Time and Availability in Trusted Execution Environments

Fritz Alder, Gianluca Scopelliti, Sepideh Pouyanrad, Job Noorman, Jo Van Bulck, Frank Piessens (KU Leuven), **Jan Tobias Mühlberg** (ULB)
jan.tobias.muehlberg@ulb.be
Université libre de Bruxelles, Cybersecurity Research Center, Belgium

SoSySec Seminar at Inria Rennes, 2023-05-26

ULB UNIVERSITÉ LIBRE DE BRUXELLES

CYBERSECURITY RESEARCH CENTER

beams

Trusted Computing / Trusted Execution...

- Strong integrity protection and isolation for software components
- Software attestation: cryptographically bind protected software to the executing hardware, remotely verifiable
- Sealed storage: bind data to attested software

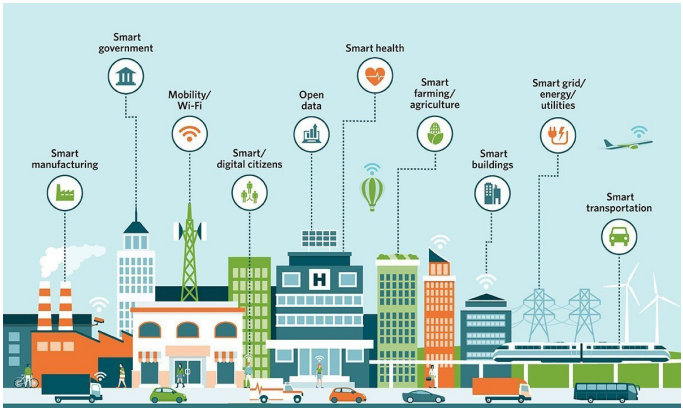
Trusted Computing / Trusted Execution...

- Strong integrity protection and isolation for software components
- Software attestation: cryptographically bind protected software to the executing hardware, remotely verifiable
- Sealed storage: bind data to attested software

...for mixed-criticality systems?

- Effective isolation of different criticalities?
- Real-time and progress guarantees?
- Does it improve safety?

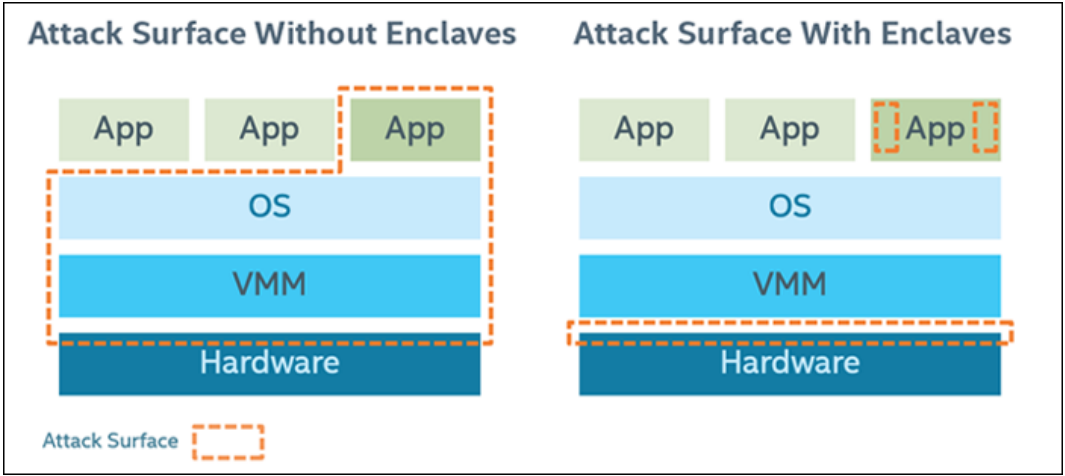
Security in Smart Environments



Bugs and vulnerabilities can hide anywhere: There are 150M lines of code in a modern car. Compartmentalisation can help with managing complexity and bug containment.

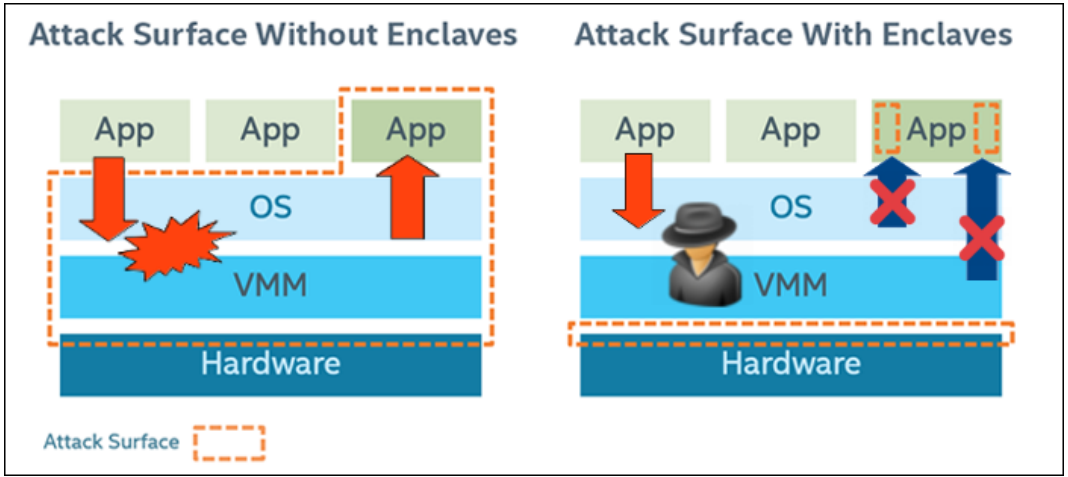
Image source: `https://internetofthingsagenda.techtarget.com/definition/smart-city`

Trusted Execution: Reducing the Application Attack Surface



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

Trusted Execution: Reducing the Application Attack Surface



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

Layered architecture $\leftrightarrow$ **hardware-only TCB**

Sancus: Strong and Light-Weight Embedded Security [NVBM+17]

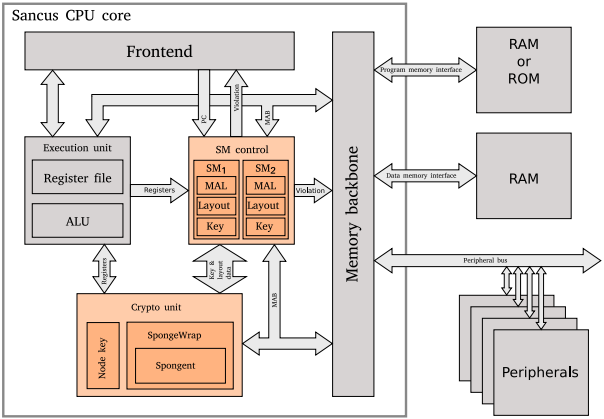
Extends openMSP430 with strong security primitives

- Software Component Isolation
- Cryptography & Attestation
- Secure I/O through isolation of MMIO ranges

Efficient

- Modular, $\leq$ 2 kLUTs
- Authentication in μs
- $+$ 6% power consumption

Cryptographic key hierarchy for software attestation



Isolated components are typically very small ($<$ 1kLOC)

Sancus is Open Source: https://distrinet.cs.kuleuven.be/software/sancus/

Sancus: Strong and Light-Weight Embedded Security [NVBM+17]

Extends openMSP430 with strong security primitives

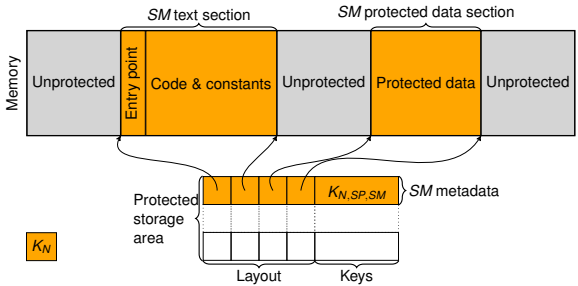
- Software Component Isolation
- Cryptography & Attestation
- Secure I/O through isolation of MMIO ranges

Efficient

- Modular, $\leq$ 2 kLUTs
- Authentication in μs
- $+$ 6% power consumption

Cryptographic key hierarchy for software attestation

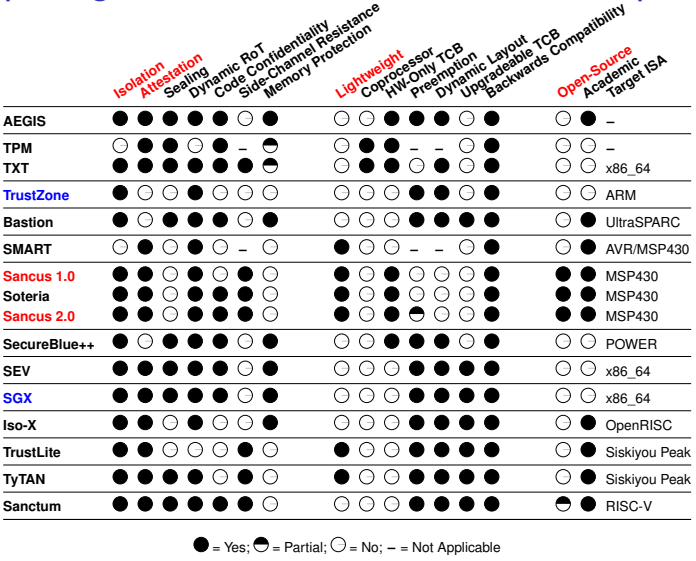
N = Node; SP = Software Provider / Deployer
SM = protected Software Module (== enclave)



Isolated components are typically very small ($<$ 1kLOC)

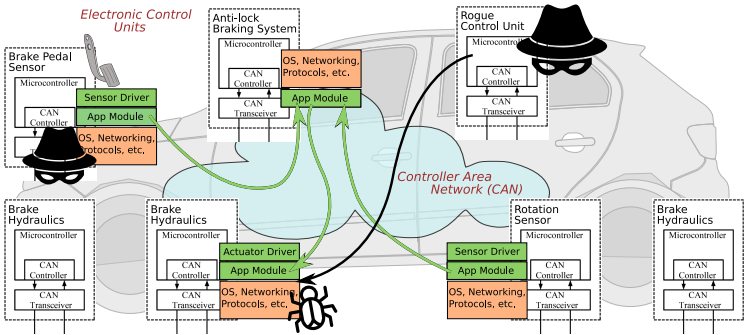
Sancus is Open Source: https://distrinet.cs.kuleuven.be/software/sancus/

Comparing Hardware-Based Trusted Computing Architectures



Adapted from "Hardware-Based Trusted Computing Architectures for Isolation and Attestation", Maene et al., IEEE Transactions on Computers, 2017. [MGdC+17]

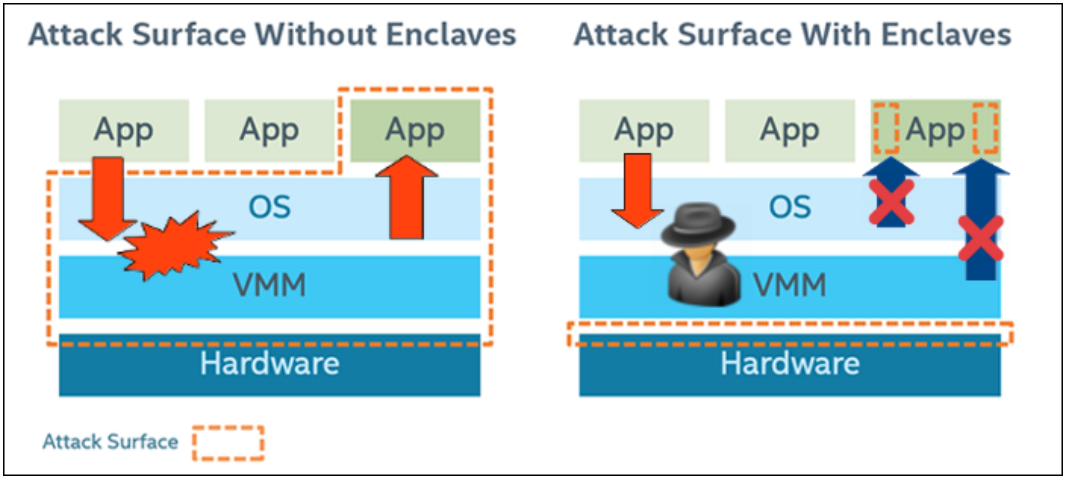
Secure Automotive Computing with TEEs [VBMP17]



- Message authentication
- Trusted Computing: software component isolation and cryptography
- Strong software security
- Applicable in automotive, ICS, IoT, . . .

Secure Automotive Computing with TEEs [VBMP17]

Trusted Execution: Reducing the Application Attack Surface



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

What do we trust for time, availability & safety? $\rightarrow$ a lot of software

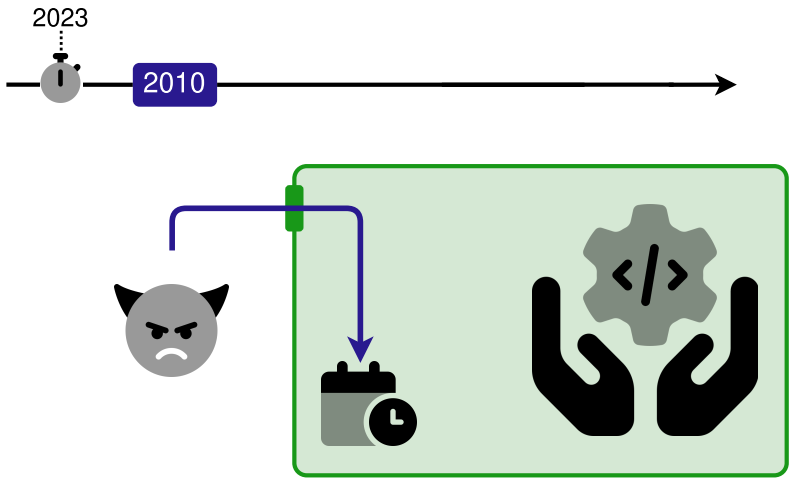
TEE Time [ASVBM23]

- Uses of time are common:
  - Certificate validity check
  - Rate limiting
  - Time-based policies, resource counting, DRM, ...
  - Sensing and actuation
- In many architectures, enclaves have no direct access to a clock

→ Time comes from or passes through the untrusted environment

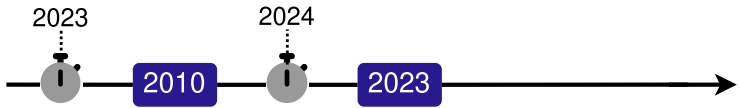
Getting reliable wall-clock time is hard!

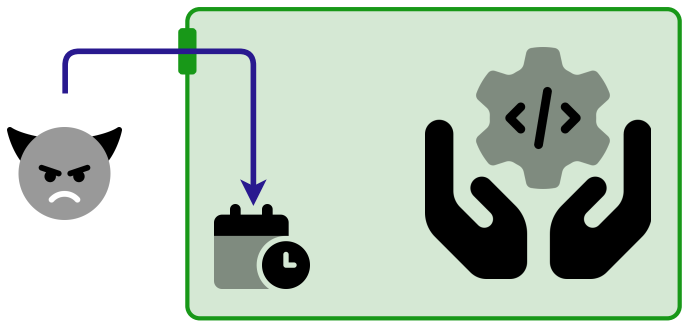
T_0: No guarantees on time



Getting reliable wall-clock time is hard!

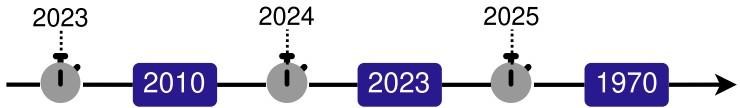
T_0: No guarantees on time

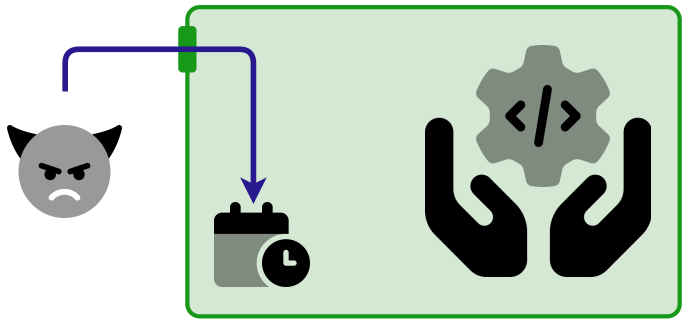




Getting reliable wall-clock time is hard!

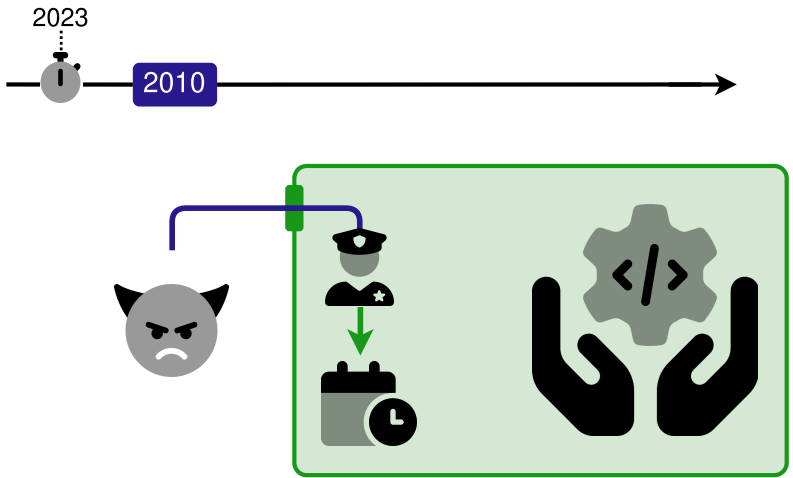
T_0: No guarantees on time





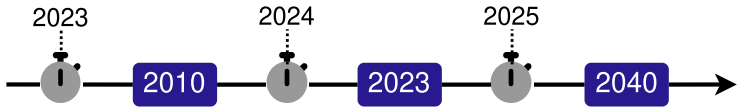
Getting reliable wall-clock time is hard!

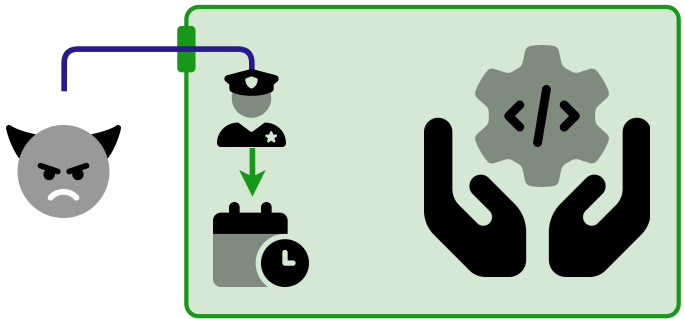
T_1: Time monotonically advances



Getting reliable wall-clock time is hard!

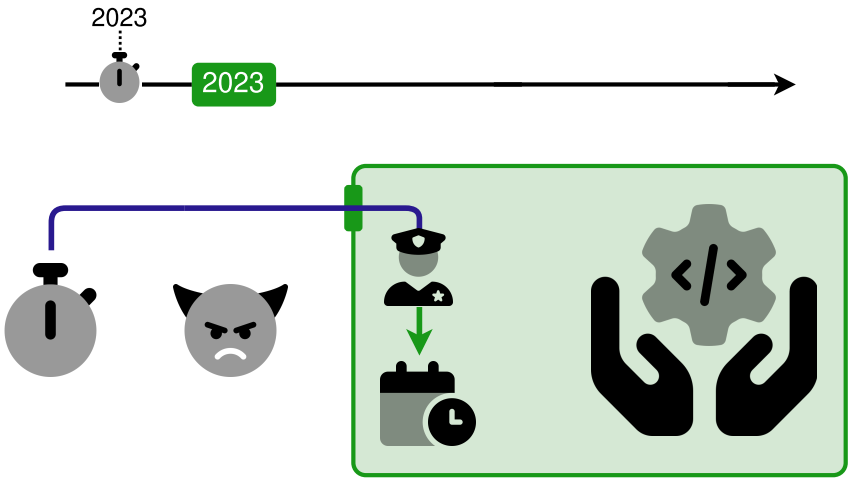
T_1: Time monotonically advances





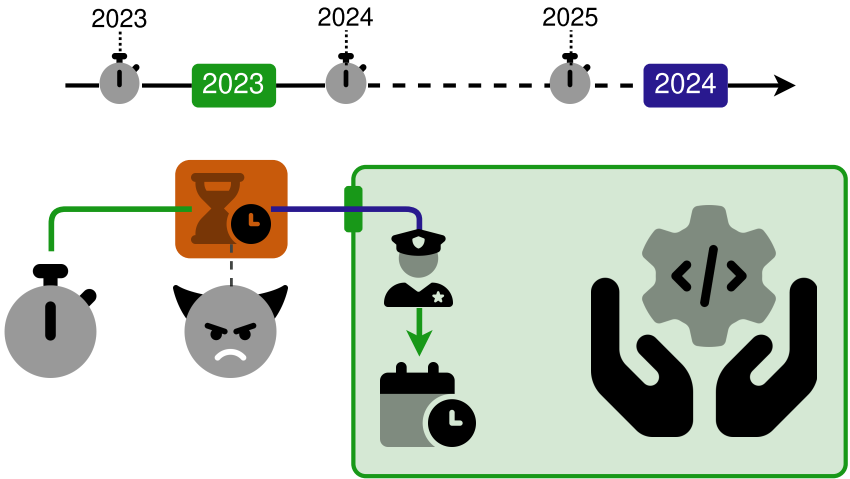
Getting reliable wall-clock time is hard!

T_2: Time moves at constant pace



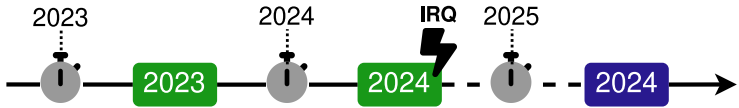
Getting reliable wall-clock time is hard!

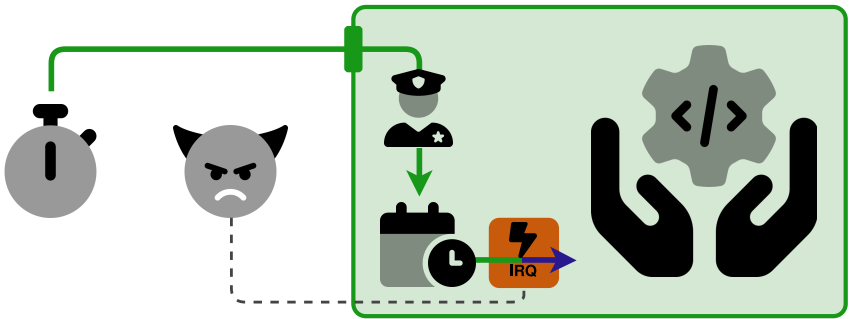
T_2: Time moves at constant pace



Getting reliable wall-clock time is hard!

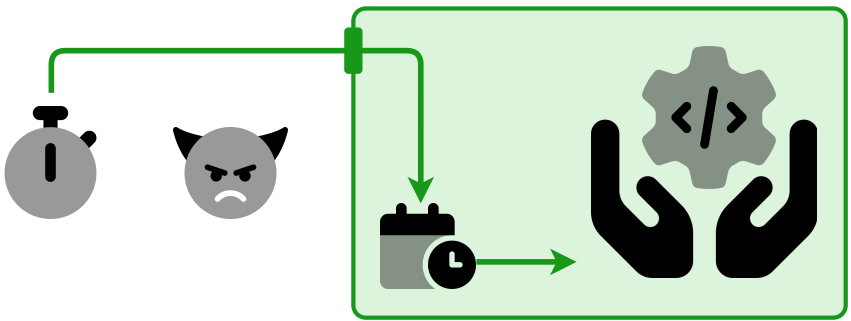
T_3: Time is read with known delay





Getting reliable wall-clock time is hard!

T_4: use of time is atomic



Overview of time levels

Type	Rollback	Freq.	Delay	Interrupt	Example time source
T_0					Untrusted OS
T_1	✓				Untrusted OS + check
T_2	✓	✓			ME, timer thread, remote server
T_3	✓	✓	✓		Secure TSC, MMIO timer
T_4	✓	✓	✓	✓	Trusted scheduler

Intel SGX — **On Windows**

4.6 Trusted Time Service Architecture

As discussed in Section 3.2, for trusted time service, the PSE uses the CSME Protected Real-Time Clock (PRTC) based timer, and provides a timer source epoch to allow application enclaves to detect timer discontinuity. A high-level view of the architecture for the trusted time service is shown in Figure 7.

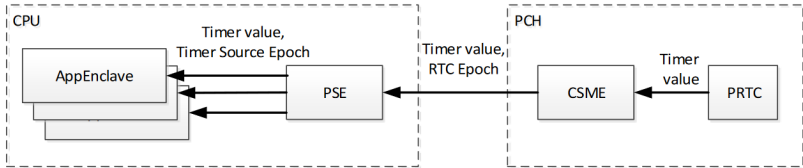


Figure 7 High-level Architecture for Trusted Time Service

- Monotonic counter but can be delayed

→ T_2

Source: Trusted Time and Monotonic Counters with Intel Software Guard Extensions Platform

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Intel SGX — **On Linux**

RDTSC and RDTSCP are legal inside an enclave for processors that support SGX2 (subject to the value of CR4.TSD). For processors which support SGX1 but not SGX2, RDTSC and RDTSCP will cause #UD.

RDTSC and RDTSCP instructions may cause a VM exit when inside an enclave.

Software developers must take into account that the RDTSC/RDTSCP results are not immune to influences by other software, e.g., the TSC can be manipulated by software outside the enclave.

- `RDTSC` can be trapped by the adversary
- Arbitrary modifications are undetectable for the enclave

→ T_1

Intel 64 and IA-32 Architectures Software Developer Manual March 2023 – Volume 3, Section 36.6.1

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Intel SGX — In the future

CHAPTER 8
ASYNCHRONOUS ENCLAVE EXIT NOTIFY AND THE EDECCSSA USER LEAF FUNCTION

8.1 INTRODUCTION

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel® SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function

- `AEX-Notify` will make an enclave aware when it was interrupted
- If never interrupted, the enclave can rely on `RDTSC`

→ T_4 (If uninterruptability is feasible for deployment)

Source: Intel Architecture Instruction Set Extensions and Future Features v47

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Trusted Platform Module (TPM) – 2 timers: *Clock* and *Time*

36 Timing Components

36.1 Introduction

The TPM has timing components for use in time-stamping of attestations and for gating policy

Time is a free-running hardware value that is not under software control. *Time* advances when the *Time* circuit is powered and is reset to zero when power to the *Time* hardware is lost.

NOTE 1 Typically, the *Time* hardware will be powered down when the rest of the TPM is powered down.

Clock is a value that is derived from *Time* and advances as *Time* advances. *Clock* may be advanced in order to bring it into alignment with real time. However, *Clock* may not be set back except by installing a new owner.

Source: Trusted Platform Module Library Part 1: Architecture. Level 00 Revision 01.59. Section 36.

Trusted Platform Module (TPM) – 2 timers: *Clock* and *Time*

The value of *Clock* may be set forward by external software (TPM2_ClockSet()) to compensate for power interruptions or clock slew, but, except for changes in ownership (TPM2_Clear()), the TPM will not allow external software to set *Clock* backward.

The value of *Clock* may be advanced by TPM2_ClockSet() using either platform or owner authorization.

NOTE The value of *Clock* may not be advanced beyond FF FF 00 00 00 00 00 00_{16}. This restriction prevents any possibility of *Clock* rolling over during its lifetime and simplifies use of *Clock* in policies.

- *Clock* advances monotonically
- Can be advanced by the attacker

→ T_1

Source: Trusted Platform Module Library Part 1: Architecture. Level 00 Revision 01.59. Section 36.

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Trusted Platform Module (TPM) – 2 timers: *Clock* and *Time*

36.2 Time

Time is a 64-bit value that contains the time in milliseconds that the circuit providing *Time* has been powered.

NOTE Depending on the frequency of the TPM oscillator and the setting of the frequency divisor (TPM2_ClockRateAdjust()), the rate at which *Time* advances may be in error by as much as 32.5%.

Time is unaffected by TPM2_ClockSet().

- *Time* advances monotonically
- *Time* cannot be influenced by the attacker ($\pm$32.5% freq.)
- Use is atomic *within* the TPM

→ T_4

Source: Trusted Platform Module Library Part 1: Architecture. Level 00 Revision 01.59. Section 36.

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Intel TDX

18.17.3 Time-Stamp Counter Adjustment

Software can modify the value of the time-stamp counter (TSC) of a logical processor by using the WRMSR instruction to write to the IA32_TIME_STAMP_COUNTER MSR (address 10H). Because such a write applies only to that logical processor, software seeking to synchronize the TSC values of multiple logical processors must perform these writes on each logical processor. It may be difficult for software to do this in a way that ensures that all logical processors will have the same value for the TSC at a given point in time.

The synchronization of TSC adjustment can be simplified by using the 64-bit IA32_TSC_ADJUST MSR (address 3BH). Like the IA32_TIME_STAMP_COUNTER MSR, the IA32_TSC_ADJUST MSR is maintained separately for each logical processor. A logical processor maintains and uses the IA32_TSC_ADJUST MSR as follows:

- On RESET, the value of the IA32_TSC_ADJUST MSR is 0.
- If an execution of WRMSR to the IA32_TIME_STAMP_COUNTER MSR adds (or subtracts) value X from the TSC, the logical processor also adds (or subtracts) value X from the IA32_TSC_ADJUST MSR.
- If an execution of WRMSR to the IA32_TSC_ADJUST MSR adds (or subtracts) value X from that MSR, the logical processor also adds (or subtracts) value X from the TSC.

Unlike the TSC, the value of the IA32_TSC_ADJUST MSR changes only in response to WRMSR (either to the MSR itself, or to the IA32_TIME_STAMP_COUNTER MSR). Its value does not otherwise change as time elapses. Software seeking to adjust the TSC can do so by using WRMSR to write the same value to the IA32_TSC_ADJUST MSR on each logical processor.

Source: Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3. Version 325462-079US March 2023.

Intel TDX

```
// We read TSC below.  Compare IA32_TSC_ADJUST to the value sampled on TDHSYSINIT
// to make sure the host VMM doesn't play any trick on us.
IF_RARE (ia32_rdmsr(IA32_TSC_ADJ_MSR_ADDR) != global_data_ptr->plt_common_config.ia32_tsc_adjust)
{
    return_val = api_error_with_operand_id(TDX_INCONSISTENT_MSR, IA32_TSC_ADJ_MSR_ADDR);
    TDX_ERROR("Inconsistent IA32_TSC_ADJUST MSR!\n");
    goto EXIT_FAILURE;
}
```

`src/td_transitions/tdh_vp_enter.c` lines 314-321.

- Clock advances monotonically with fixed frequency
- Guest has direct access, but can be interrupted

→ T_3

Source: Intel® Trust Domain Extension (Intel® TDX) Module. Version 1.0.01.01.

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

AMD SEV

15.36.18 Secure TSC

SNP-active guests may choose to enable the Secure TSC feature through SEV_FEATURES bit 9 (SecureTscEn). When enabled, Secure TSC changes the guest view of the Time Stamp Counter when read by the guest via either the TSC MSR, RDTSC, or RDTSCP instructions. The TSC value is first scaled with the GUEST_TSC_SCALE value from the VMSA and then is added to the VMSA GUEST_TSC_OFFSET value. The P0 frequency, TSC_RATIO (C001_0104h) and TSC_OFFSET (VMCB offset 50h) values are not used in the calculation.

- Secure offset and scale parameters

? Unclear whether TSC manipulations can be detected

→ T_1 ?

Source: AMD64 Architecture Programmer's Manual Volume 2: System Programming. Version 3.40.

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

ARM Trustzone

2022 IEEE Symposium on Security and Privacy (SP)

RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone

Jinwen Wang, Ao Li, Haoran Li, Chenyang Lu, Ning Zhang
Washington University in St. Louis, MO, USA

ARM Trustzone

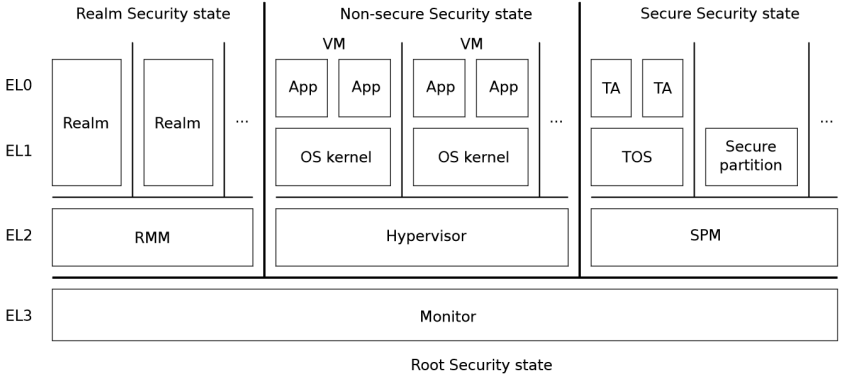
To bridge this gap, we present RT-TEE, a real-time trusted execution environment. There are three key research challenges. First, RT-TEE bootstraps the ability to ensure availability using a minimal set of hardware primitives on commodity embedded platforms. Second, to balance real-time performance and scheduler complexity, we designed a policy-based event-driven hierarchical scheduler. Third, to mitigate the risks of having device drivers in the secure environment, we designed an I/O reference monitor that leverages software sandboxing and driver debloating to provide fine-grained access control on peripherals while minimizing the trusted computing base (TCB).

- Secure world can control scheduling and I/O

→ T_4

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

ARM CCA



Source: Realm Management Monitor specification. Version 1.0-eac1.

A6.2 Realm timers

This section describes the programming model for Realm EL1 timers.

R_{LKNDV}	Architectural timers are available to a Realm and behave according to their architectural specification.
R_{YWXTJ}	During Realm execution, if a Realm EL1 timer asserts its output, a Realm exit occurs.
I_{VFYJV}	If the Host has programmed an EL1 timer to assert its output during Realm execution, that timer output is not guaranteed to assert.
R_{FKCHX}	If the Host has programmed an EL2 timer to assert its output during Realm execution, that timer output is guaranteed to assert.
R_{RJZRP}	Both the virtual and physical counter values are guaranteed to be monotonically increasing when read by a Realm, in accordance with the architectural counter behavior.
R_{JSMQP}	When read by a Realm, either the virtual or physical counter returns the same value at a given point in time on a given PE.
X_{YCDMW}	In order to ensure that the Realm has a consistent view of time, the virtual timer offset must be fixed for the lifetime of the Realm. The absolute value of the virtual timer offset is not important, so the value zero has been chosen for simplicity of both the specification and the implementation.

Source: Realm Management Monitor specification. Version 1.0-eac1.

ARM CCA

and register save / restore sequences to manage Realms. At the same time, the RMM is much simpler than a typical hypervisor because it does not do any of the following:

- Dynamic resource allocation
- Make scheduling decisions
- Manage interrupts
- Provide complex device emulation

Instead, the RMM relies on the Non-secure hypervisor (the Host) to provide this functionality, and its own activities are limited to only those required to protect the confidentiality and integrity of Realms. As a result, its implementation can be much smaller than a typical bare-metal hypervisor.

- Clock cannot be influenced by attacker
- Untrusted hypervisor controls scheduling and interrupts

→ T_3

Source: Arm Confidential Compute Architecture Software Stack Guide. Version r1p0.

T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

TEE overview [ASVBM23]

Intel SGX	TPM	Intel TDX	AMD SEV	ARM TrustZone	ARM CCA
$T_1 - T_4$	$T_1 - T_4$	T_3	T_1 (?)	T_4	T_3

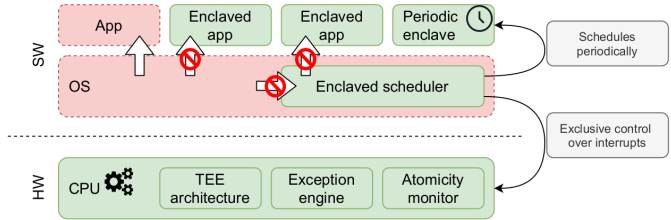
T_1 No rollback
T_2 Consistent frequency
T_3 Known delay
T_4 Interrupt prevention

Can we do T4 with multiple enclaves on a light-weight TEE?

Dependable Mixed-Criticality with TEEs [AVBPM21]

We want security (Sancus):

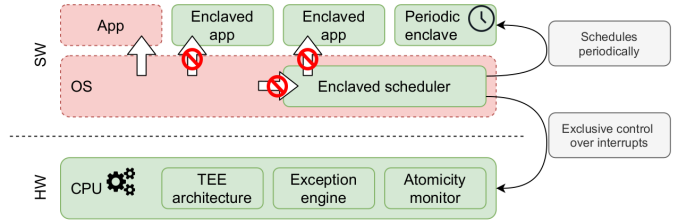
- Spacial isolation, memory curtaining, enclaves
- Enclave attestation
- Dynamic deployment



Dependable Mixed-Criticality with TEEs [AVBPM21]

We want security (Sancus):

- Spacial isolation, memory curtaining, enclaves
- Enclave attestation
- Dynamic deployment



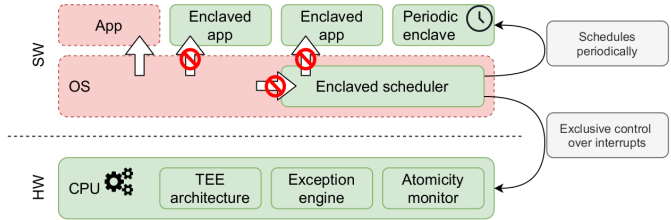
Availability Extensions (Aion):

- Exception Engine facilitates interruption of (protected) threads
- Atomicity Monitor provides control over interrupts to scheduler, guarantees bounded critical sections

Dependable Mixed-Criticality with TEEs [AVBPM21]

We want security (Sancus):

- Spacial isolation, memory curtaining, enclaves
- Enclave attestation
- Dynamic deployment



Availability Extensions (Aion):

- Exception Engine facilitates interruption of (protected) threads
- Atomicity Monitor provides control over interrupts to scheduler, guarantees bounded critical sections

Trusted Software

- Protected Scheduler controls interrupts and scheduling decisions

Aion Guarantees & Limitations

- Aion implementation provides deterministic scheduling and trusted time, even in the presence of strong software adversaries
- Aion only guarantees an Interrupt Arrival Time of 6920 cycles
- After this, the handling job starts to execute with its own atomically bounded periods
  $\rightarrow$ Guaranteeing progress is not trivial!
- Right now: Progress = critical section bound (1000 cycles)

Authentic Execution of Distributed Event-Driven Applications

“Authentic Execution of Distributed Event-Driven Applications with a Small TCB”, Noorman et al., STM 2017 [NMP17], CCS 2021 [SPN+21], TOPS 2023 [SPN+23]

Summary

About Time

1. Time does not exist in enclaves
2. Not all tasks need the best time
3. Different TEEs provide different notions of enclave time

Embedded TEEs: Sancus & Aion

1. Light-weight, hardware-only, open-source TEE
2. Built upon openMSP430 16-bit MCU, applications in IoT and embedded control systems
3. Now with real-time and availability support

Exciting Use Cases

1. Strong security and availability for distributed control systems
2. Mixed-criticality with safety functions on same platform
3. Try it out: https://distrinet.cs.kuleuven.be/software/sancus/




Open Questions

Dependable Systems Stuff

1. Secure intermittent computing
2. Schedulability with non-trivial resource sharing
3. Demonstrators for critical domains: energy transition, etc.

Formal Stuff

1. Schedulability with non-trivial resource sharing
2. Formalise and verify isolation properties
3. Language, tooling, verification support




4th edition of SICT 2023: July 3rd – July 7th 2023 in Grenoble, France.

`https://www.sictdoctoralschool.com/`

9th edition of Computing within Limits: June 14th – June 15th 2023.

`https://computingwithinlimits.org/2023/`

Thank you!

Thank you! Questions?

https://distrinet.cs.kuleuven.be/
https://distrinet.cs.kuleuven.be/software/sancus/
https://github.com/sancus-pma/tutorial-dsn18
https://github.com/AuthenticExecution/examples

References I

F. Alder, G. Scopelliti, J. Van Bulck, and J. T. Mühlberg.
About time: On the challenges of temporal guarantees in untrusted environments.
In *Workshop on System Software for Trusted Execution (SysTEX 2023)*, 2023.

F. Alder, J. Van Bulck, F. Piessens, and J. T. Mühlberg.
Aion: Enabling Open Systems through Strong Availability Guarantees for Enclaves.
In *CCS '21*, New York, NY, USA, 2021. ACM.

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):1–1, 2017.

J. Noorman, J. T. Mühlberg, and F. Piessens.
Authentic execution of distributed event-driven applications with a small TCB.
In *STM '17*, vol. 10547 of *LNCS*, pp. 55–71, Heidelberg, 2017. Springer.

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), 20:7:1–7:33, 2017.

G. Scopelliti, S. Pouyanrad, J. Noorman, F. Alder, F. Piessens, and J. T. Mühlberg.
POSTER: An open-source framework for developing heterogeneous distributed enclave applications.
In *CCS '21*, New York, NY, USA, 2021. ACM.

G. Scopelliti, S. Pouyanrad, J. Noorman, F. Alder, C. Baumann, F. Piessens, and J. T. Mühlberg.
End-to-End Security for Distributed Event-Driven Enclave Applications on Heterogeneous TEEs.
ACM Transactions on Privacy and Security, p. 3592607, April 2023.

References II

J. Van Bulck, J. T. Mühlberg, and F. Piessens.
VulCAN: Efficient component authentication and software isolation for automotive control networks.
In *ACSAC '17*, pp. 225–237, New York, NY, USA, 2017. ACM.