PROJECT OUTLINE

This PhD project will use Trusted Execution Environments (TEEs) to implement a notion of availability and real-time guarantees in embedded and high-end systems. Our research hypothesis is that TEEs can enable the development of real-time systems with a small Trusted Computing Base (TCB). Current real-time systems are based on specialized design patterns and programming languages and often require expert knowledge. However if the core components of a real-time system would be provided by a small TCB inside of a TEE, this would massively reduce development costs due to a reduced complexity of the system and lower deployment costs due to the possibility of using commercial off-the-shelf (COTS) hardware. Instead of requiring specialized hardware or programming paradigms, it would be possible for non-experts to deploy low-cost mixed-criticality systems that provide availability guarantees and can be used in scenarios that need real-time compliance.

We first establish a common ground by giving background on the state of the art of hardware isolation and TEEs. Next, we explain the objectives of this PhD project and discuss the methodology and the associated work plan to achieve the mentioned objectives.

1 Rationale and positioning with regard to the state-of-the-art

TEEs provide an environment in which small pieces of code, called enclaves, are isolated from the Rich Execution Environment (REE) which consists of the Operating System (OS) and its executed software [10]. If an attacker compromises the REE, e.g. through vulnerabilities in the OS that is running within the REE, this isolation protects the confidentiality and integrity of enclaves that are executed by the TEE. While the attacker might be able to influence and tamper with all data that is held within the REE, she has no access to data that is held by enclaves within the TEE. Furthermore, TEEs allow external parties to authenticate the system's hardware and software configuration through a process called remote attestation. Remote attestation is based on cryptographic primitives that are used to establish a secure channel directly into the enclave within the isolation of the TEE and allows remote parties to attest the integrity of the enclave that they communicate with. This enables users of a TEE to perform integrity and confidentiality protected computations that can be communicated to external parties and verified as provably originating from the correct enclave [10]. Lastly, TEEs protect the integrity of code that is loaded into the isolated environment and give enclaves access to a trusted storage where they can store persistent data.

TEEs provide strong security properties, i.e. attestation, isolation, code integrity, and trusted storage which can be beneficial in a wide range of applications such as cloud computing, real-time systems, autonomous transportation, cyber physical systems, 5G services, or robotics. However, any real-time dependent application and also many other usages require varying degrees of guaranteed availability on their used resources. With most existing TEEs, availability has so far been considered as out-of-scope. Most of these TEEs put the TEE under the control of the REE to initiate, load, and schedule the execution of enclaves inside the TEE. While this ensures full compatibility with existing operating system designs and allows for simple error handling, it also places the REE in full control of the availability of the TEE. A malicious OS inside the REE could as such delay, disrupt, or completely deny the service of an enclave inside the TEE. This is unacceptable for real-time applications and prevents the use of TEEs in any area that has real-time constraints.

There exist several real world implementations of TEEs, some of which are deployed in commercial off-the-shelf (COTS) hardware. The two most deployed TEEs systems are Intel SGX and ARM TrustZone which are either available on end-user processors in desktop PCs (Intel SGX [12]) or on mobile devices (ARM TrustZone [2]). There also exist a wide range of research based TEEs such as Sancus [18], and the Risc-V-based Sanctum [7] and its successor Keystone [1]. Other academic research projects include TyTan [3] and TrustLite [14]. All of the above TEEs view the challenge of availability as out of scope

[1] https://keystone-enclave.org/

of their attacker model. As such, any attacker that compromised the REE can completely deny any service of the TEE which is not desirable in any real-time sensitive application.

2 Scientific research objectives

The current state-of-the-art systems lack support for availability guarantees of executed enclaves within a TEE. Our research hypothesis is that TEEs can enable the development of real-time systems with a small TCB. This PhD project aims to develop tools and formalize requirements for a TEE to support this kind of real-time compliance in hardware without requiring a large TCB. As a long-term objective, this means that a TEE could be used as a hardware-based scheduler in real-time and mixed-criticality systems that enforces availability of possibly mission critical elements. One intuitive example where this can be a crucial benefit is autonomous driving where safety mechanisms need to react to sensor input in an instant. If the safety mechanism has hardware-guaranteed access to the necessary sensors periodically, no other system can delay a possibly life-saving reaction of the safety mechanism by blocking this access. Similar intuitive examples exist for any real-time system where multiple processes need to access resources either periodically or non-periodically but where a blocking of a resource is undesired. There are three main advantages if a TEE-based solution can be used for real-time systems. First, a TEE-based solution with a small TCB would rapidly decrease the engineering effort required for a real-time system as the system could be abstracted away for the developer and had a lower complexity in general due to its low overhead. Second, a TEE-based solution built on common COTS hardware would allow systems to be deployed with minimal costs compared to custom-built or industry-specific hardware. The third main advantage is that a TEE-based solution can still maintain these guarantees in the presence of adversaries on the same system where even if an adversary has control over the REE of the device, she can not break its real-time guarantees. Another benefit would be that enclaves could gain exclusive access to resources which could be used for secure I/O into enclaves. Currently, no existing TEE natively supports I/O operations due to a range of issues with the trusted path between input device and enclave.

In the following, we denote an entity such as an enclave instance with the symbol $\mathcal{E}$, the trusted scheduler with $\mathcal{S}$, any specific resource $\mathcal{E}$ tries to access as $\mathcal{R}$, and the time that it needs access to this resource as t_{dur}. If $\mathcal{E}$ requires a periodic access to $\mathcal{R}$, e.g. at most every 30 seconds, this limit is denoted as t_{lim}. We identify the following core objectives:

Resource generality. While it is simple to view the resource $\mathcal{R}$ as solely the cpu time, it is highly beneficial to design access to and scheduling of a resource in a general fashion. Our first research question is how to maintain a universal definition of a resource so that it can be scheduled and accessed type-agnostic. With a generic resource, the scheduler can not only give access to cpu time but also to memory, disk space, possible sensors, I/O operations and any resource that might be available to the underlying device. Not only would a universal handling of a system resource be beneficial due to the low complexity of extending the system, it would also decrease developer effort by well-defined APIs to access any resource.

Guaranteed resource scheduling. The second research question is related to the scheduling of resources and investigates how TEEs can be utilized as resource schedulers for shared or exclusive access to resources. Such a scheduler $\mathcal{S}$ should allow any $\mathcal{E}$ to request that $\mathcal{S}$ gives it access to $\mathcal{R}$ for t_{dur} seconds in increments of exactly or at most t_{lim} seconds. This allows for time based real-time systems to periodically execute services but also enables non real-time applications to maintain crucial tasks that can not be shut down by an attacker that has control of the REE. One simple example is an auditing mechanism that periodically checks the proper use of all resources and sends an alert if it detects malicious or erroneous behavior. Without a guaranteed resource scheduling, a malicious REE could either deny the auditing mechanism's access to the I/O resource that it needs to report its alerts or even deny the whole scheduling and as such execution of the whole auditing mechanism. With a guaranteed resource scheduling however, the auditing mechanism, in this case $\mathcal{E}$, could periodically

gain access to its necessary resources such as cpu time, memory, network, and maybe sensors that it is monitoring while this access is enforced on hardware level by the scheduler $\mathcal{S}$. Similar approaches have been made with respect to Protected Module Architectures (PMAs) [25].

Multi-process resource usage. The next research question is how to allow multiple processes to request access to the same resource which can either mean sharing a resource between multiple enclaves or even sharing between an enclave and the REE. This is an important step to make the system usable in real-world and real-time systems that might have multiple stakeholders or simply multiple services requiring access to the same resources. Sharing a resource holds several challenges that must be overcome in order to guarantee availability in this scenario. One challenge is that a malicious REE or enclave might gain access to $\mathcal{R}$ but never release it. This would prevent the correct and periodic scheduling of the next stakeholder and break the availability guarantee. To overcome this issue, $\mathcal{S}$ must be able to retract access to $\mathcal{R}$ once t_{dur} has been exceeded, even if $\mathcal{E}$ is not cooperating. Another challenge is that a malicious entity $\mathcal{E}_2$ might try to access $\mathcal{R}$ while it is currently scheduled for $\mathcal{E}_1$. Depending on the type of $\mathcal{R}$, shared access to $\mathcal{R}$ has different issues ranging from breaking the availability guarantees to a breach of confidentiality if the resource is an I/O channel. As a result, to enable multi-process resource usage, the trusted scheduler $\mathcal{S}$ must be able to grant and revoke shared and exclusive access of entities to $\mathcal{R}$. This task is influenced by static rules based on the periodic accesses of at most t_{lim} seconds for t_{dur} seconds and additionally dynamic queries that only request access to a resource once but can not be accounted for ahead of time. A trusted scheduler will need to account for both in order to make the system secure in multi-stakeholder settings. t-kernel [11] approached this topic by periodically transferring control back to their trusted OS every few instructions. The TyTan security architecture [3] allows the REE to schedule dynamically loaded enclaves between normal tasks but they rely on an omnipotent software layer in the REE for loading and scheduling which we try to avoid and place in our TEE.

Real-time Compliance. In addition to a guarantee that $\mathcal{E}$ will receive access to $\mathcal{R}$ at most every t_{lim} seconds, we will also investigate the research question of how to provide full support for interruptible isolated execution and event-driven resource requests. This would allow the TEE to be useful in real-time systems. Interruptible isolated execution is crucial to enforce that an important service can take over execution from a currently running process if this is necessary. Especially for mixed-criticality systems where some processes have a higher priority than others, this is an important objective [4]. This paves the way for event driven resource requests where $\mathcal{E}$ can register to event e in order to be executed within t_{lim} seconds after this event triggers. While most TEE systems allow interrupts in their system model, none support the guaranteed scheduling based on events. Interrupts are supported in Intel SGX [12], in Sancus since Sancus 2.0 [18], in TyTan [3], and in TrustLite [14]. To confirm to real-time compliance, entities in a mixed-criticality system might also require dynamic prioritized access to resources, for example to perform an action as a result of an event. In the above mentioned example of a safety mechanism for autonomous driving, this mechanism would dynamically request priority access to core car components to brake the vehicle and park on the side of the street in the event of a failure or error. While the autonomous driving services have regular control of these components, the safety mechanism would need to interrupt running resource accesses, be scheduled a prioritized slot that overwrites all periodic access requests, and should not be interrupted itself. Masti et al. [17] propose a mix of hard- and software to prevent misbehaving applications from holding resources instead of releasing them after their allocated time. The state of the art on real-time compliance are real-time operating systems (RTOS) or separation kernels which is a widely researched topic [5, 15, 13, 16].

Remote attestation. The last research question is how to allow the attestation of the device's scheduling operations by remote parties. One of the main benefits of a TEE are its remote attestation capabilities that can be used by external parties to verify the integrity of the attested system. If $\mathcal{S}$ can

be attested by a third party and show its properties as executing correctly, this can establish trust into the attested system and allow for simple policy checking of devices. It might also be crucial to not only integrity check the scheduler $\mathcal{S}$ and its current policies but also receive an audit of previous resource accesses by entities on the attested system. This is an extension on the simple requirement of a scheduler but could be realized by the scheduler itself as part of its normal operation. There is a wide range of previous work on remote attestation in embedded devices, both for solely hard- and software designs and also several combined approaches. SMART [9] is a hard- and software co-design that establishes a dynamic root of trust on low-end micro-controller units. Trusted Platform Modules [23] are secure co-processors and as such pure hardware solutions and provide remote attestation based on platform configuration registers that define the current state of the system. Multiple projects realized software-only remote attestation such as SWATT [20], SAKE [19], or work by Shaneck et al. [21]. We will be able to build on this earlier work to also provide remote attestation in our system.

3 Methodology

To approach our research questions we will follow hypothesis driven research, initiated by investigating changes to the existing Sancus platform which is actively being developed at our research group. Starting with Sancus is a low-risk option due to the high level of expertise at our research group and its open-source nature that is open to adjustments and changes to achieve our goals. We will use the Sancus platform to establish a minimal set of (possibly hardware) requirements to achieve these objectives. This minimum can then be used as a baseline for formalizations of TEE schedulers, explorations of our objectives on other platforms, and implementation of applications.

By working from an existing open-source platform towards formal requirements and COTS platforms that might not be open-source, we guarantee an incremental approach to a holistic view of TEE enforced availability. In addition to moving from Sancus to other platforms, we will also move from investigating and formalizing a subset of the functionality to a complete solution of all above discussed objectives. This can be supplemented by periodically designing and implementing real-world challenges to verify and check our assumptions and objectives. In the following, we describe the three work packages as depicted in Figure 1 with three tasks for each package. We discuss the individual risk of failure for each of these options and describe their placement in the research project. Due to the high level of flexibility of combining several work packages with each other, we mitigate the failure of one or several tasks and ensure the success of the overall project.

Each work package will follow an iterative approach to investigate the research questions stated earlier. It is sensible to start with a subset of the final objectives and build from these core capabilities. In this way, we will evaluate the research questions one by one and answering each individual question will yield novel publishable results. We will approach the objectives in three steps of incrementing complexity: Solely investigating a scheduler that enables guaranteed resource scheduling to a single entity, extending this scheduler by allowing multiple stakeholders and processes to access resources at the same time, and finally approaching real-time compliance by supporting interrupts, events, and prioritization during scheduling.

Guaranteed (shared or exclusive) resource scheduling: The first subset of research questions ask how to allow an entity $\mathcal{E}$ to request shared or exclusive access to resource $\mathcal{R}$. In this first iteration, the isolation of this access is the most critical aspect together with the fact that the requesting $\mathcal{R}$ from the scheduler $\mathcal{S}$ will guarantee that this access will happen at a specific time. Without accounting for multiple processes or other trusted parties on the same system that might also need access to the same resource, investigating this first research question subset is a viable approach to the overall challenge and can clearly be considered as the first step.

Multi-stakeholder resource usage: As an extension of the first research question, we will then expand the scope of the system to account for multiple stakeholders that might require access to

the same resource. This not only includes other enclaves within the TEE but also the REE itself. An important aspect here is that while several entities might have access to $\mathcal{R}$ scheduled, access to $\mathcal{R}$ should always be isolated to the one specific $\mathcal{E}_1$ that currently has access to it while no other $\mathcal{E}_2$ should be able to read out the current interaction with this resource. This objective adds interesting multi-tenant scenarios to the list of possible deployment scenarios and requires strong isolation but also a possibility for $\mathcal{S}$ to revoke the resource again if it is not surrendered voluntarily after the expiration of t_{dur}.

Multi-stakeholder real-time resources: The last step in approaching our research questions adds the last objective of real-time compliance and remote attestation to our scope. With this last step, $\mathcal{S}$ will be able to accept event-based registrations to resources and be able to remotely attest not only the policies but also the accesses each entity has to the given resources. Due to the complexity and interconnection of registering events, interrupts for real-time compliance, and multiple stakeholders, this will be the last step in the development process of availability guaranteed resource scheduling.

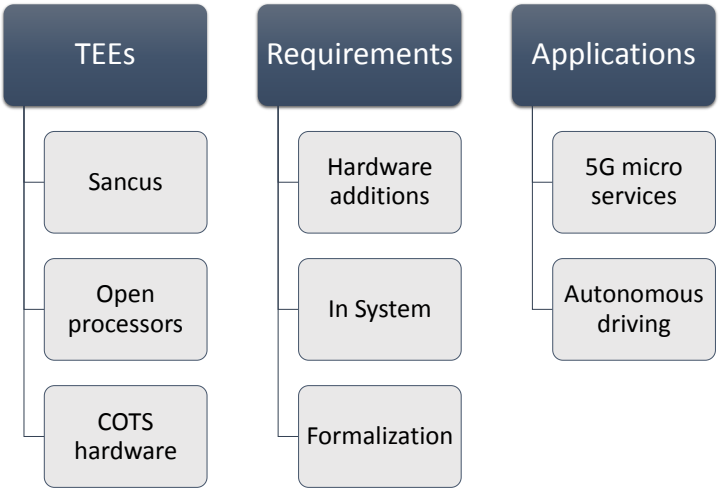

Figure 1: Overview of all three work packages and their tasks.

This order of incrementally widening the scope of our research questions will be applied to all of the following work packages.

3.1 Work package 1: Availability for specific TEEs

The first work package bundles the investigation of the different TEEs that can be used to answer our research questions. We identify three suitable environment groups, each with different levels of risk: Sancus, open processors like the Risc-V based Keystone enclave, and COTS hardware such as Intel SGX and ARM TrustZone.

Sancus (low-risk). Sancus [18] is a security architecture that aims to provide attestable isolated execution without requiring trust in software for low-cost, resource-constrained embedded devices. Our first approach will be to work on the mentioned objectives on the Sancus architecture as it is fully open-source and actively being developed at our research group. We view working with the Sancus architecture as low-risk as I will be able to benefit from the collaboration with the experts at the research group. Another benefit of Sancus is that it is built on minimal assumptions on the trusted computing base and in fact does not require trust in any software on the deployed system to operate. This means that we can investigate our research questions and their feasibility in a low-risk environment before moving on to more challenging environments.

Open processors - Keystone enclave (medium-risk). The second possible environment that we identify is the area of open-hardware processors, namely the RISC-V processor, and the Keystone

project. Keystone is a research project that aims to develop an open-source and open-hardware based TEE by establishing a TCB in one of the privileged management modes that is available in the RISC-V architecture. So far, Keystone supports physical memory protection, memory isolation, interrupts, and exceptions [2]. The benefit of evaluating and implementing our objectives in an open-hardware environment is that there are no proprietary, possibly closed-source components that belong to the TCB that might hinder the realization of our scheduler. At the same time, the RISC-V architecture is an internationally respected and actively researched project and as such extending the Keystone enclaves with availability guarantees would be timely and a publishable improvement of the current state of the art.

COTS hardware - Intel SGX, ARM TrustZone (medium to high-risk). The third possible environment for this PhD project will be working directly with COTS hardware such as Intel SGX or ARM TrustZone. It is also possible to not work directly with TEEs already embedded into COTS hardware but utilize virtualization extensions such as Fides [22] to answer the research questions. While extending COTS hardware would have the largest impact and value for real-life applicability of our work, it also has the greatest risk attached to it due to their usually proprietary nature. Most often, developers on these platforms do not have full access to the whole TCB but can instead only use predefined APIs and libraries whose source code might never be revealed by the manufacturer. Additionally, there can be elements on these systems that are locked down for developers, such as the System Management Mode on Intel's SGX hardware. We rate approaching COTS hardware as medium risk and will need to mitigate possible dead ends in the implementation and evaluation of our objectives on these platforms by researching multiple trajectories.

3.2 Work package 2: Requirements and formal guarantees for availability

The second work package deals with the requirements and the formal guarantees that are necessary for TEE-based availability. This includes investigating hardware additions to the existing environment, only utilizing the system's capabilities to achieve our goals, and finally formalizing the requirements. We now give an overview of each of these tasks.

Hardware additions (low-risk). When investigating changes to an existing system to realize novel functionality, the least error-prone option is to propose hardware additions to the system in order to achieve specific goals. In this sense, we will first evaluate the minimal set of hardware additions to the Sancus and possibly other environments that are required to achieve the objectives stated in the previous section. While hardware additions have the downside of being impractical, they can give interesting insights into the minimum requirements that are necessary to answer all research questions. As such, the hardware additions will be the first learning step for us and allows us to build on this knowledge with successive work tasks.

In system (medium-risk). The second task of this work packages investigates our research questions within the constraints of a given system without changing or proposing any hardware. By extending existing systems in software, we can provide highly valuable work as it is reproducible and directly applicable by other researchers. At the same time, this approach also holds a larger risk than simply suggesting hardware additions as we will be restricted by the existing capabilities of the underlying system. Especially if the environment contains source code that is not open-source, such as the proprietary COTS environments, this approach holds a medium-level risk as it simply might not be possible to realize our given objectives within this environment. We mitigate this risk by diversifying the TEEs that we investigate which allows us to still evaluate the given challenges broadly, even without assessing one specific environment.

Formalization (high-risk). The last task of this work package is the formalization of the given requirements. Formalization will give a fundamental approach to our objectives and will give a definition of

[2] https://keystone-enclave.org/

the smallest subset of features required, independently of the environment that these features are implemented on. This work package will require formal models to properly define the system and its requirements and will utilize existing standards and best-practices [8, 6]. We identify a high-risk for this methodology as I personally do not have any experience with formalizing a whole system or like in this case with formalizing requirements for system components yet.

3.3 Work package 3: Applications

The last work package that we will explore is to design and implement applications that make use of a trusted scheduler with a specific TEE environment. Such an application can either complement another work package or simply give context to an otherwise academic approach to the underlying challenges. We identify two possible applications that can be explored and will yield novel insights with potential economic impact for their respective application domain: Migrating 5G micro services and safety mechanisms in autonomous driving.

A1: Migrating 5G micro services. This application will evaluate the research question whether it is possible to give real-time guarantees for migrating enclaves, which is required by micro services that run in the 5G infrastructure. 5G network cells will be smaller compared to cell towers of older generations and it is to be expected that end devices will move regularly between towers. Micro services that are attached to client data needs to be passed over to the next cell tower within strict time-limits to prevent interruptions of service. Especially with the high data rates and low latency of 5G, any interruption needs to be kept as short as possible and it would be beneficial if it is possible to give an exact time guarantee on the duration of this handoff. TEEs offer a security and privacy advantage in the realm of 5G and allow providers to comply to privacy regulations and isolate client services from each other. This application is closely connected to my earlier publication on migrating SGX enclaves [1] and I have experience with the challenges and requirements of this task. While migrating enclaves is discussed by this earlier work, it did not explore any real-time challenges on the migration process and especially can not give any guarantee for the completion of the migration. As such, we think that extending this previous work and applying it on the timely topic of 5G micro services will yield publishable results that are relevant to the overall field. At the same time, my experience with this topic lowers the risk involved with this application significantly.

A2: Safety mechanisms in autonomous driving. The second application will tackle the research challenge of how to enable multiple TEE-based services in an autonomous car to cooperate while allowing for safety mechanisms to take over in an instant whenever abnormal behavior is detected. This application is a natural extension on the objectives that we will develop throughout our research questions and the discussed objectives and adds some complexities due to the context of an autonomous car. While the application requires all real-time compliance objectives like interrupts, event- and time-based resource scheduling, and multi-stakeholder compatibility, there are also additional requirements such as redundancy, error-mitigation techniques, and resource constraints that make this application a relevant contribution to the state of the art. Development of this application can be performed similarly to the demo scenario of VulCAN [24] that was developed at our research group. Such a demo is not only an effective way of explaining the project to a non-expert audience, it also demonstrates the feasibility of the developed tools in a real-world scenario.

4 Work plan

Based on the earlier mentioned methodology and research vectors, we now present our work plan for the 4-year grant period. Some of the work package subtasks have a high risk of either being impossible or posing a dead end and we alleviate this impact by following multiple research trajectories. We first present our work plan and then discuss possible adjustments to this plan if they become necessary through failures of one or several tasks. Within our work plan, answering each individual question will yield novel and publishable results with potential economic impact.

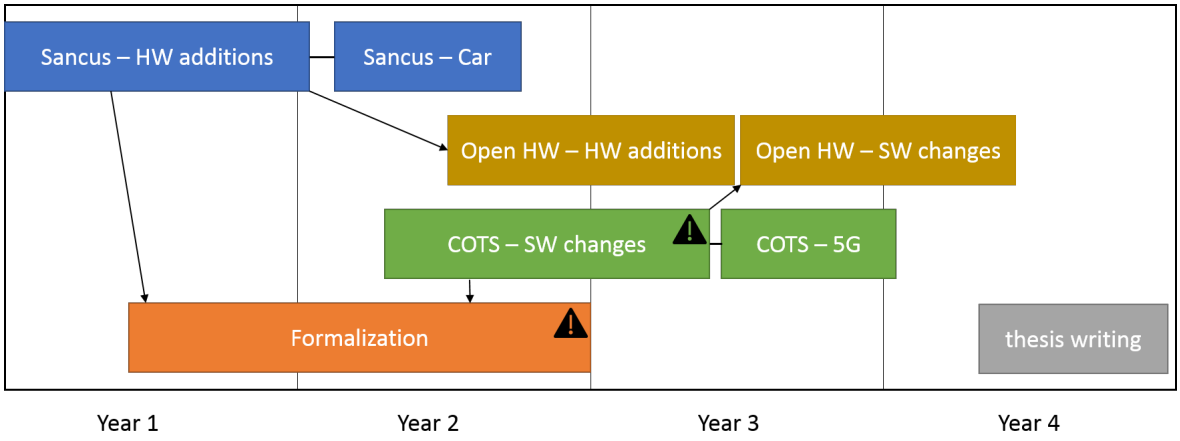

Figure 2: Our work plan with Sancus related tasks depicted in blue, open-hardware related tasks depicted in gold, COTS related tasks in green, and the formalization in orange. Arrows depict a knowledge transfer between work packages while lines depict a natural extension of a task.

4.1 Ideal work plan

Figure 2 depicts our work plan for the 4-year grant period. The blue work packages symbolize the Sancus related tasks, gold the open hardware environments, green COTS hardware, and orange the formalization that is system-agnostic. As previously mentioned, we will start by investigating hardware additions to the existing Sancus security architecture (**Sancus — HW additions**). Since this is a low-risk work package due to its nature of proposing new features, we will quickly be able to expand this and achieve a complete overview of our research questions early in the project. We expect the first publishable results from this work as it will give insights into the domain of guaranteed TEE-based availability. Once this is complete, we will have gained first experience with the requirements needed for our objectives, and maybe also gained first experience about the limits of different approaches. These insights will allow us to start the formalization work task that we expect to be developed and refined over the course of Year 2.

At the end of the Sancus related work packages, we will have gained insight into our reserach questions and will be able to use this experience for two following work tasks: Evaluating hardware changes on open-hardware processors (**Open HW — HW additions**) and developing the autonomous driving application on the Sancus architecture with the proposed hardware changes (**Sancus — Car**).

Our research group already has an existing prototype regarding attestation in autonomous cars for a previous scientific publication [24] which could be used as a base for the autonomous car application. This first application also allows us to showcase the underlying work to non-expert audiences and explain its importance for the development of future mixed-criticality systems across domains. **Open HW — HW additions** aims to investigate hardware additions to existing open-hardware processors and we will learn from the previous completion of the same work on the Sancus environment. Independently but in parallel, we will start to work on the work package **COTS — SW changes** to evaluate how COTS hardware can be used without hardware additions to answer our research questions. I personally, and our research group in general have a high level of expertise with certain COTS systems and will be able to apply this knowledge for this project. In the third year, we expect to have gained enough experience and progress to explore the 5G application with COTS hardware (**COTS — 5G**). At the same time, we expect to make progress with open-hardware environments and the development on these platforms to apply our knowledge of software additions from the COTS environment to open-processors (**Open HW — SW changes**). The latter half of the fourth year is then reserved for thesis writing.

4.2 Flexible work plan in case of failures

With our diverse set of possible research directions, we are able to mitigate failures and provide publishable results even in the case of one research trajectory not yielding promising results. We are able to mitigate failures in one work package by taking a different path than depicted in the ideal work plan. One example is a failure of the formalization of our requirements. Since we identified this work package as high-risk, we will alleviate impact of a failure in this direction by instead focusing more on software additions to open-hardware platforms such as Keystone. This is a work package that is only considered for the end of the third year in the ideal work plan but which can also be moved to earlier in the process.

If it is not possible to provide availability guarantees in COTS hardware through software changes (Failure of **COTS — SW changes**), we will alleviate this impact by two actions: Realizing application one (migrating 5G micro services) in an open-hardware environment instead of COTS hardware, and investigating hardware changes to the existing COTS hardware to answer our research questions instead of only relying on software additions. First, approaching the 5G application in open-hardware will yield scientific results of the same quality even if it will have less economic impact due to the academic nature of the underlying platform. Second, if it is not possible to achieve our objectives on COTS hardware with software changes only, we will add a work package that evaluates what minimal hardware changes are necessary to the COTS hardware to properly achieve our research objectives. We expect these alternative options to yield qualitative results that will provide publishable findings and are confident that the work packages are flexible enough to mitigate any failures of single tasks within a work package.

5 References

[1] Alder, F., Kurnikov, A., Paverd, A., and Asokan, N. Migrating sgx enclaves with persistent state. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2018), IEEE, pp. 195–206.

[2] ARM. Building a secure system using TrustZone technology. *ARM white paper* (2009).

[3] Brasser, F., El Mahjoub, B., Sadeghi, A.-R., Wachsmann, C., and Koeberl, P. Tytan: tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (2015), IEEE, pp. 1–6.

[4] Burns, A., Davis, R. I., Baruah, S., and Bate, I. Robust mixed-criticality systems. *IEEE Transactions on Computers 67*, 10 (2018), 1478–1491.

[5] Burns, A., and Wellings, A. J. *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[6] Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H. C., Sangiovanni-Vincentelli, A., and Lavagno, L. Hardware-software codesign of embedded systems. *IEEE micro 14*, 4 (1994), 26–36.

[7] Costan, V., Lebedev, I. A., and Devadas, S. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium, USENIX Security 16* (2016), pp. 857–874.

[8] Edwards, S., Lavagno, L., Lee, E. A., and Sangiovanni-Vincentelli, A. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE 85*, 3 (1997), 366–390.

[9] Eldefrawy, K., Tsudik, G., Francillon, A., and Perito, D. Smart: Secure and minimal architecture for (establishing dynamic) root of trust. In *NDSS* (2012), vol. 12, pp. 1–15.

[10] Global Platform. The trusted execution environment: Delivering enhanced security at a lower cost to the mobile market. *Global Platform white paper* (2015).

[11] Gu, L., and Stankovic, J. A. t-kernel: Providing reliable os support to wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (2006), ACM, pp. 1–14.

[12] Intel. Software guard extensions developer guide. `https://software.intel.com/en-us/documentation/sgx-developer-guide`, 2017.

[13] Jensen, E. D., Locke, C. D., and Tokuda, H. A time-driven scheduling model for real-time operating systems. In *RTSS* (1985), vol. 85, pp. 112–122.

[14] Koeberl, P., Schulz, S., Sadeghi, A.-R., and Varadharajan, V. Trustlite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems* (2014), ACM, p. 10.

[15] Krishna, C. M. Real-time systems. *Wiley Encyclopedia of Electrical and Electronics Engineering* (2001).

[16] Leiner, B., Schlager, M., Obermaisser, R., and Huber, B. A comparison of partitioning operating systems for integrated systems. In *International Conference on Computer Safety, Reliability, and Security* (2007), Springer, pp. 342–355.

[17] Masti, R. J., Marforio, C., Ranganathan, A., Francillon, A., and Capkun, S. Enabling trusted scheduling in embedded systems. In *Proceedings of the 28th Annual Computer Security Applications Conference* (2012), ACM, pp. 61–70.

[18] Noorman, J., Van Bulck, J., Mühlberg, J. T., Piessens, F., Maene, P., Preneel, B., Verbauwhede, I., Götzfried, J., Müller, T., and Freiling, F. Sancus 2.0: A low-cost security architecture for IoT devices. *ACM Transactions on Privacy and Security (TOPS) 20*, 3 (September 2017), 7:1–7:33.

[19] Seshadri, A., Luk, M., and Perrig, A. Sake: Software attestation for key establishment in sensor networks. In *International Conference on Distributed Computing in Sensor Systems* (2008), Springer, pp. 372–385.

[20] Seshadri, A., Perrig, A., Van Doorn, L., and Khosla, P. Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004* (2004), IEEE, pp. 272–282.

[21] Shaneck, M., Mahadevan, K., Kher, V., and Kim, Y. Remote software-based attestation for wireless sensors. In *European Workshop on Security in Ad-hoc and Sensor Networks* (2005), Springer, pp. 27–41.

[22] Strackx, R., and Piessens, F. Fides: Selectively hardening software application components against kernel-level or process-level malware. In *Proceedings of the 2012 ACM conference on Computer and Communications Security (CCS)* (2012), ACM, pp. 2–13.

[23] Trusted Computing Group. TPM main specification level 2 version 1.2, revision 116.

[24] Van Bulck, J., Mühlberg, J. T., and Piessens, F. VulCAN: Efficient component authentication and software isolation for automotive control networks. In *Proceedings of the 33th Annual Computer Security Applications Conference (ACSAC'17)* (2017), ACM.

[25] Van Bulck, J., Noorman, J., Mühlberg, J. T., and Piessens, F. Towards availability and real-time guarantees for protected module architectures. In *Companion Proceedings of the 15th International Conference on Modularity (MASS'16)* (2016), ACM, pp. 146–151.