Verifying Hardware/Software Co-Design for Remote Attestation in IoT

Norrathep Rattanavipanon
(PSU, Phuket Campus)

Joint work with:
Ivan de Oliveira Nunes (UCI)
Gene Tsudik (UCI)
Karim Eldefrawy (SRI Intl)
Michael Steiner (Intel Research)

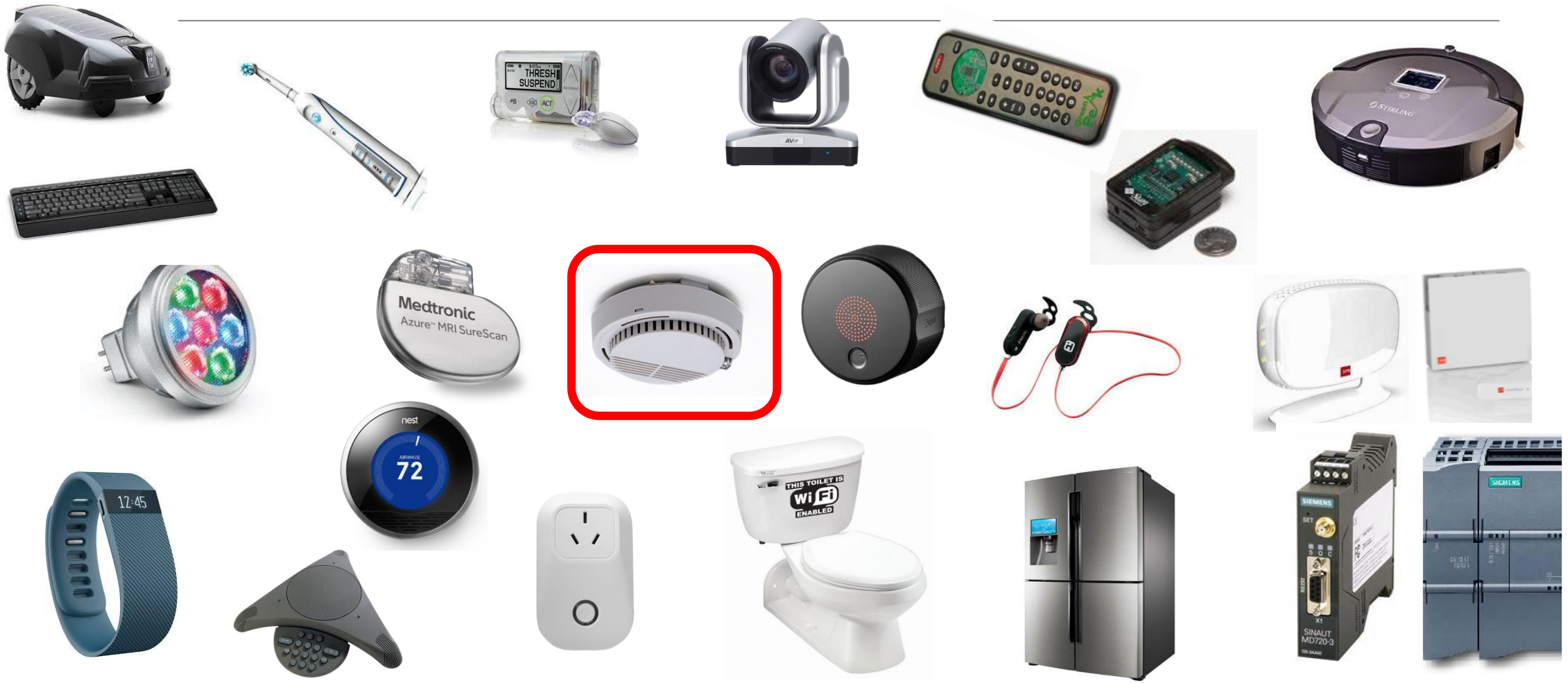
VRASED: A Verified Hardware/Software Co-Design for Remote Attestation,
Usenix Security Symposium (Usenix'Sec), 2019.
https://www.usenix.org/conference/usenixsecurity19/presentation/de-oliveira-nunes

Outline

- Past
- Present
  - Background on Remote Attestation
  - Our Approach (VRASED)
  - Implementation + Results
- Future Work

Background on Remote Attestation for Low-End IoT/CPS Devices

Internet-of-Things (IoT) Gadgets



IoT-specific Attacks On:

Sensing: Privacy

Actuation: Security & Safety

Either: DDoS Sourcing (aka Zombification)

Constraints for Simple IoT Devices: large scale + low price

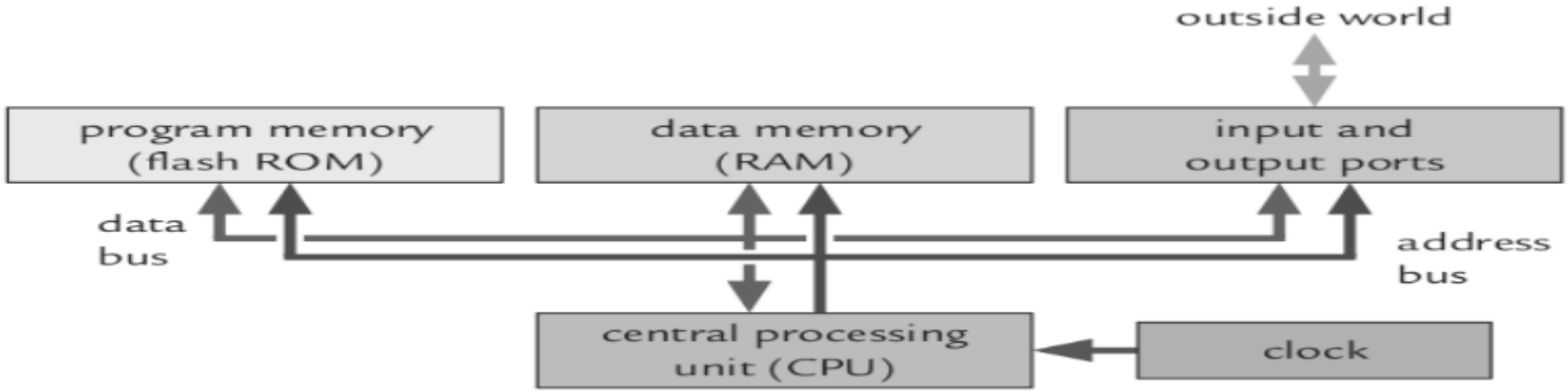
CPU Power	Battery

Hard to prevent malware from entry

Next best thing is to detect malware!

Memory Size	Hardware Cost

Low-end IoT Devices
(aka *amoebas* of the computing world)



- Designed for: **Low-Cost**, **Low-Energy**, **Small-Size.**
- Memory: Program (~32kB) and Data (~2-16 kB)
- Single core CPU (~8-16MHz; 8- or 16-bit)
- Simple Communication (I/O) Interfaces (a few kbps)
- Examples: TI MSP-430, AVR ATMega32 (Arduino)

Detection vs. Prevention for IoT Amoebas

- Prevention is hard & expensive:
  - Simple devices can not run fancy crypto, anti-malware, verify certificates, etc.
- Detection is the next best thing:
  - Goal: Remotely measure internal state of device and detect anomalous/compromised states

Remote Attestation (RA)

- A general approach of detecting malware presence on devices
- Two-party interaction between:
  - **Verifier**: trusted entity
  - **Prover**: potentially infected and untrusted **remote** IoT device
- Goal: measure current internal state of **prover**

RA Interaction

Verifier

Prover

(1) Challenge

(2) Response = authenticated challenge-based measurement (via some cryptographic integrity-ensuring function)

(3) Response

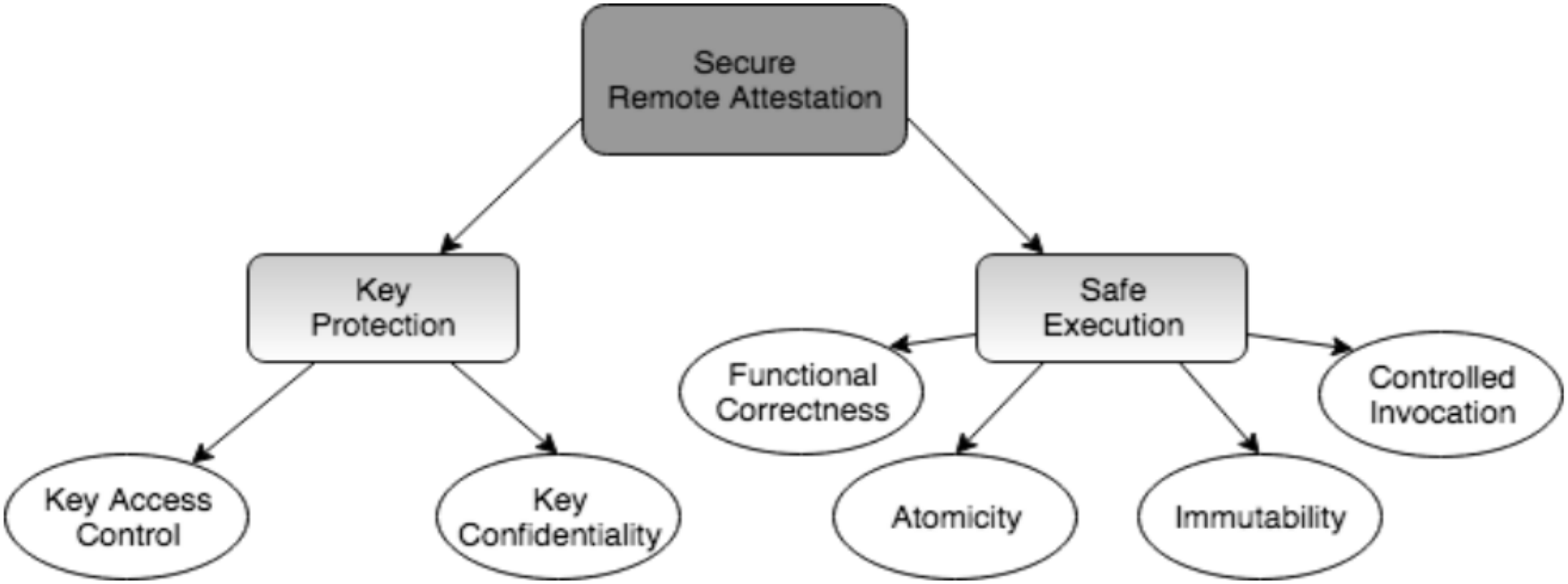
(4) Verify response, decide outcome

Often implemented as a Message Authentication Code (MAC) over prover's memory

RA Techniques

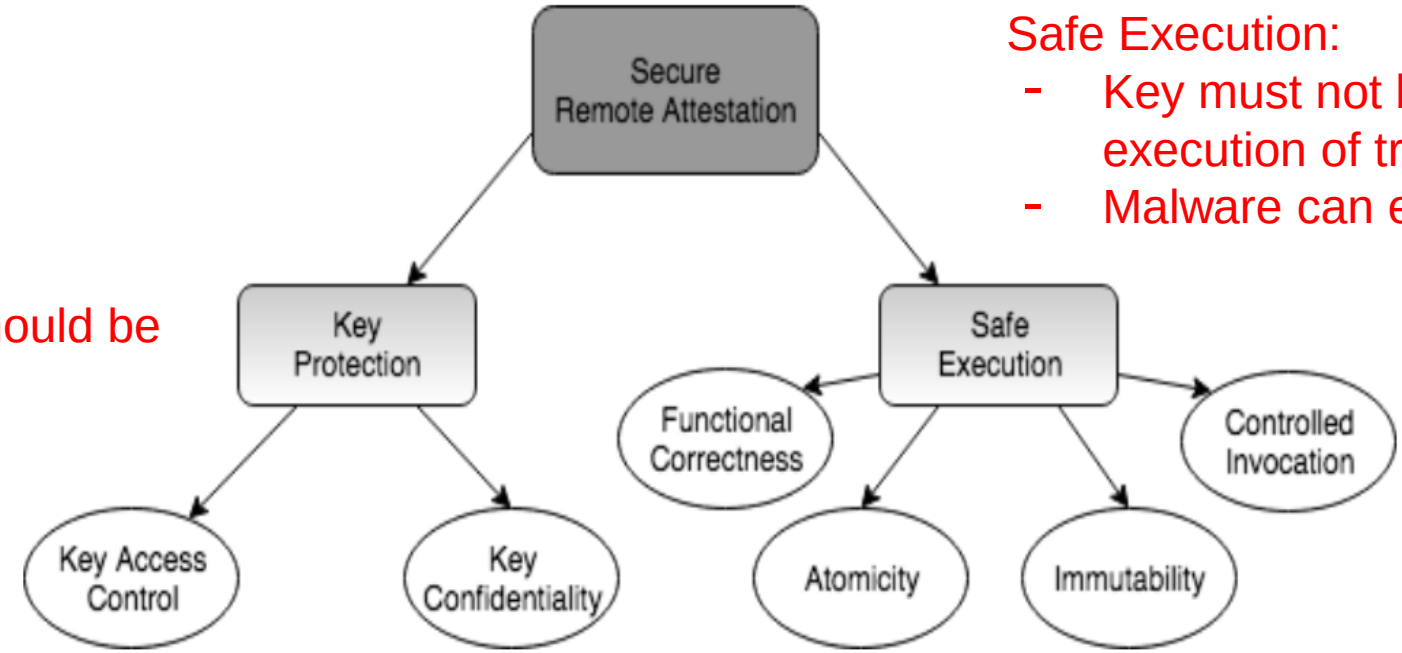
- Hardware-based RA
  - Dedicated hardware support (e.g., TMP, TrustZone-M, SGX)
  - Effective
  - Overkill for low-end IoT devices
- Software-based RA
  - Relies on precise or negligible timing
  - Unrealistic assumptions for **remote** provers, except for peripherals and legacy devices
- Hybrid RA
  - SW/HW co-design
  - Minimal hardware impact
  - **Best fit for resource-constrained IoT devices?**

Hybrid RA Security Properties



Hybrid RA Security Properties

Authenticated measurement requires prover to have a unique secret key
If this key is leaked, RA is totally broken/useless



Safe Execution:

- Key must not be leaked during execution of trusted code
- Malware can escape detection

Potential malware on prover should be unable to access this key

Why bother with formally verified RA?

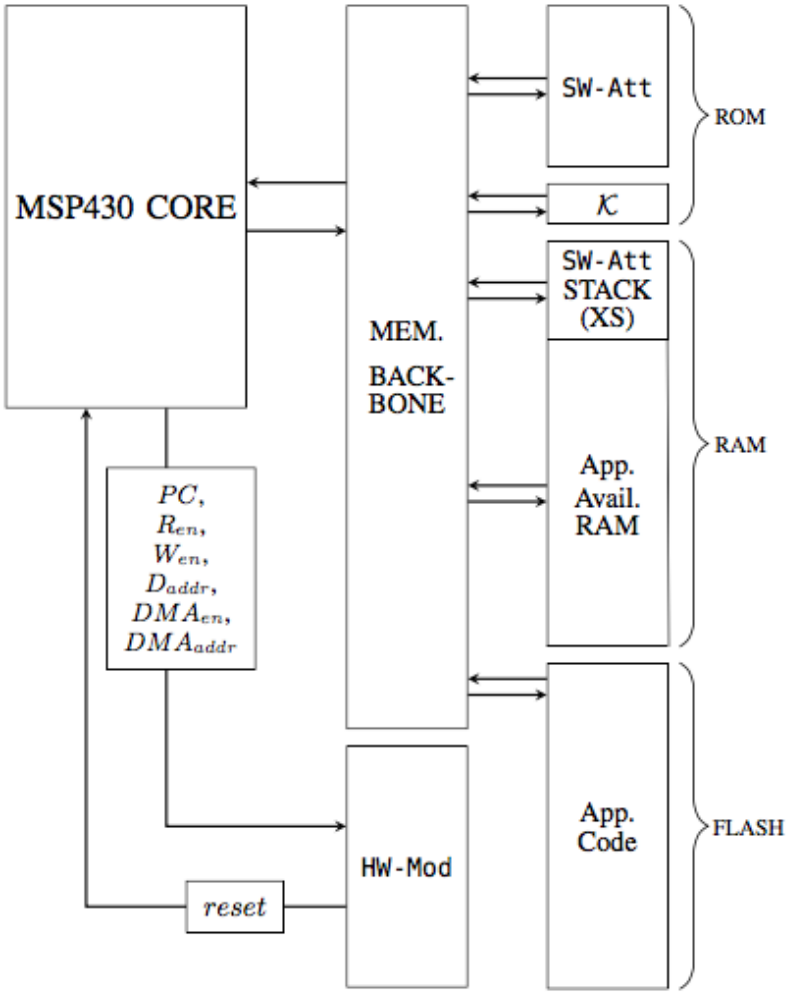
- FV promises higher confidence and concrete security guarantees
- Current RA techniques do not offer high-assurance and rigor derivable from FV to guarantee security of the design and its implementation.
- Since they are not systematically designed from abstract models, soundness and security (of current RA) cannot be formally argued.
- Need to design more-or-less from scratch in order to construct a formally verifiable RA scheme.
- **No prior formally verified secure HW/SW co-design**

VRASED:
Verifiable **R**emote **A**ttestation for **S**imple **E**mbedded **D**evices

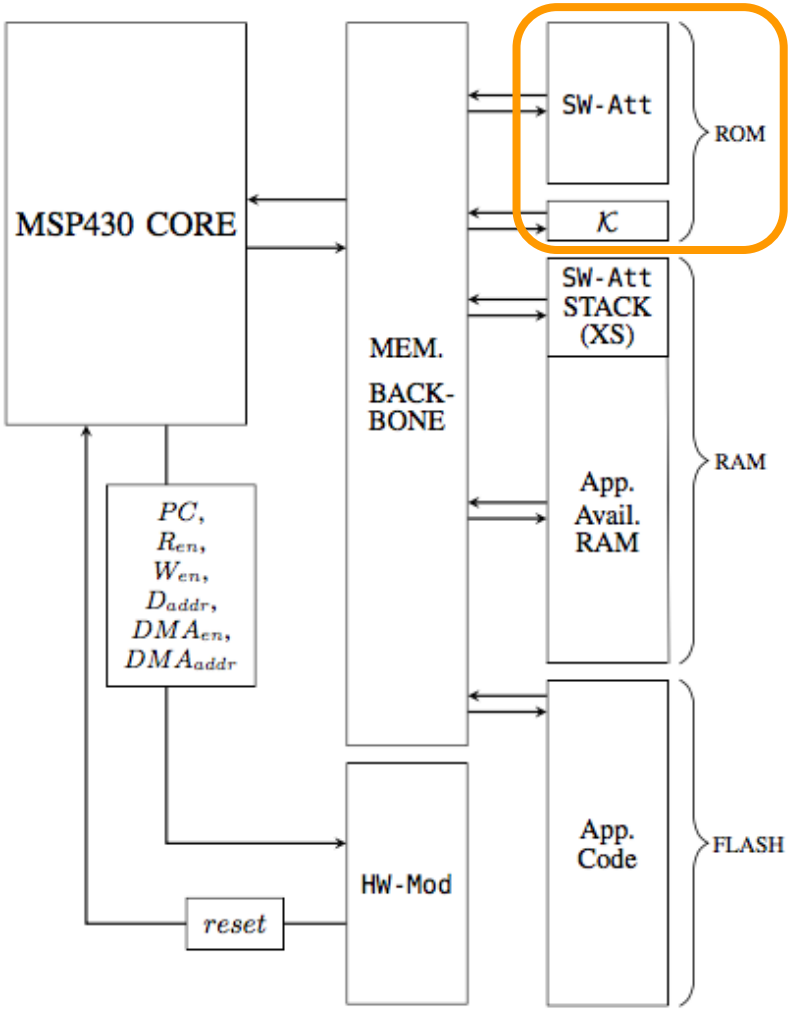
ARCHITECTURE Overview

OUR MAIN GOAL: Formally Verified RA Design and Implementation

VRASED Architecture

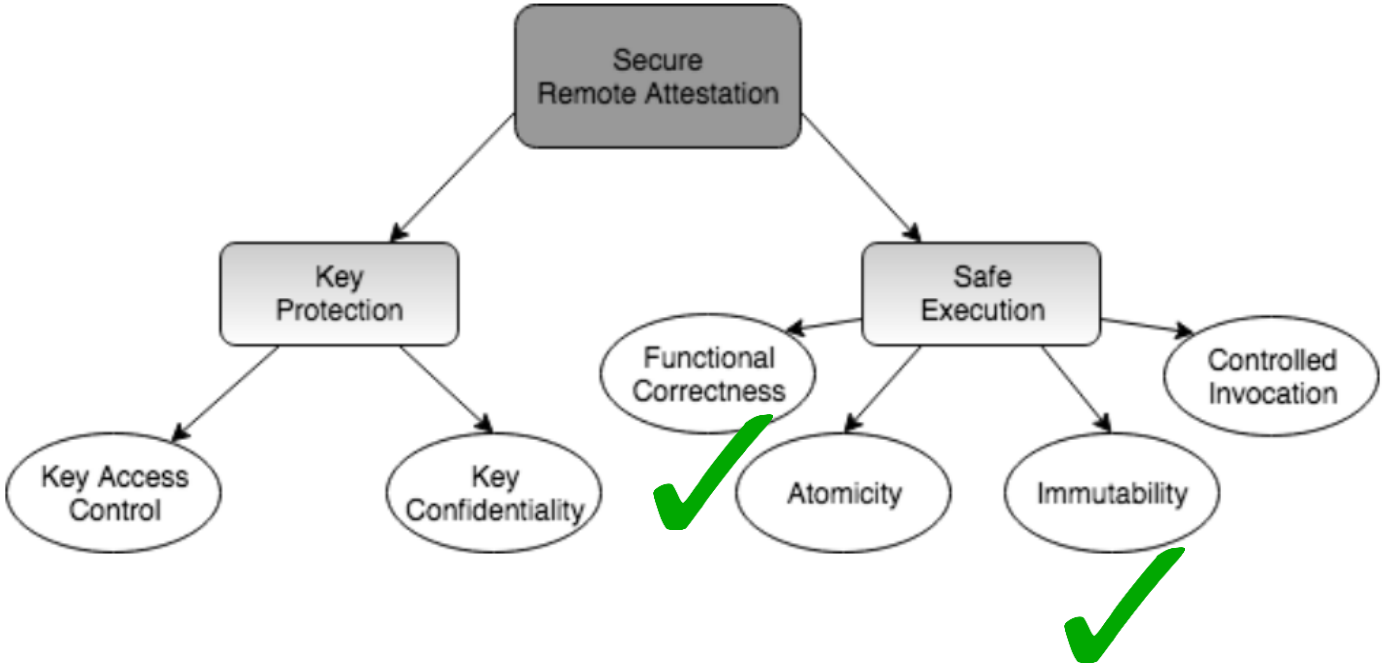


VRASED Architecture

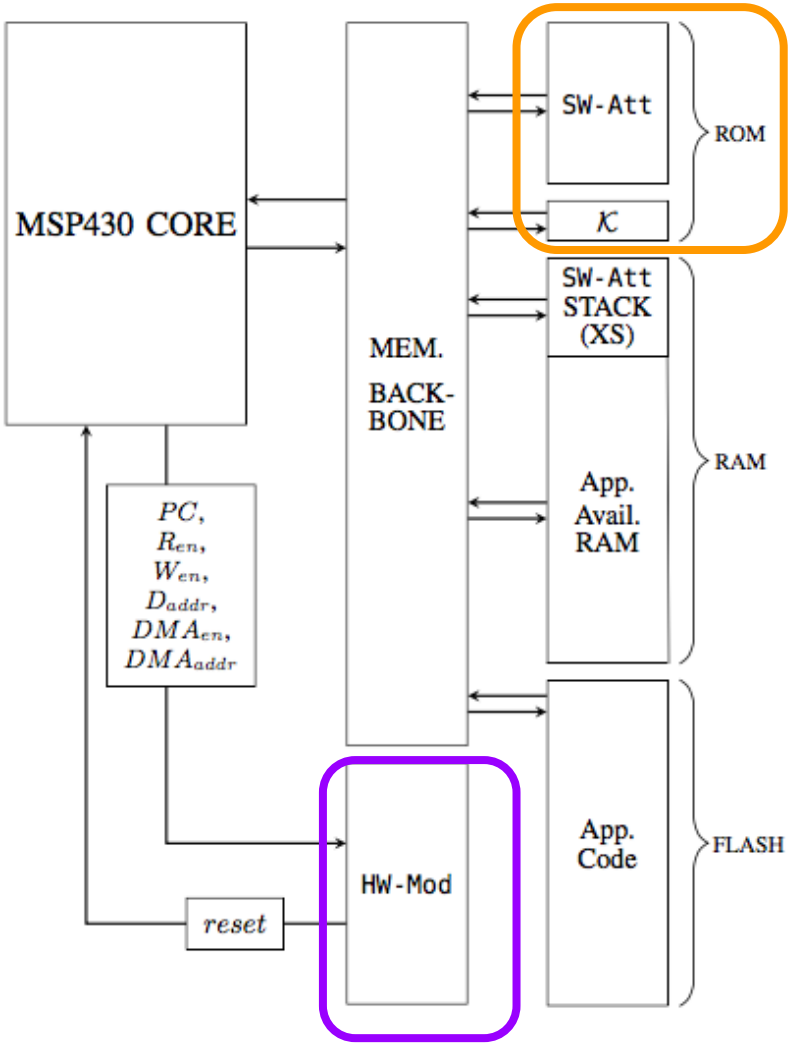


- Verified Implementation of HMAC stored in **ROM** (***SW-Att***): Immutability + SW correctness
  - Malware can not modify ***SW-Att***

Hybrid RA Security Properties



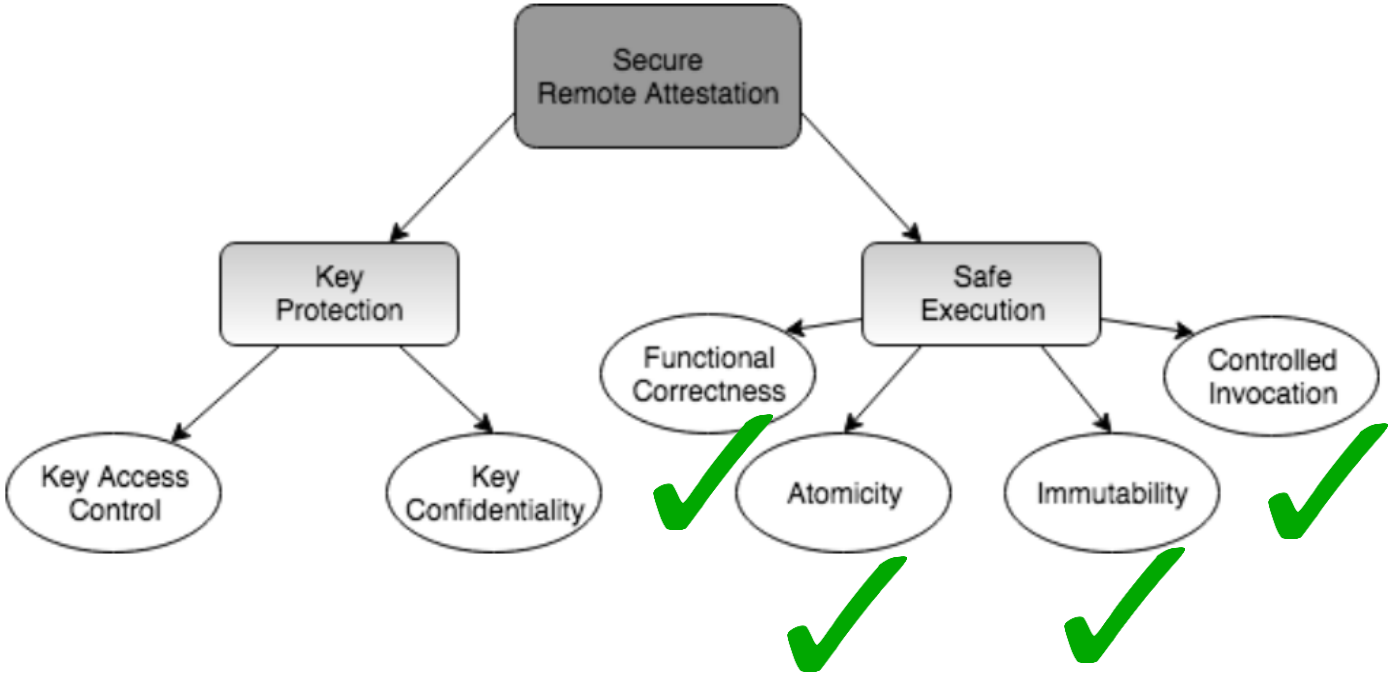
VRASED Architecture



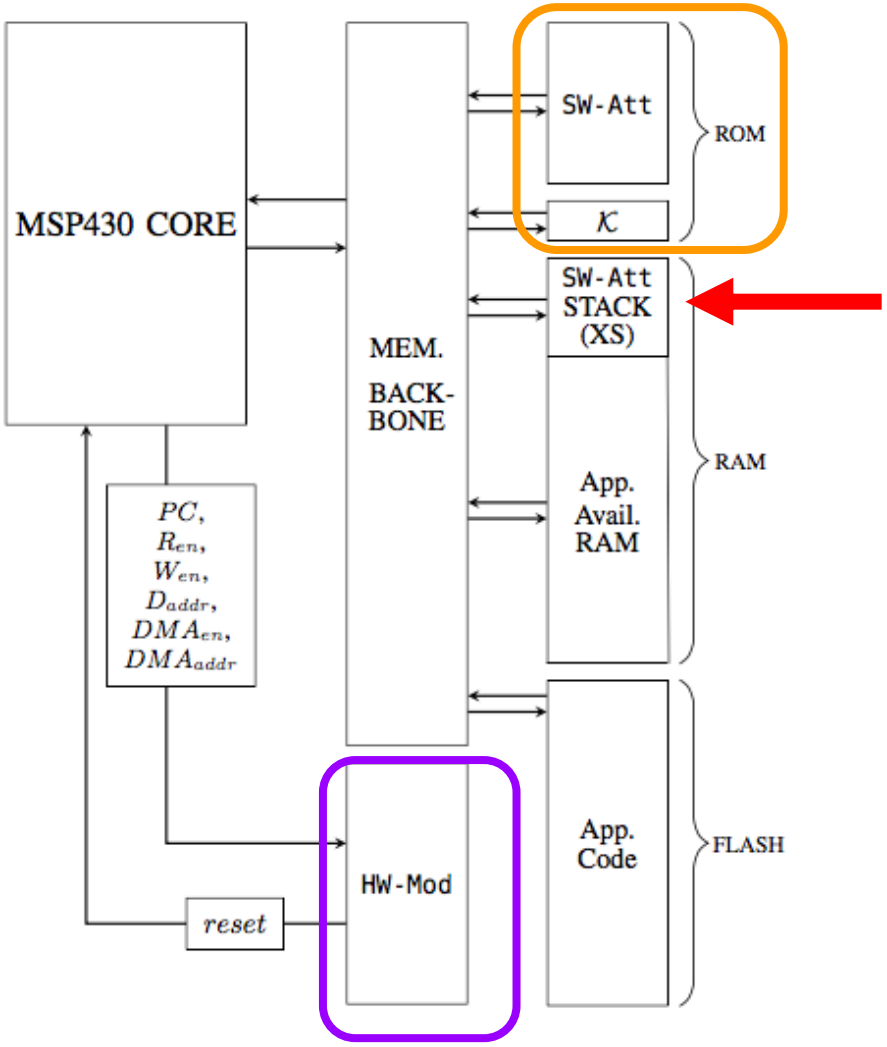
- Verified Implementation of HMAC stored in read only memory (***SW-Att***): Immutability + SW correctness
  - Malware can not modify attestation software

- Does ***SW-Att*** execute properly?
  - ***HW-Mod*** enforces proper invocation and atomicity

Hybrid RA Security Properties

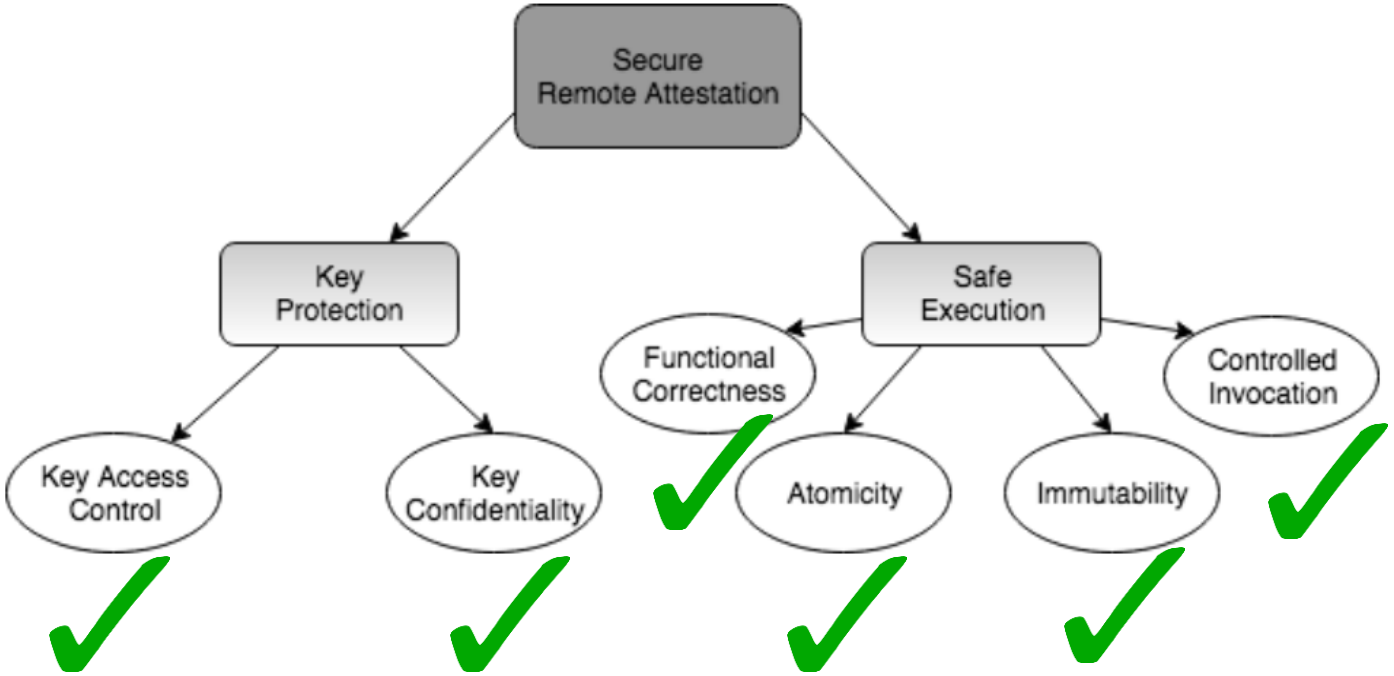


VRASED Architecture



- Verified Implementation of HMAC stored in read only memory (***SW-Att***): Immutability + SW correctness
  - Malware can not modify the attestation software
- Does ***SW-Att*** execute properly?
  - ***HW-Mod*** enforces proper invocation and atomicity
- Can Malware learn something it should not from ***SW-Att*** execution?
  - ***HW-Mod*** makes sure it can not
  - Access control to the key and to memory used by *SW-Att*

Hybrid RA Security Properties



VERIFYING VRASED

Verifying VRASED Security

Goals:

- Verify *HW-Mod* hardware design with respect to aforementioned properties
- Verify software implementation of ***SW-Att***

Verifying VRASED Security

- Verify *HW-Mod* hardware design with respect to aforementioned properties
- Verify implementation of ***SW-Att***
- Prove that conjunction of such properties implies some formal notion of end-to-end "Secure Remote Attestation"

Our Approach

1. Use a formally verified HMAC implementation for ***SW-Att***
2. Model RA security properties as Linear Temporal Logic (LTL) Specs
3. Design HW-Mod as a set of FSMs, and use a model-checker to verify its conformance to LTL Specs
4. Prove (in LTL) that combination of ***SW-Att*** & HW-Mod properties imply "Secure Remote Attestation"

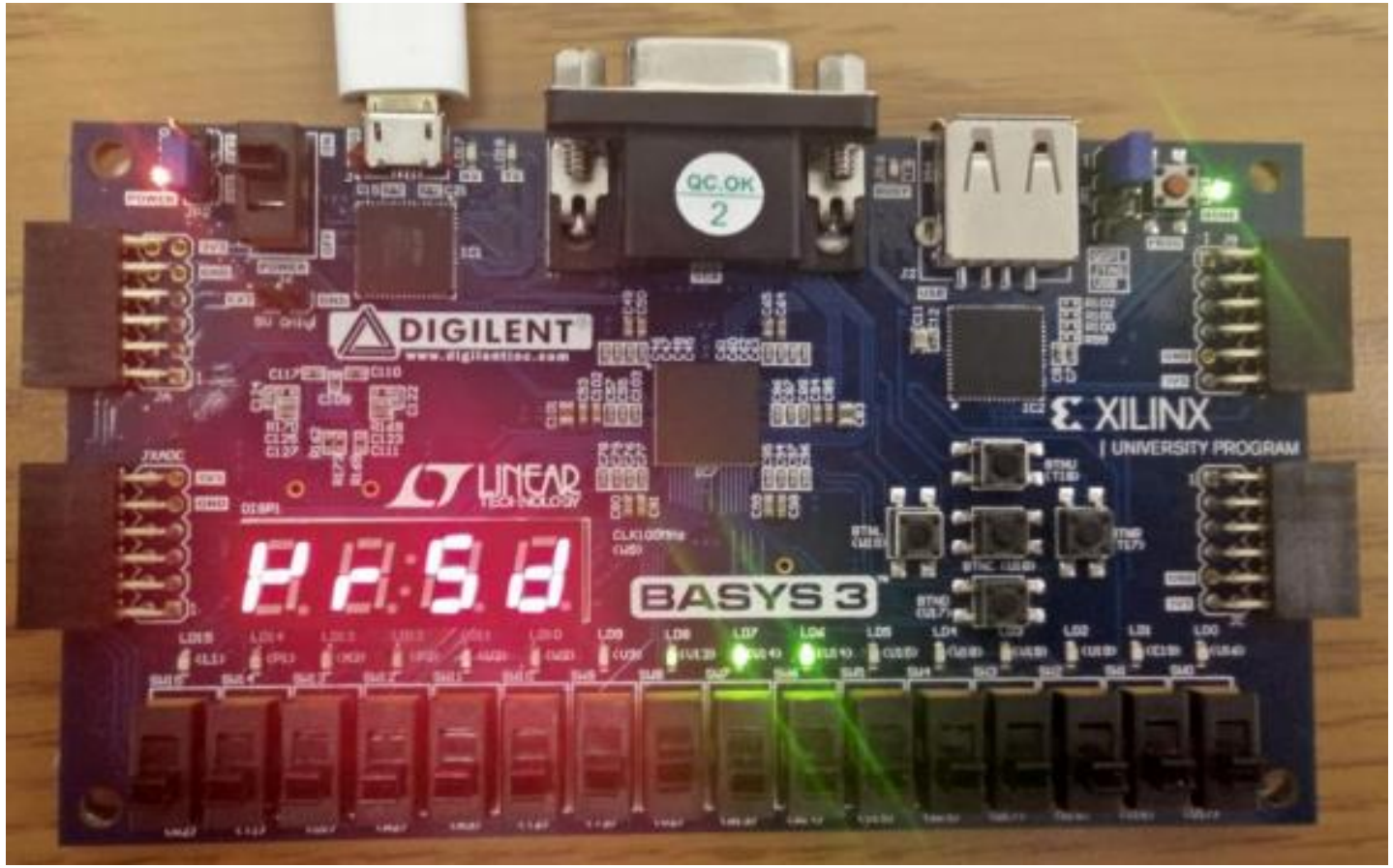
Design and Tools:

- Design hardware modules as FSMs in Verilog
- Use ***Verilog2SMV*** to convert between Verilog and SMV FSM representations
- Use ***NuSMV*** (Symbolic Model Checker) to verify FSMs against LTL specifications
- Implement by extending OpenMSP430 open-hardware project

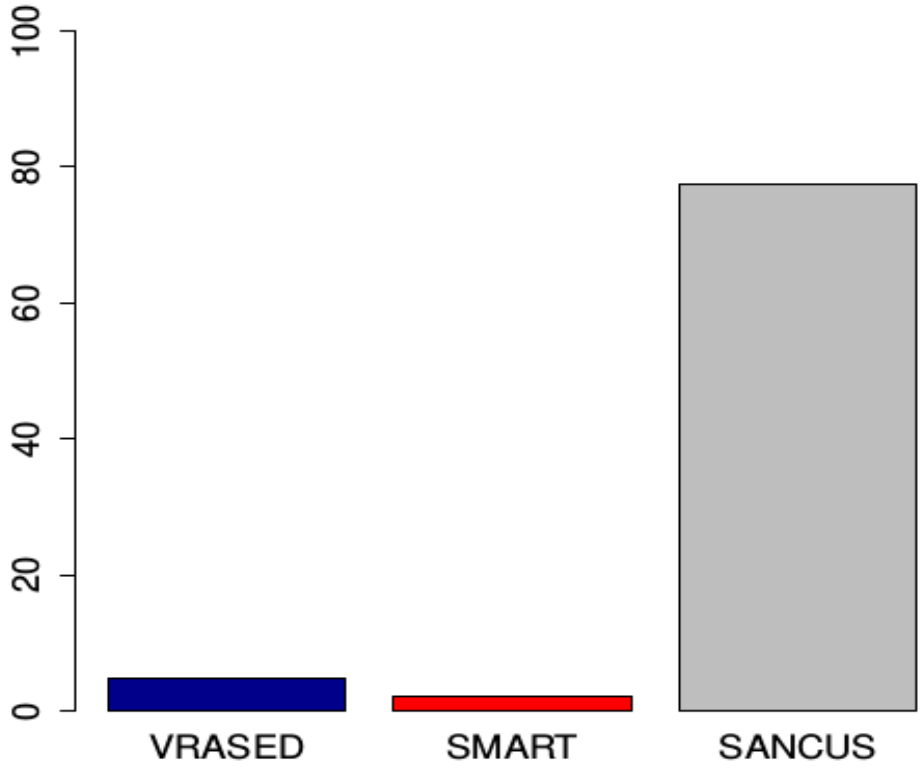
Implementation and Performance

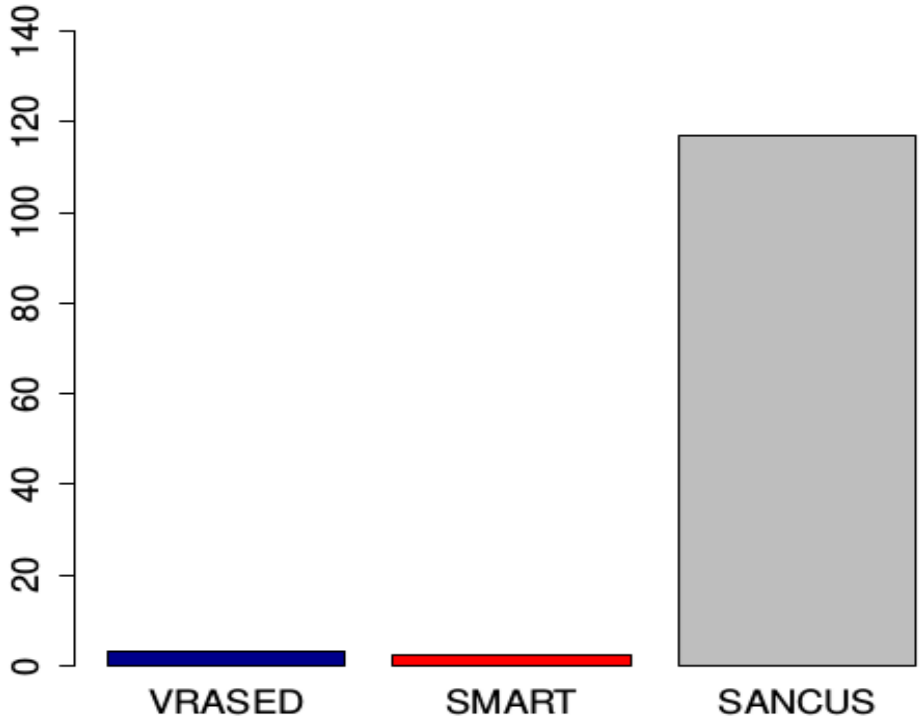
Implementation

- VRASED prototype on Open Cores OpenMSP430 Verilog Design
- Synthesized on Basys3 FPGA



Performance

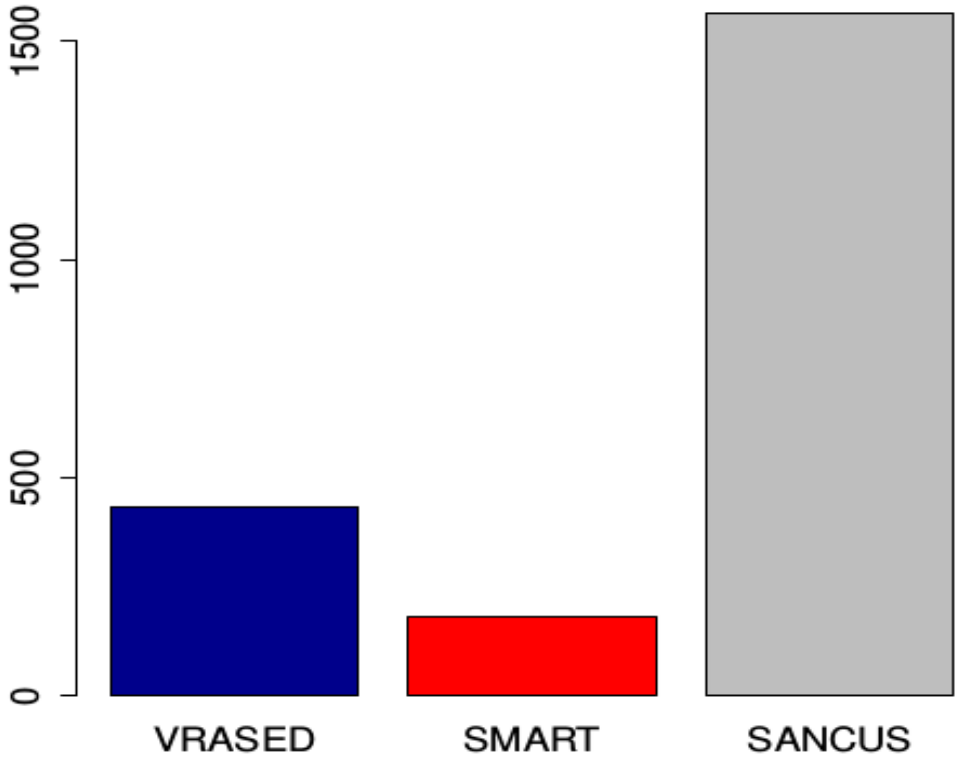




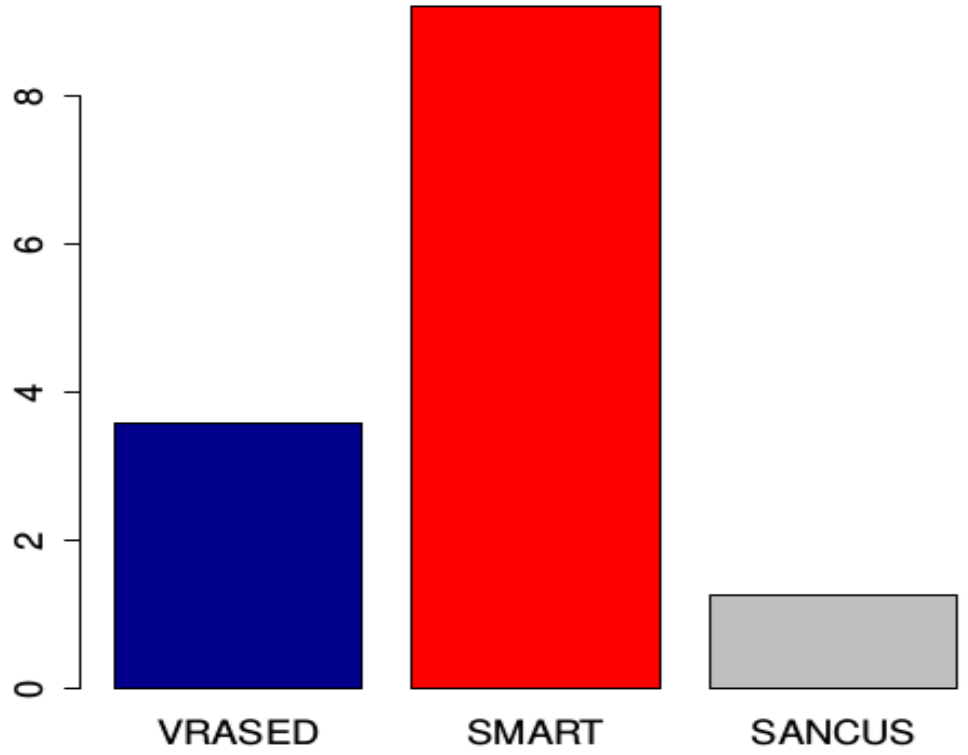
(a) Additional HW overhead (%) in Number of Look-Up Tables

(b) Additional HW overhead (%) in Number of Registers

Performance



(c) Additional Verilog Lines of Code



(d) Time to attest 4KB (in millions of CPU cycles)

SUMMARY

Formally Verified RA:

- ❑ Important
- ❑ Has not been done before
- ❑ Hard
- ❑ But doable!

For More Info:

I. De Oliveira Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner and G. Tsudik,
VRASED: A Verified Hardware/Software Co-Design for Remote Attestation,
Usenix Security Symposium (Usenix'Sec), 2019.

I. De Oliveira Nunes, K. Eldefrawy, N. Rattanavipanon and G. Tsudik,
PURE: Using Verified Remote Attestation to Obtain Proofs of Update, Reset and Erasure in Low-End Embedded Systems,
International Conference On Computer Aided Design (ICCAD), 2019.

K. Eldefrawy and G. Tsudik,
Advancing Secure Remote Attestation via Automated Formal Verification of Designs and Synthesis of Executables,
ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSEC), 2019.

What's next?

- Extend VRASED verified TCB to use RA for Provable Code Execution

(Done: https://www.usenix.org/conference/usenixsecurity20/presentation/nunes, USENIX Sec 2020)

- Formal Verification of other hybrid architectures aimed at medium/higher-end devices

(On-going!)

- Formal Verification of collective (e.g., swarm) attestation

(somewhat done: https://ieeexplore.ieee.org/document/8885355/, ICDCS 2019)