

Secure Resource Sharing for Embedded Protected Module Architectures

Jo Van Bulck

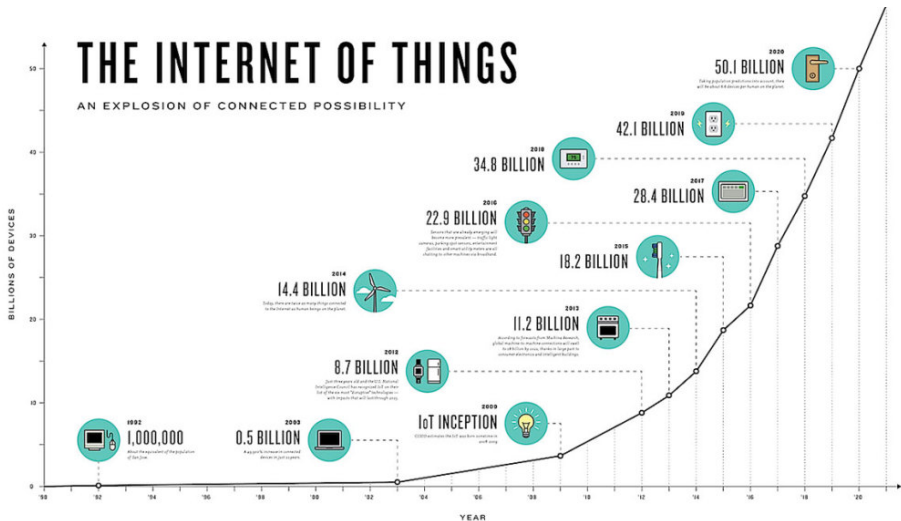
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Belgium

MSc Thesis Presentation

BELCLIV-CLUSIB, April 21, 2017

*“Internet of Things [in]security
keeps me up at night.”*

— Rob Joyce, NSA’s Tailored Access Operations chief
(MIT Technology Review, January 2016).

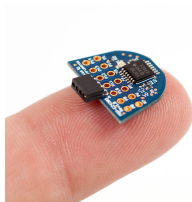


Source: <https://www.ncta.com/platform/industry-news/infographic-the-growth-of-the-internet-of-things/>

Motivation: Embedded Device Security

TI MSP430: low-cost, low-power computing

- Runs ~13 years on a single AA battery [Sea08]
- *Single-address-space* without memory protection
- Attacker can modify all code and data, and forge sensor readings or node identity

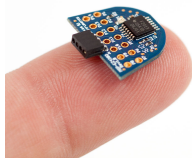


<http://martybugs.net/electronics/msp430/>

Motivation: Embedded Device Security

TI MSP430: low-cost, low-power computing

- Runs ~13 years on a single AA battery [Sea08]
- *Single-address-space* without memory protection
- Attacker can modify all code and data, and forge sensor readings or node identity



<http://martybugs.net/electronics/msp430/>

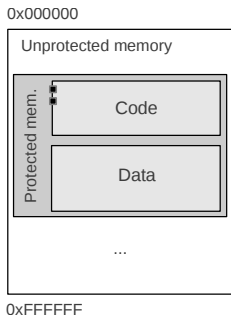
Protected Module Architectures: isolation and attestation

- Minimal (hardware-only) Trusted Computing Base
- Server/desktop: Intel SGX, ARM TrustZone
- Low-end embedded: SMART, TrustLite, TyTAN, Sancus

Maene et al.: "Hardware-Based Trusted Computing Architectures for Isolation and Attestation", 2017 [MGDC⁺17].

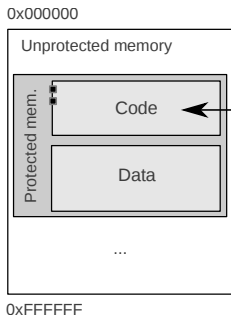
Background: Protected Module Architectures

- **Isolated execution** in a single-address-space



Background: Protected Module Architectures

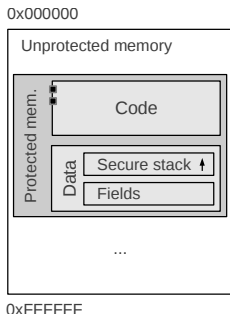
- **Isolated execution** in a single-address-space
- **Program counter** based access control



From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwX
Unprotected / other SM	r-x	r--	---	rwX

Strackx et al.: "Efficient Isolation of Trusted Subsystems in Embedded Systems", 2010 [SPP10].

Background: Protected Module Architectures



- **Isolated execution** in a single-address-space
- **Program counter** based access control
- Secure fully abstract **compilation**

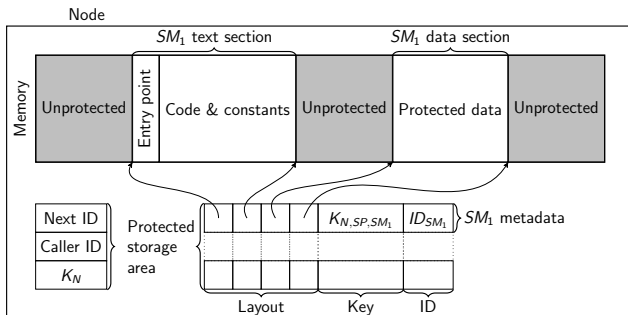
From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwX
Unprotected / other SM	r-x	r--	---	rwX

Strackx et al.: "Efficient Isolation of Trusted Subsystems in Embedded Systems", 2010 [SPP10].

Agten et al.: "Secure Compilation to Modern Processors", 2012 [ASJP12].

Sancus PMA [NAD⁺13, NVBM⁺17]

Zero-software TCB: extended openMSP430 instruction set



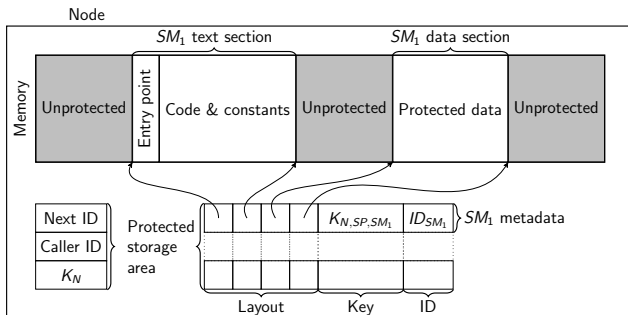
Noorman et al.: "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices", 2017 [NVBM⁺17].

Sancus PMA [NAD⁺13, NVBM⁺17]

Zero-software TCB: extended openMSP430 instruction set

SM == unit of **isolation** + **authentication**:

- Remote attestation / secure linking
- Hardware-level cryptographic key + ID per SM

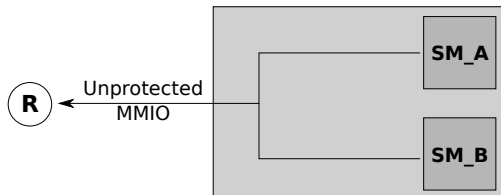


Noorman et al.: "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices", 2017 [NVBM⁺17].

Secure Resource Sharing

PMAAs assume the presence of an attacker:

- 😊 Strong **HW-enforced security** guarantees
- 😞 No **secure sharing** of platform resources



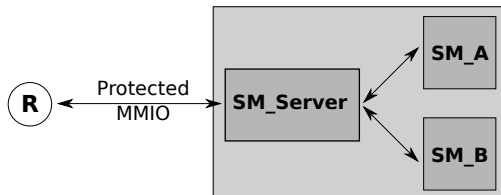
Secure Resource Sharing

PMAAs assume the presence of an attacker:

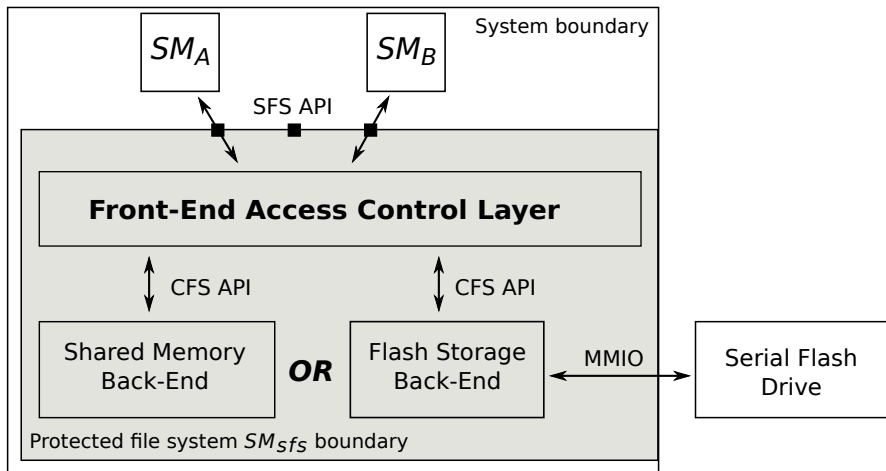
- ☺ Strong **HW-enforced security** guarantees
- ☹ No **secure sharing** of platform resources

⇒ Self-protecting “OS” modules to supplement HW:

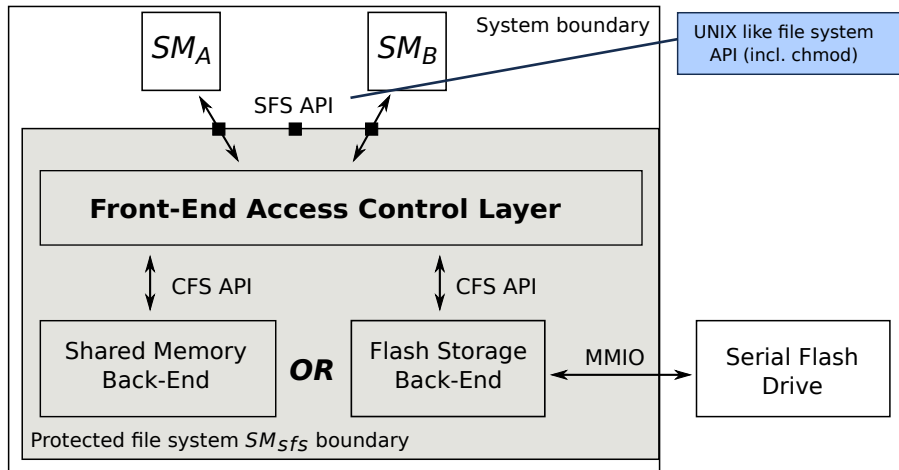
- ↔ Monolithic privileged kernel
- ~ Extreme microkernel idea



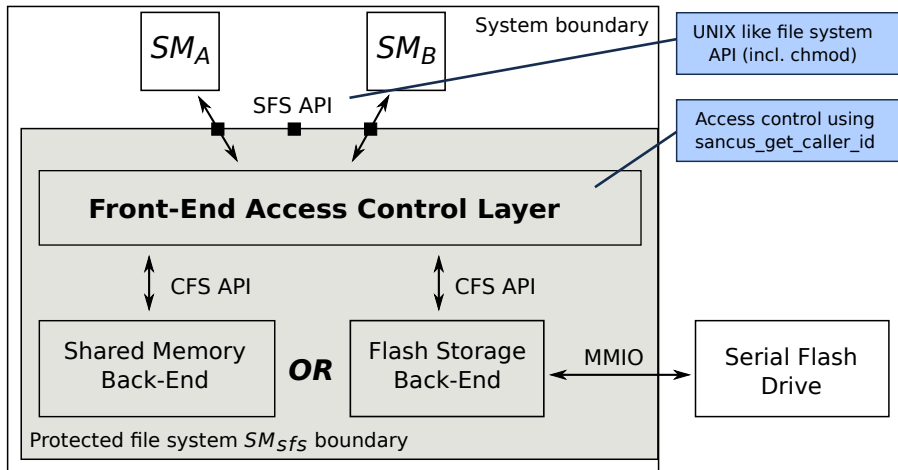
Sancus File System (SFS)



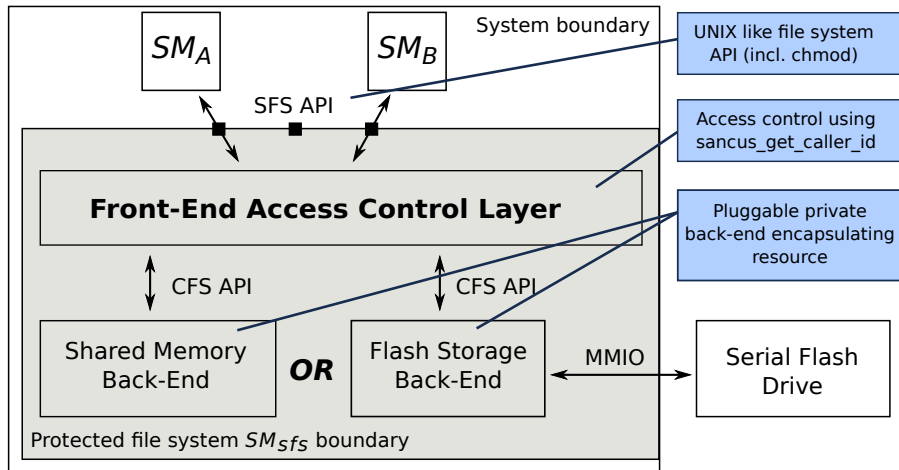
Sancus File System (SFS)



Sancus File System (SFS)



Sancus File System (SFS)



Example Scenario

```
[clientA] revoking B permissions
[sfs-ram] INFO::sfs_chmod: trying to modify ACL for file 'a'
[sfs-ram] WARNING::ACL entry currently open; setting to SFS_NIL
[sfs-ram] INFO::sfs_chmod: trying to modify ACL for file 'b'
[sfs-ram] WARNING::ACL entry currently open; setting to SFS_NIL
[sfs-ram] INFO::sfs_dump: dumping global protected ACL data structures:
-----
FILE with name 'b' at 0x554; open_count = 2; next_ptr = 0x54c
  PERM (2, 0xff) at 0x586; file_ptr = 0x554; next_ptr = 0x58e
  PERM (3, 0x00) at 0x58e; file_ptr = 0x554; next_ptr = 0
FILE with name 'a' at 0x54c; open_count = 2; next_ptr = 0
  PERM (2, 0xff) at 0x576; file_ptr = 0x54c; next_ptr = 0x57e
  PERM (3, 0x00) at 0x57e; file_ptr = 0x54c; next_ptr = 0
-----
[sfs-ram] INFO::sfs_dump: dumping global protected file descriptor cache:
(0, 0x576); (1, 0x586); (2, 0x57e); (3, 0x58e); (4, 0x0); (5, 0x0); (6, 0x0); (7, 0x0);
[clientA] accessing B files (shouldn't work)
[clientB] accessing bunch of files
[sfs-ram] INFO::sfs_getc: read a char from file with fd 2
[sfs-ram] ERROR::permission check failed.
[sfs-ram] INFO::sfs_getc: read a char from file with fd 3
[sfs-ram] ERROR::permission check failed.
[sfs-ram] INFO::sfs_putc: write a char to file with fd 3
[sfs-ram] ERROR::permission check failed.
[clientA] closing b files
```

Discussion

⇒ *Generic resource sharing mechanism*

SW-based **access control** guarantees:

- Build upon HW primitives (isolation + authentication)
- Non-persistent file protection

Confined and explicit **TCB**:

- Principle of least privilege (~ microkernel)
- Attestable via `sancus_verify`

Secure Multithreading

Thread == synchronous control flow within address space

- Local thread context on **call stack**
- Conventional OS kernel saves **CPU state** on interrupt

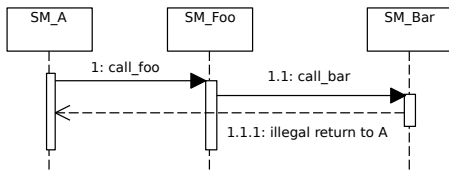
Secure Multithreading

Thread == synchronous control flow within address space

- Local thread context on **call stack**
- Conventional OS kernel saves **CPU state** on interrupt

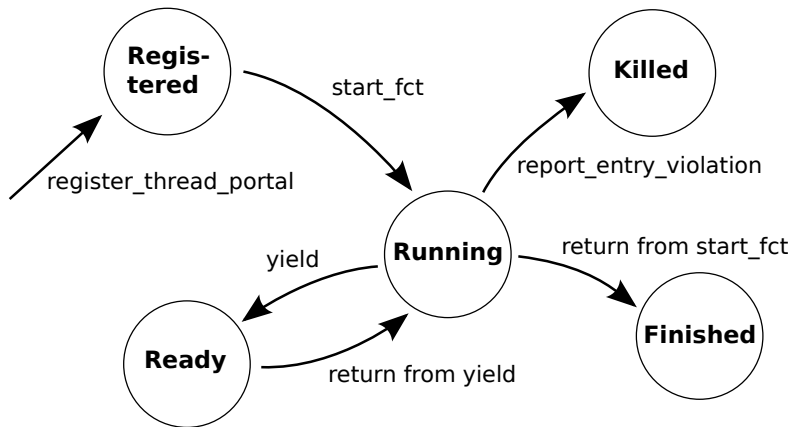
PMA multithreading challenges:

- Unit of **threading** >> **SM**
- Compiler-generated `sm_entry` **asm stubs**
- Inter-SM **call/return flow integrity** guards



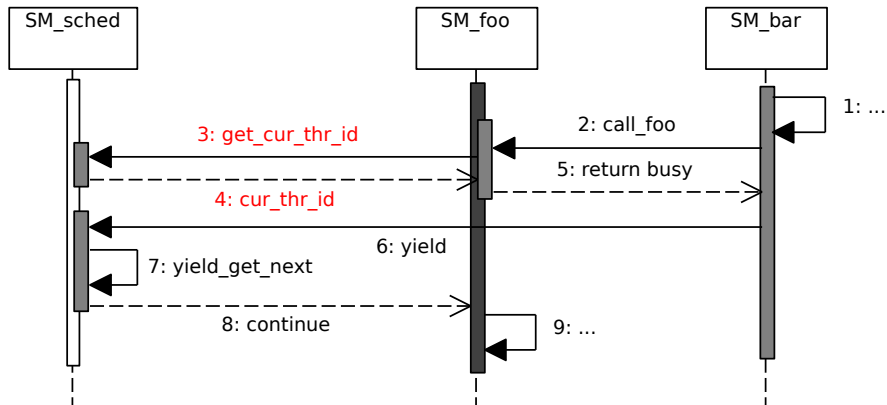
Cooperative Scheduler Prototype

⇒ Scheduler SM interleaves *multiple control flows*



Threading-aware SMs

⇒ SM maintains at most *one internal call stack per thread-ID*



Example Scenario

```

[foo] now calling bar
[enter_bar] self_id = 4 ; caller_id = 3
[bar] now bypassing foo and calling 'a' directly
    [sched] SM 2 reports an entry violation by SM 4
    [sched] I will kill logical thread 1 now
    [sched] now running logical thread:
    THREAD with thr_id 2 and state REG at 0x4f2 ; next_ptr = 0x502
           sm_id = 3; pub_start = 0x99e0 ; entry = 0
[enter_foo] self_id = 3 ; caller_id = 1
[foo] dumping scheduler
[sched] dumping internal state; I have SM ID 1
ready queue:
-----
THREAD with thr_id 3 and state REG at 0x502 ; next_ptr = 0
           sm_id = 4; pub_start = 0xa978 ; entry = 0
-----
done queue:
-----
THREAD with thr_id 1 and state KILLED at 0x4e2 ; next_ptr = 0
           sm_id = 2; pub_start = 0x9c60 ; entry = 0x1
-----
current thread:
THREAD with thr_id 2 and state RUNNING at 0x4f2 ; next_ptr = 0x502
           sm_id = 3; pub_start = 0x99e0 ; entry = 0

```

Discussion

⇒ *Isolated cross-SM control flow threads*

Division of responsibilities:

- Hardware: SM **confidentiality/integrity** (memory isolation)
- Compiler: entry **call stack consistency** (SM call/return)
- Unprivileged scheduler: **scheduling policy** (temporal isolation)

Future and Ongoing Work

Real-time secure multitasking [VBNMP16]:

- Hardware primitives: secure interrupts + atomicity monitor
- Compiler: SM-internal multithreading
- Preemptive scheduler: FreeRTOS prototype

Efficient resource sharing:

- Controlling access to a multi-device I/O bus
- Hardware mechanisms for inter-SM sharing

Case studies:

- Smart metering [MCM⁺16]
- Automotive computing [Müh17]

Thank you! Questions?

<https://distrinet.cs.kuleuven.be/software/sancus/>

<https://github.com/jovanbulck/thesis-src/>

<https://distrinet.cs.kuleuven.be/software/sancus/publications/vanbulck15thesis.pdf>

References I



T. Alves and D. Felton.

Trustzone: Integrated hardware and software security.
ARM white paper, 3(4):18–24, 2004.



P. Ageton, R. Strackx, B. Jacobs, and F. Piessens.

Secure compilation to modern processors.
In *Computer Security Foundations Symposium*, pp. 171–185. IEEE, 2012.



F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl.

TyTAN: Tiny trust anchor for tiny devices.
In *Design Automation Conference (DAC 2015)*, pp. 1–6. IEEE, 2015.



R. De Clercq, F. Piessens, D. Schellekens, and I. Verbauwhede.

Secure interrupts on low-end microcontrollers.
In *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, pp. 147–152. IEEE, 2014.



K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito.

SMART: Secure and minimal architecture for (establishing a dynamic) root of trust.
In *NDSS*, vol. 12, pp. 1–15. Internet Society, 2012.



P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan.

TrustLite: A security architecture for tiny embedded devices.
In *Proceedings of the Ninth European Conference on Computer Systems*, pp. 10:1–10:14. ACM, 2014.

References II



F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar.

Innovative instructions and software model for isolated execution.

In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, pp. 10:1–10:1. ACM, 2013.



J. T. Mühlberg, S. Cleemput, A. M. Mustafa, J. Van Bulck, B. Preneel, and F. Piessens.

Implementation of a high assurance smart meter using protected module architectures.

In *10th WISTP International Conference on Information Security Theory and Practice (WISTP'16)*, vol. 9895 of LNCS, pp. 53–69. Springer, 2016.



P. Maene, J. Götzfried, R. De Clercq, T. Müller, F. Freiling, and I. Verbauwhede.

Hardware-based trusted computing architectures for isolation and attestation.

IEEE Transactions on Computers, PP(99), 2017.



J. T. Mühlberg.

A new security architecture for networked embedded devices.

eeNews Europe Automotive, June 2017.

<http://www.eenewsautomotive.com/design-center/new-security-architecture-networked-embedded-devices>.



J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens.

Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base.

In *22nd USENIX Security Symposium*, pp. 479–494. USENIX Association, 2013.



J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.

Sancus 2.0: A low-cost security architecture for IoT devices.

ACM Transactions on Privacy and Security (TOPS), 2017.

References III



M. Seaman.

Powering an msp430 from a single battery cel.
Technical report, Texas Instruments, September 2008.
<http://www.ti.com.cn/cn/lit/an/slaa398/slaa398.pdf>.



R. Strackx, F. Piessens, and B. Preneel.

Efficient isolation of trusted subsystems in embedded systems.
In *Security and Privacy in Communication Networks*, pp. 344–361. Springer, 2010.



J. Van Bulck, J. Noorman, J. T. Mühlberg, and F. Piessens.

Secure resource sharing for embedded protected module architectures.
In *9th WISTP International Conference on Information Security Theory and Practice (WISTP'15)*, vol. 9311 of *LNCS*, pp. 71–87. Springer, 2015.



J. Van Bulck, J. Noorman, J. T. Mühlberg, and F. Piessens.

Towards availability and real-time guarantees for protected module architectures.
In *Companion Proceedings of the 15th International Conference on Modularity (MASS'16)*, pp. 146–151. ACM, 2016.