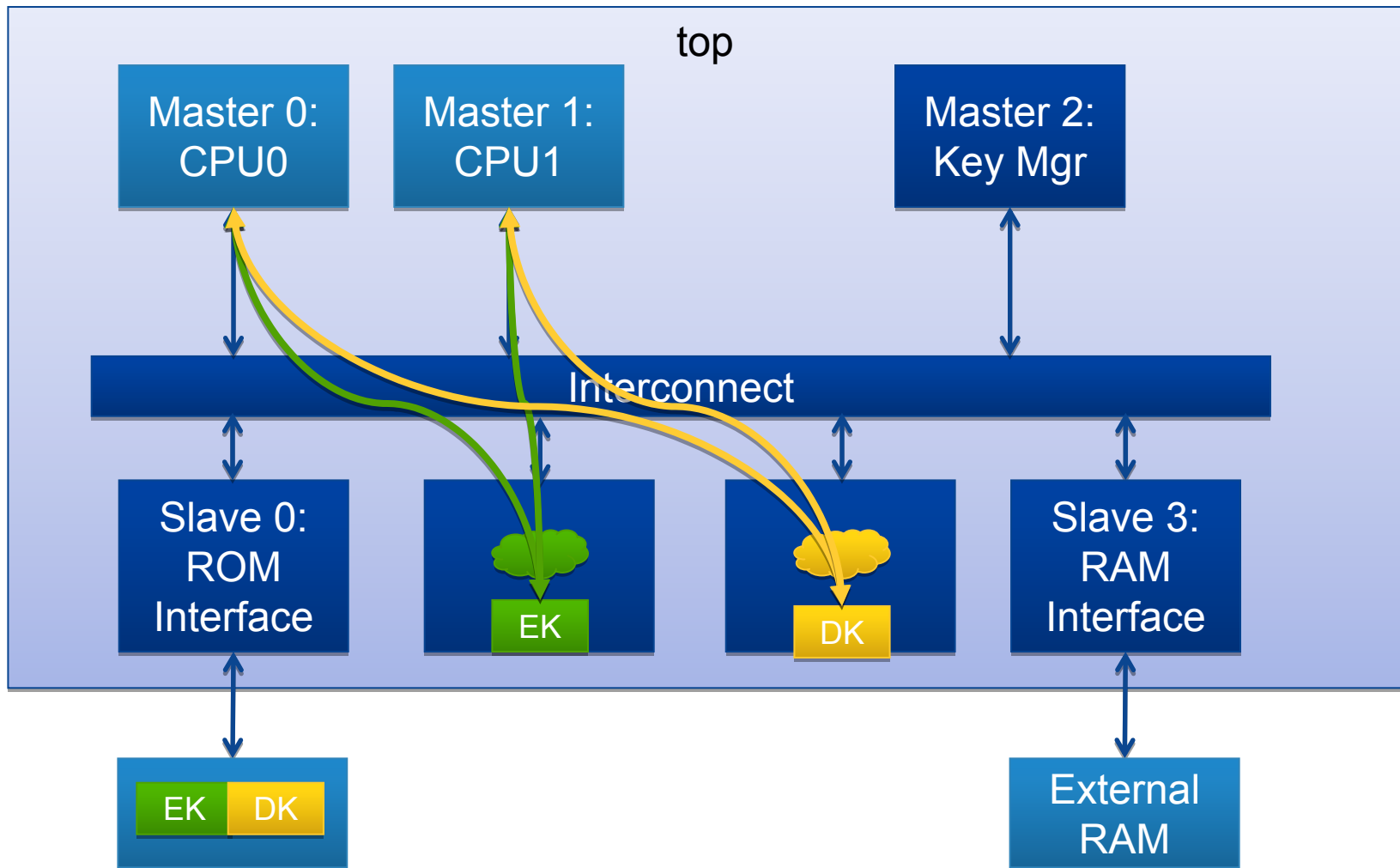
Design Overview



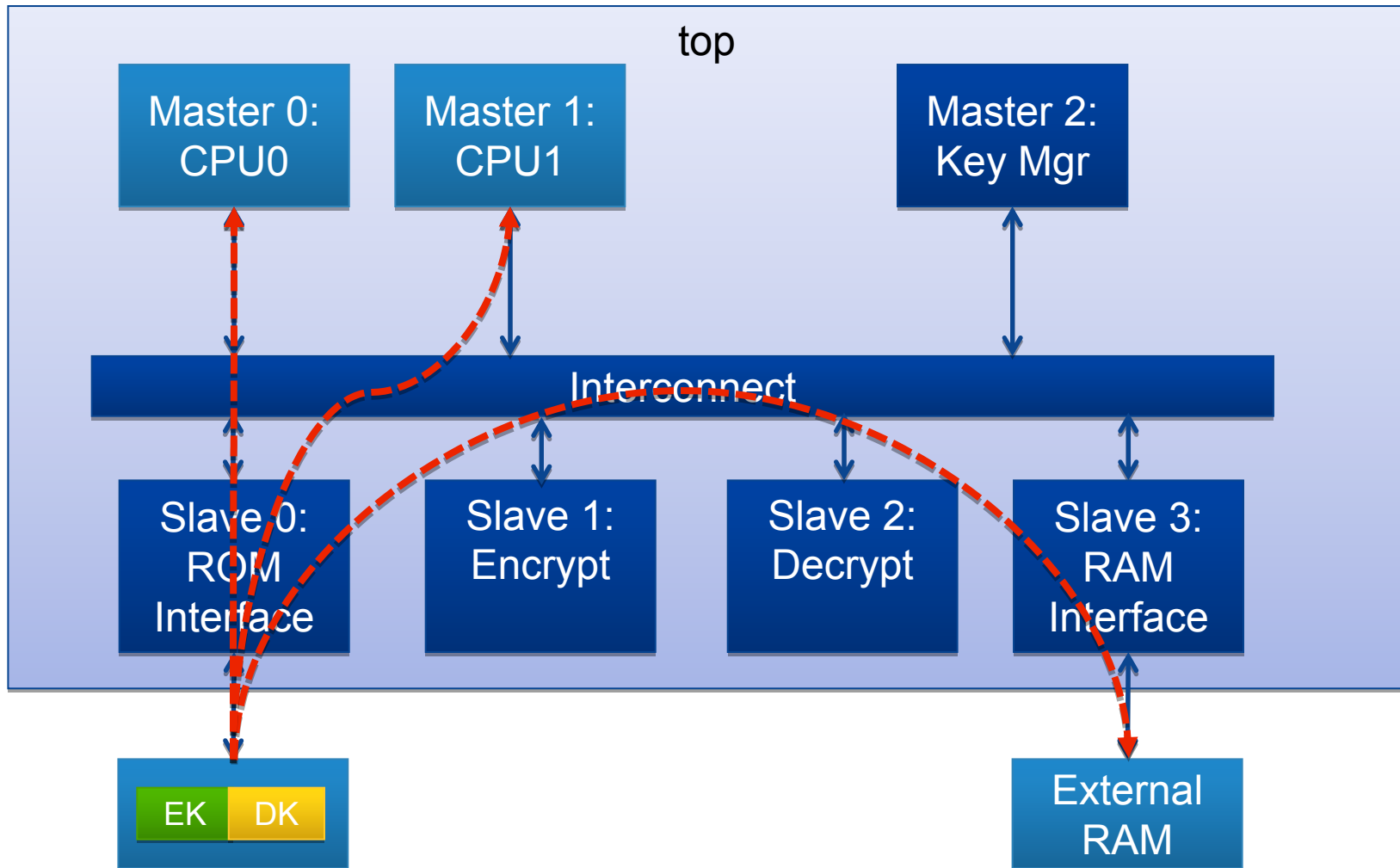
Reading the keys from the ROM to the slaves



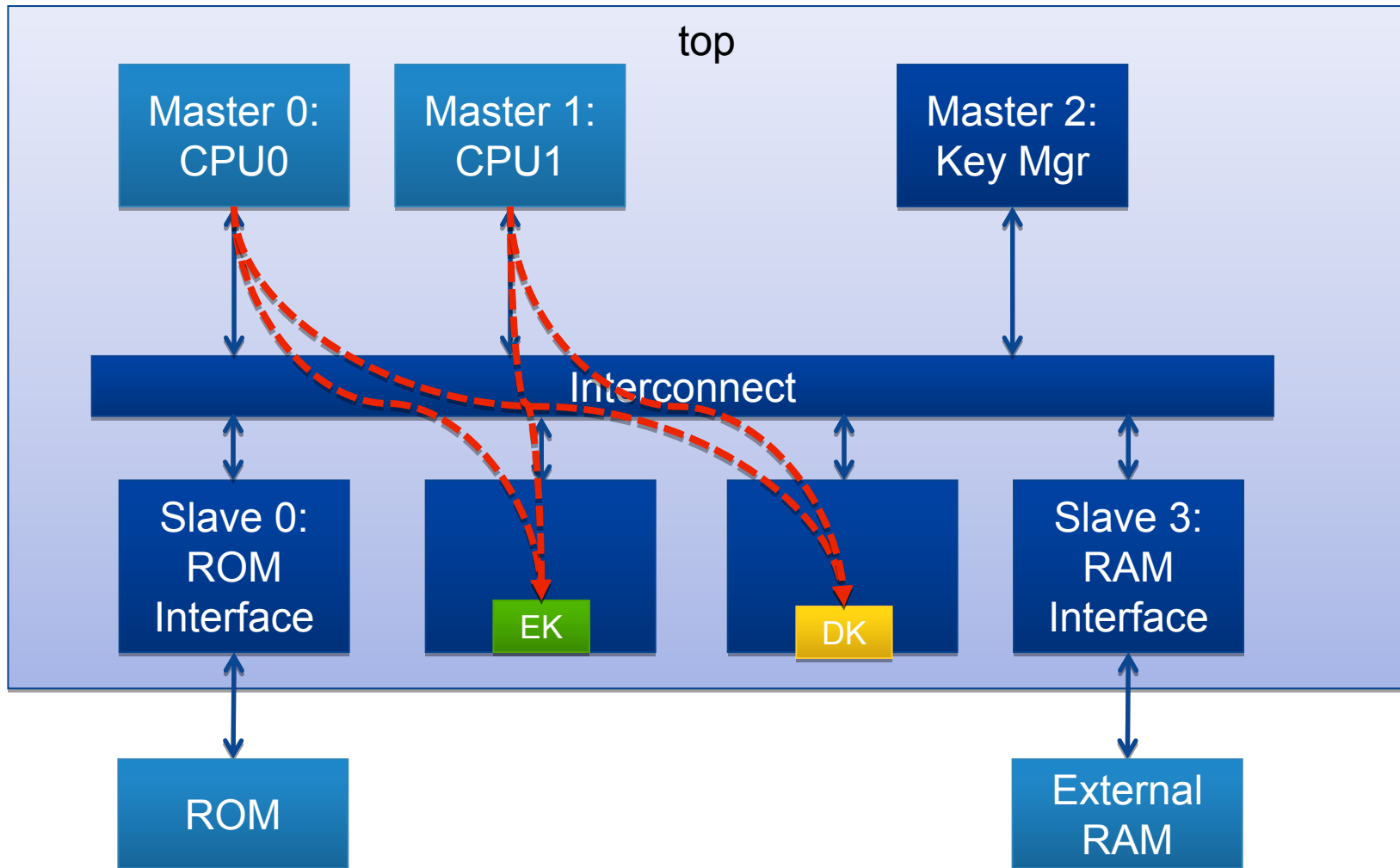
Valid encryption or decryption



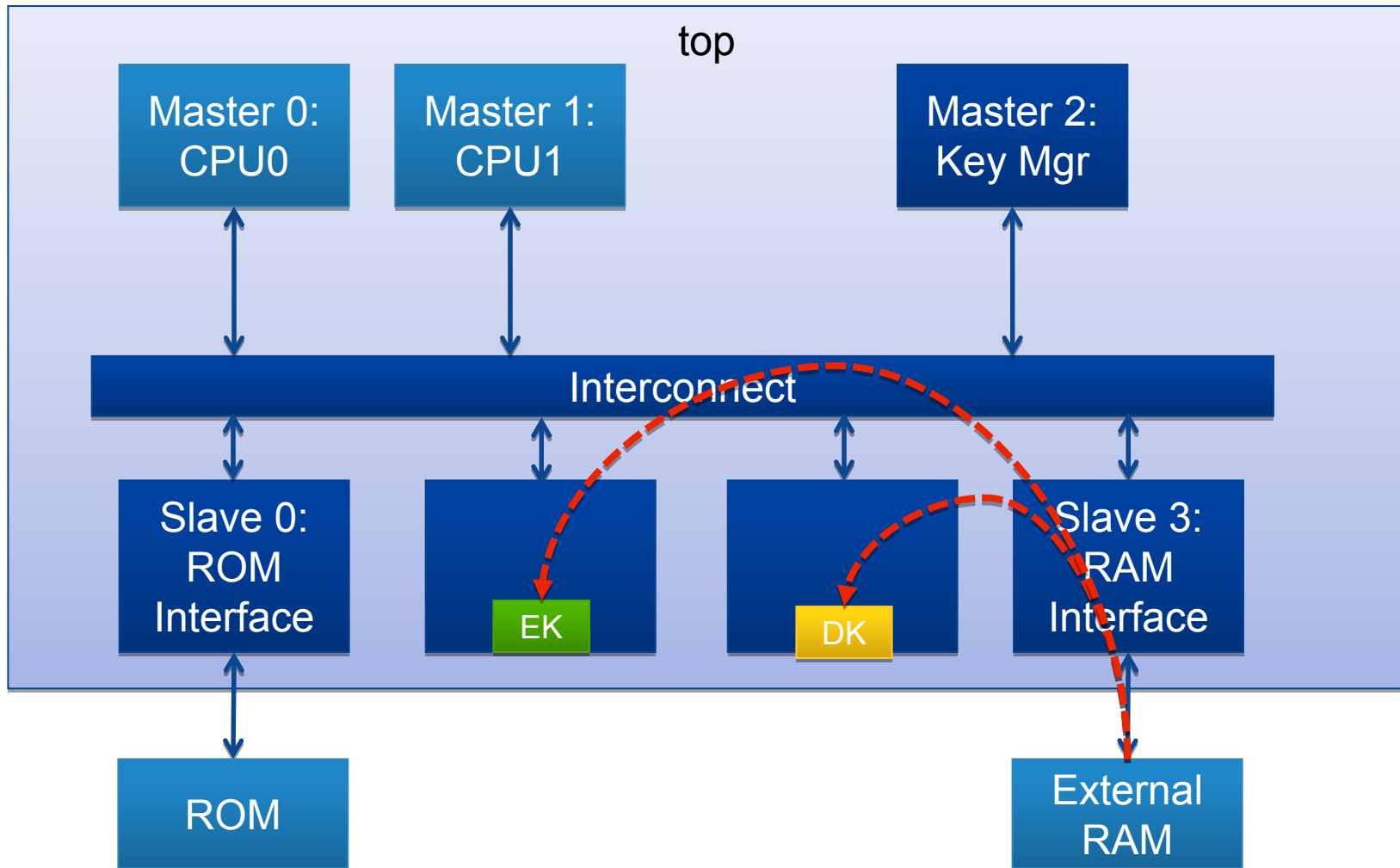
Check 1: ROM leak



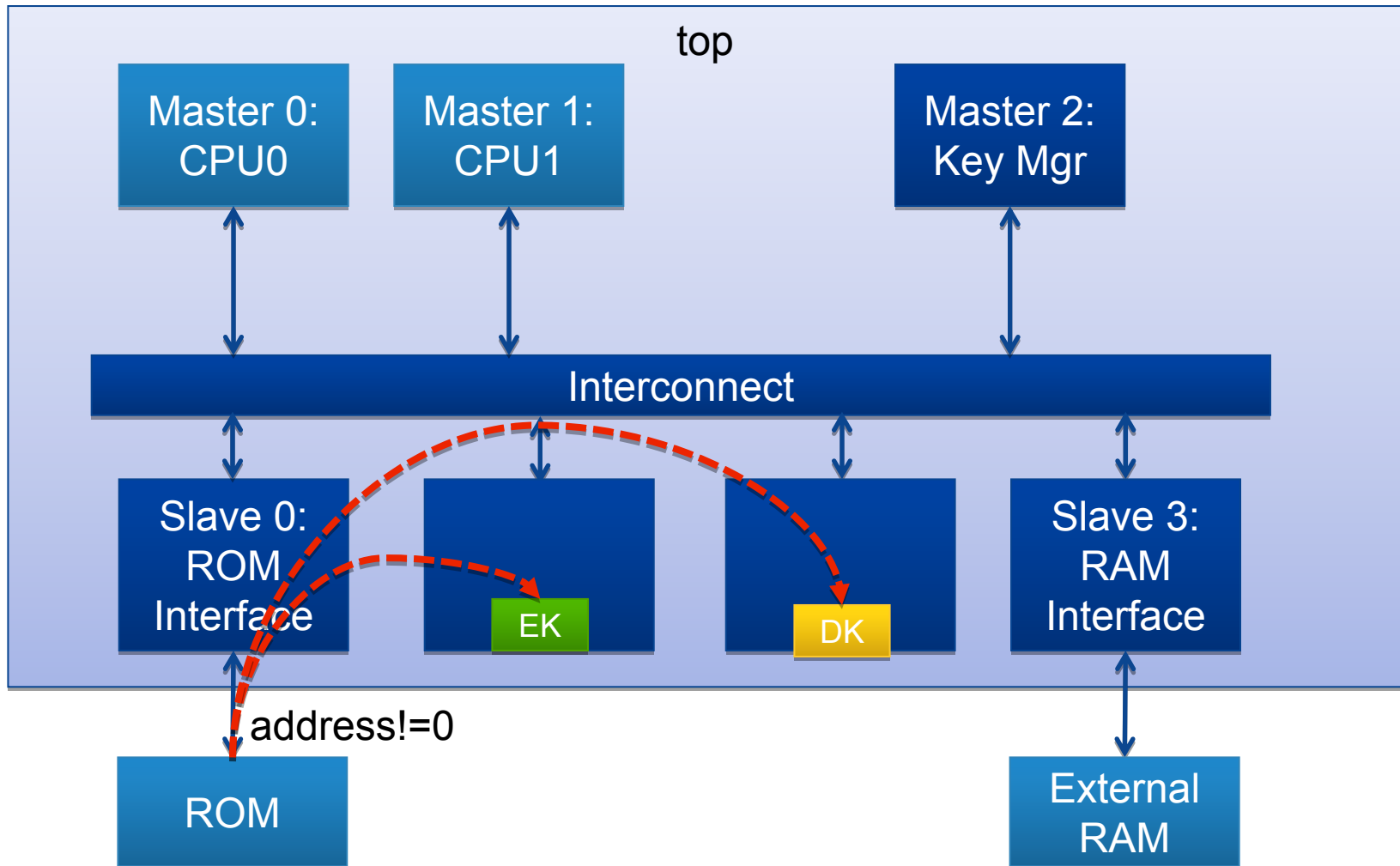
Check 2: Illegal key overwrite



Check 2: Illegal key overwrite



Check 2: Illegal key overwrite from wrong ROM address



Conclusion: Security Path Verification

- Verify the lack of functional paths to/from secure areas of a design
- Based on path sensitization technology
- Requirements are not expressible by regular SVA assertions, therefore not possible with standard FV tools
- Special black-boxing to allow scalability to bigger designs
- Easy debug with special highlighting in the waveform