A Security Kernel for Protected Module Architectures

Alexandru – Madalin Ghenea
Master of Engineering: Computer Science

Promotor:
Prof. **Frank Piessens**
Advisors:
Jan Tobias Mühlberg
Jo Van Bulck

Introduction

- Growing trend towards Internet of Things
- Large variety of application domains:
  - Automotive
  - Medical applications
  - Home automation
  - Industrial control systems
  - Electronic payment
- Need for security solutions for networked, resource-constrained embedded systems

Protected Module Architectures (PMAs)

- Reduced Trusted Computing Base (TCB)
- Small TCB, isolation, key derivation => PMA
- Large spectrum of PMA solutions
  - Both for high end and low end devices
  - Software and hardware solutions

Sancus

- Security architecture for resource-constrained networked embedded devices
- Strong security guarantees with only hardware TCB
- Dedicated C compiler
- FPGA prototype based on MSP430 processor

Noorman, Job, et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices." (2017).

Sancus security guarantees

- Software Module (SM) isolation
  - Program-Counter Based Memory Access Control (PCBMAC)
  - Single entry point per module
  - Isolated stacks
  - **protect** layout, SP
  - **unprotect**

Noorman, Job, et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices." (2017).

Sancus security guarantees

- Remote attestation
  - **encrypt** plaintext, associated data, ciphertext (output), tag (output) [,key]
  - **decrypt** ciphertext, associated data, tag, plaintext (output),[,key]

Noorman, Job, et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices." (2017).

Sancus security guarantees

- Local attestation
  - **attest** address, expected hash
  - **get-id** address
  - **attest-caller**
  - **get_caller_id**

Noorman, Job, et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices." (2017).

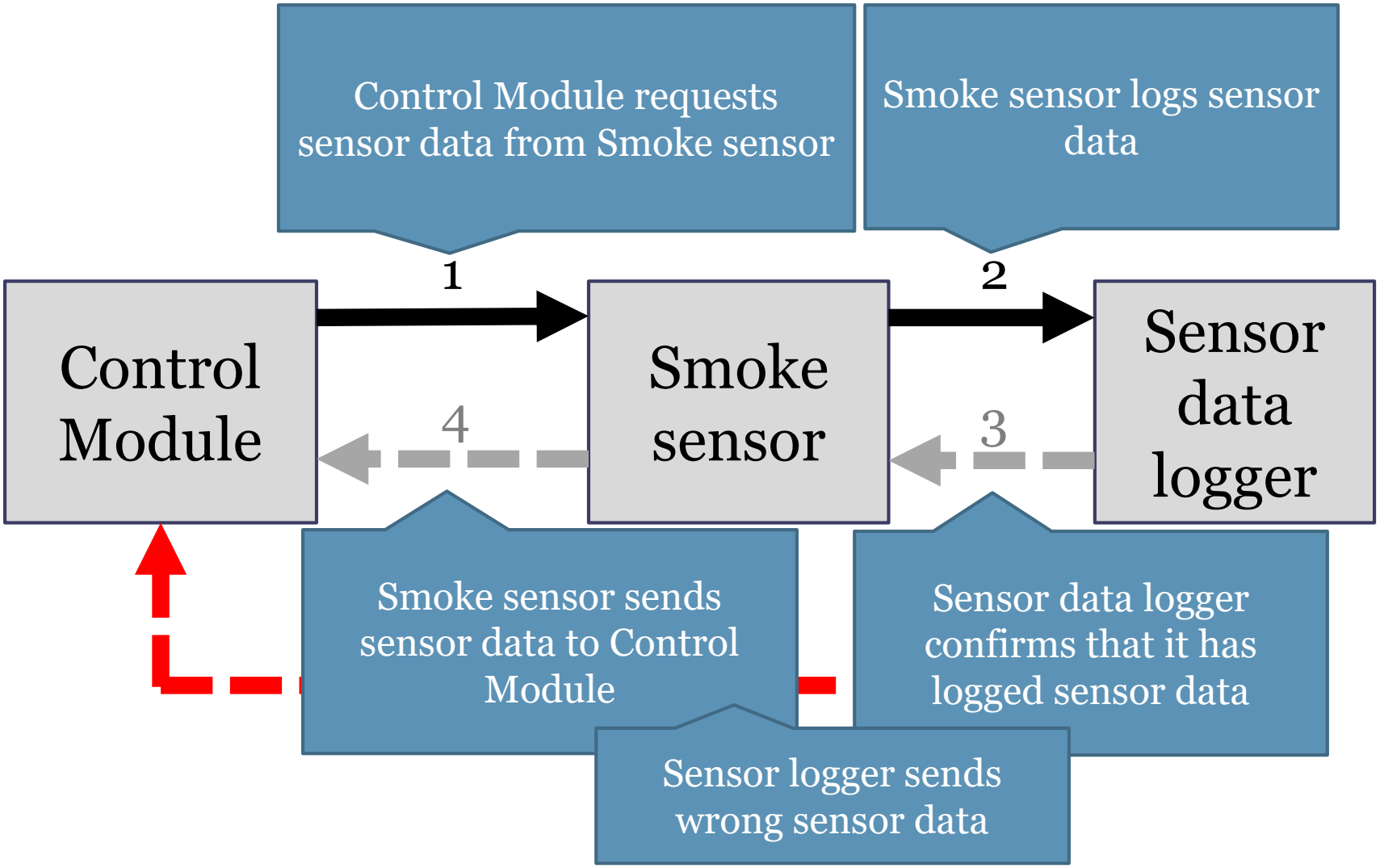
Sancus potential disadvantages

- Susceptible to call-stack shortcutting attacks
- Increased hardware costs
- Cryptographic instructions not interruptible
- Security primitives cannot be modified without hardware changes

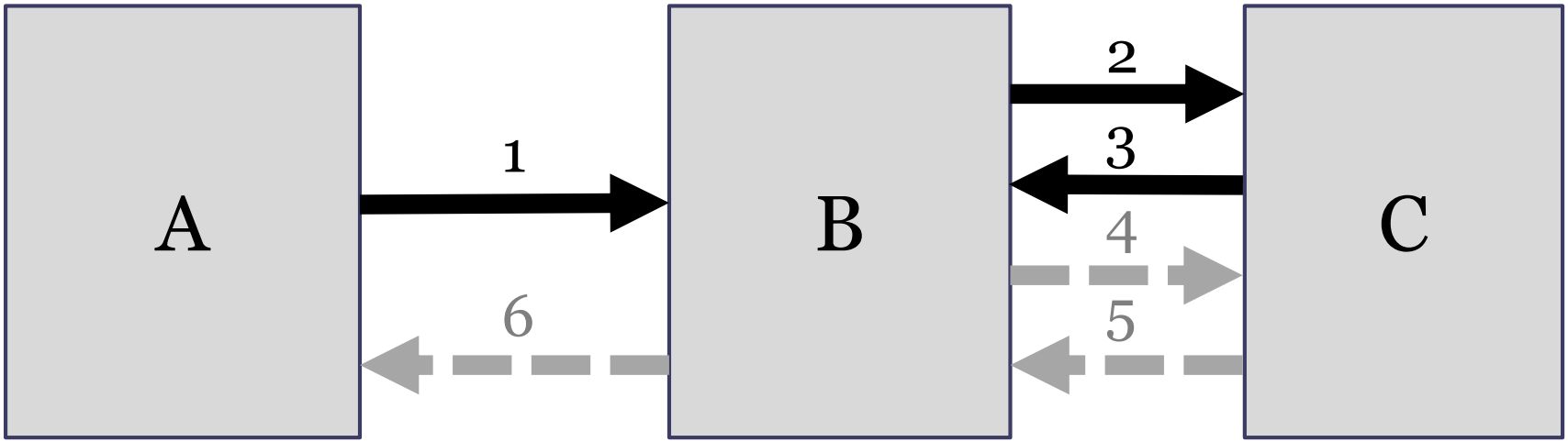
Sancus Security kernel - Hypothesis

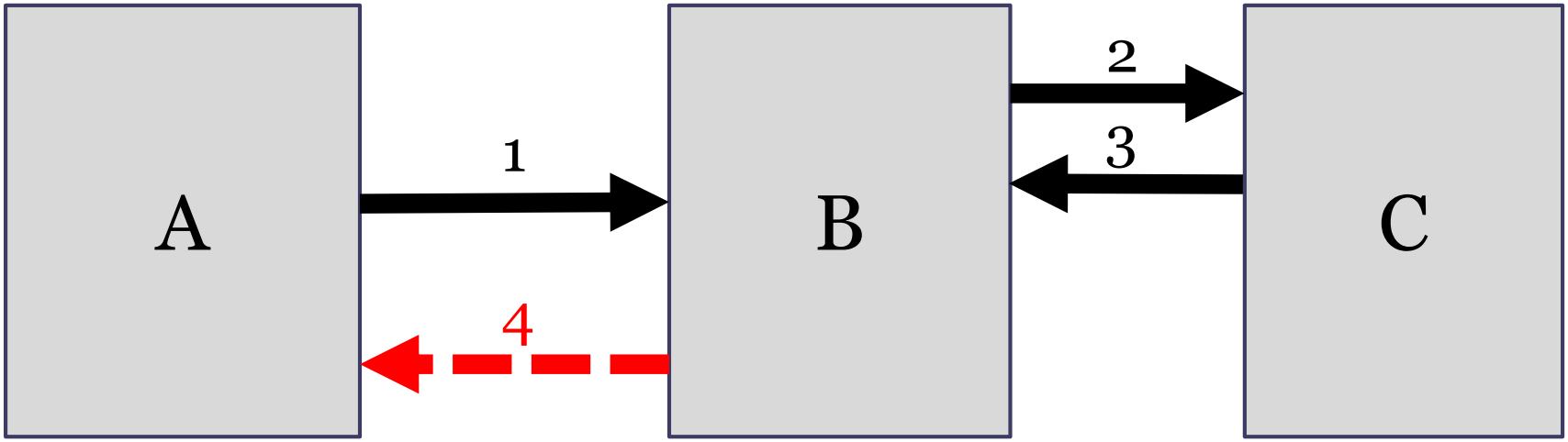
- Study the feasibility of creating a security kernel that protects against call-stack shortcutting attacks
- Study properties obtained from transferring the cryptographic component from hardware to software while trying to maintain the same security guarantees

Call-stack shortcutting attack



Call-stack shortcutting attack

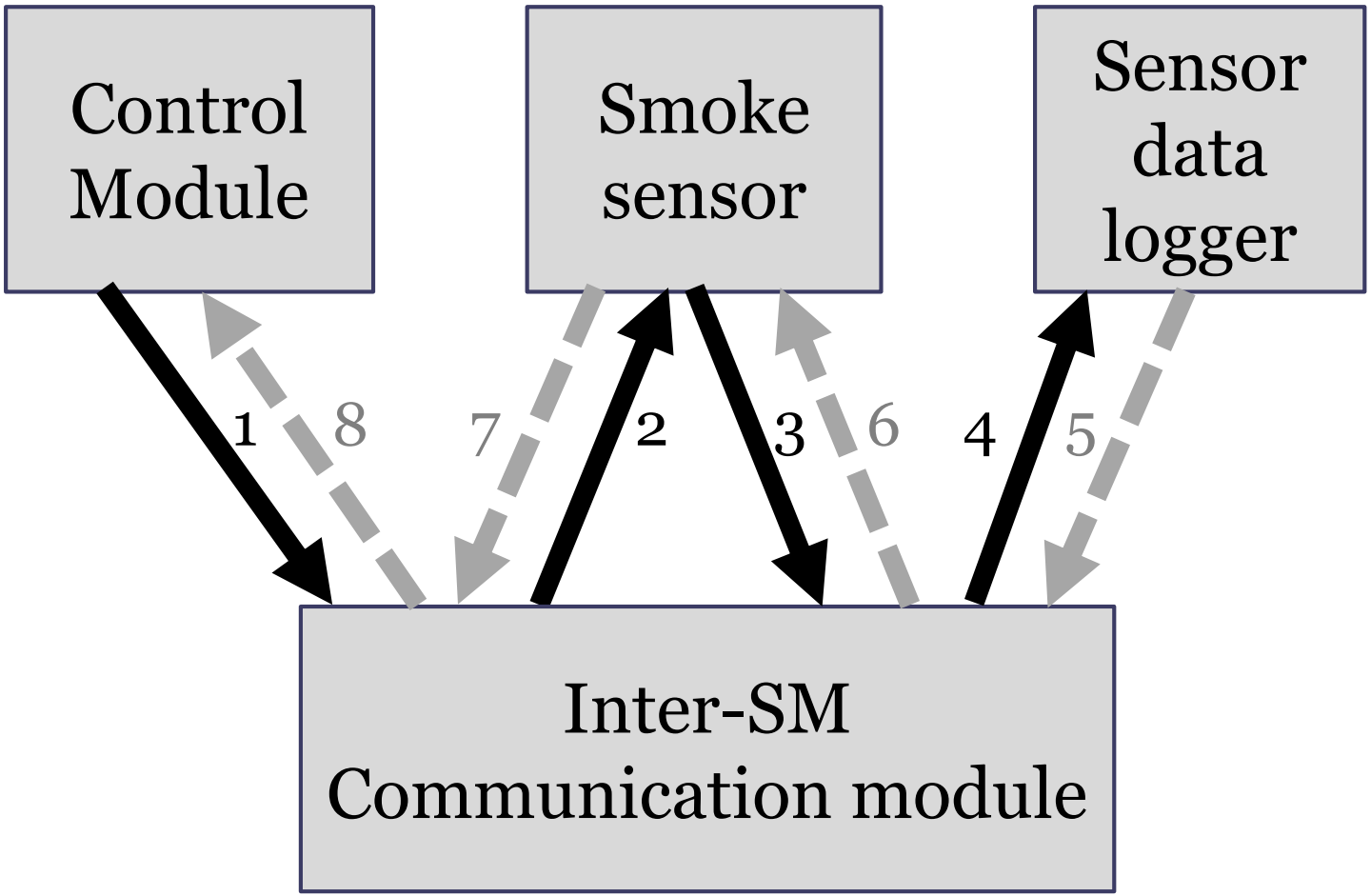




Implementation

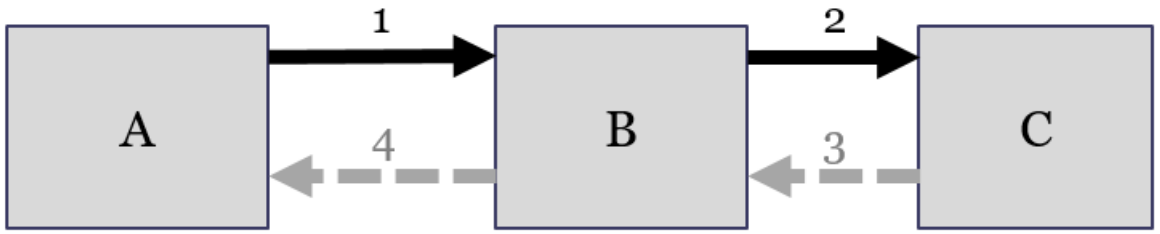
- Inter-SM communication component (ISMC) that protects against call-stack shortcutting attacks.
- Shadow call stack implementation
- For each Inter-SM call:
  - The Caller ID and return address of the module pushed in the shadow stack of the ISMC
  - After the callee returns to ISMC, the return address is popped and control is returned to the caller
- All Inter-SM communication is done via the kernel.
- Builds upon existing primitives:
  - **get-id** address
  - **get_caller_id**

Call-stack shortcutting - solution



Evaluation of ISMC

- Sancus platform with MSP430 running at 20 MHz with 64 bit SPONGENT
- Benchmarking using 3 scenarios:
  - A simple call between 2 SMs
  - Cascade call with 3 SMs
  - Scenario 2
- Comp ion of Sancu
- Meas Boot



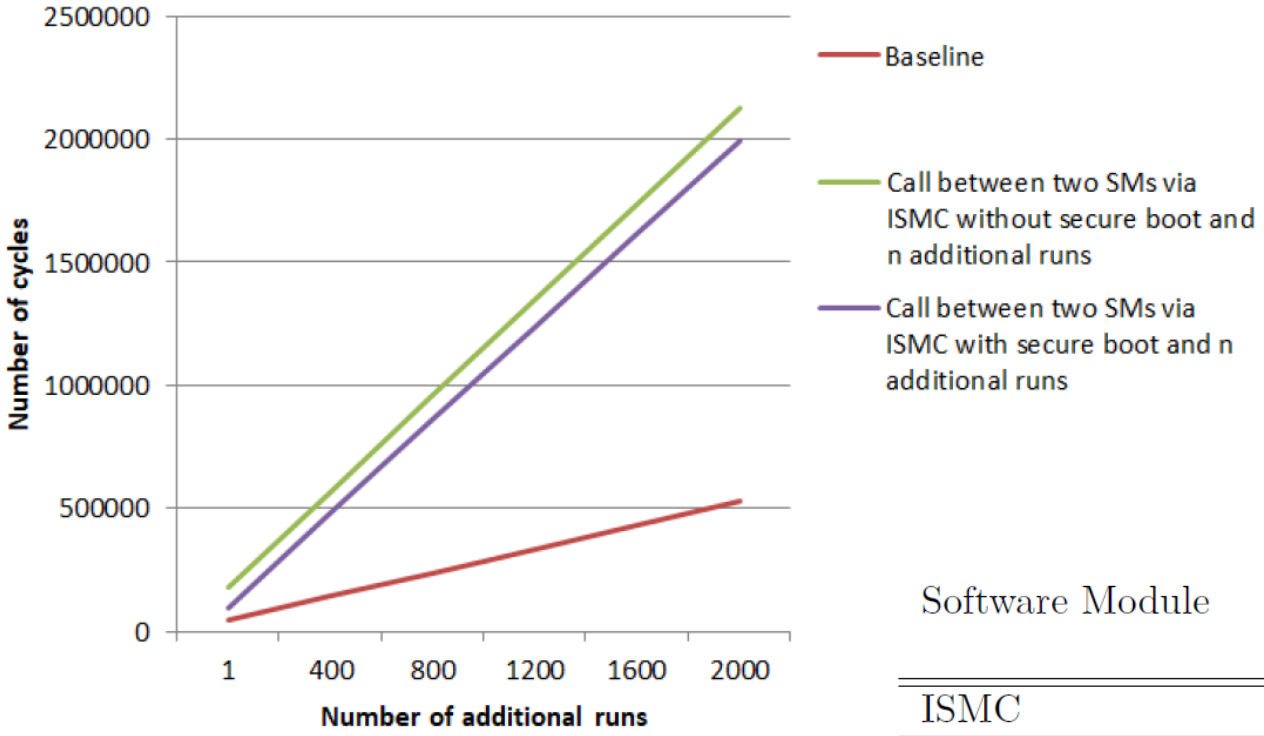
Evaluation of ISMC

Benchmark	Init. Overhead	First run overhead	Additional run overhead
Call between two SMs via ISMC with Secure Boot	50,871 (156%)	3,174 (26%)	705 (288%)
Call between two SMs via ISMC without Secure Boot	53,847 (165%)	80,792 (680%)	729 (388%)
Cascade call with 3 SMs via ISMC with Secure Boot	67,053 (138%)	20,748 (76%)	1,676 (346%)
Cascade call with 3 via ISMC without Secure Boot	67,983 (140%)	132,495 (489%)	1,696 (350%)
Scenario 2 via ISMC with Secure Boot	77,655 (142%)	47,929 (95%)	3,698 (352%)
Scenario 2 via ISMC without Secure Boot	78,213 (143%)	195,317 (387%)	3,738 (356%)

Evaluation of ISMC

Benchmark	Init. Overhead	First run overhead	Additional run overhead
Call between two SMs via ISMC with Secure Boot	2.55 ms	0.16 ms	0.03 ms
Call between two SMs via ISMC without Secure Boot	2.70 ms	4.04 ms	0.03 ms
Cascade call with 3 SMs via ISMC with Secure Boot	3.35 ms	1.04 ms	0.08 ms
Cascade call with 3 via ISMC without Secure Boot	3.4 ms	6.62 ms	0.08 ms
Scenario 2 via ISMC with Secure Boot	3.89 ms	2.39 ms	0.18 ms
Scenario 2 via ISMC without Secure Boot	3.92 ms	9.76 ms	0.18 ms

Evaluation of ISMC



Software Module	Source LOC	Binary size (B)
ISMC	177	4966
Average SM	291	2156

Mühlberg, Jan Tobias, et al. "An implementation of a high assurance smart meter using protected module architectures." *IFIP International Conference on Information Security Theory and Practice*. Springer International Publishing, 2016.

Software local Attestation

- Does not use any hardware cryptographic primitives
- Registration based mechanism
  - *register_sm*
  - *is_registered*
  - *is_registered_with_layout*
- SM protection enabled by the security kernel
- Expected hashes stored in the security kernel
- Measurement computed only once per SM

Software local Attestation

- Possibility to change measurement implementation
- Step towards interruptibility
- Sufficient for remote attestation if secure communication is provided

Evaluation

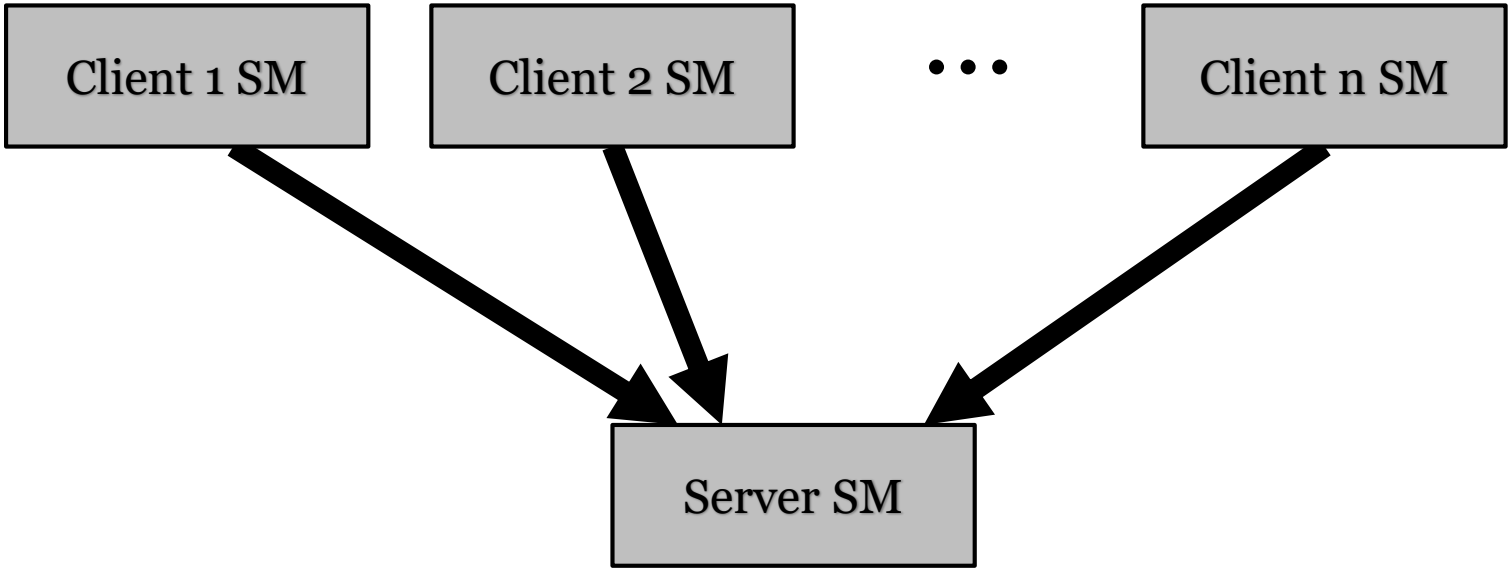
- Baseline: Sancus platform with MSP430 running at 20 MHz with 64 bit SPONGENT implementation
- Software attestation using:
  - SPONGENT 128 bit security
  - SHA-2 256 bit

Hashing micro-benchmarks

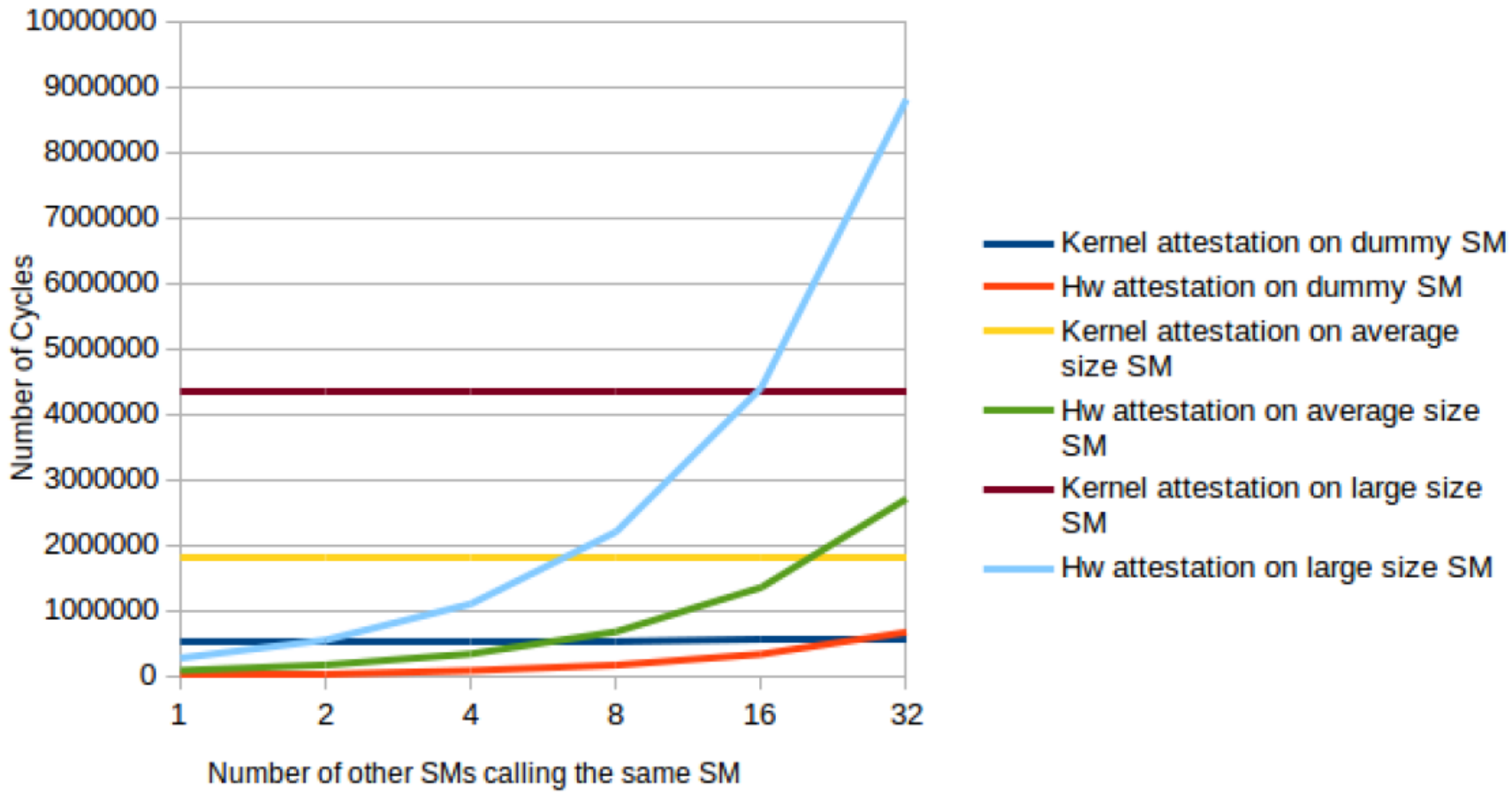
	Cycles		
Data size	SPONGENT Hw. Sancus	SPONGENT Sw. Sancus	SHA-2 Sancus
256 Bytes hash	25,000	3,574,090,309	211,739
512 Bytes hash	47,500	6,941,412,200	377,387
1024 Bytes hash	92,000	13,676,055,952	708,732

	Milliseconds		
Data size	SPONGENT Hw. Sancus	SPONGENT Sw. Sancus	SHA-2 Sancus
256 Bytes hash	1.25	178,704	11
512 Bytes hash	2.37	374,070	19
1024 Bytes hash	4.6	683,802	35

Software Attestation macro-benchmarks



Software Attestation benchmark SPONGENT hw. vs SHA-2 sw.



Conclusions - ISMC

- ISMC protects against call-stack shortcutting attacks
- ISMC adds significant overhead, but could be used for many applications

Conclusions – Software Local Attestation

- Software local attestation overhead depends on the measurement implementation
- Software attestation can be more efficient when using large number of callers
- Reduced hardware costs
- Step towards interruptibility
- Possibility to change measurement implementation after deployment

Future work

- Evaluate the software local attestation mechanism with more measurement implementations
- Extend kernel with a remote attestation mechanism
- Extend kernel to provide compatibility with real-time applications

Thank you for your attention!