

Development of an Embedded Platform for Secure CPS Services

Vincent Raes^(✉), Jan Vossaert, and Vincent Naessens

MSEC, imec-DistriNet, KU Leuven, Technology Campus Ghent,
Gebroeders Desmetstraat 1, 9000 Ghent, Belgium
{vincent.raes,jan.vossaert,vincent.naessens}@kuleuven.be

Abstract. Cyber-Physical Systems are growing more complex and the evolution of the Internet of Things is causing them to be more connected to other networks. This trend, combined with the fact that increasingly powerful embedded devices are added to these systems opens up many new opportunities for the development of richer and more complex CPS services. This, however, introduces several new challenges with respect to the data and software managed on these CPS devices and gateways. This paper proposes a platform for the development of secure cyber-physical devices and gateways. The platform provides a secure environment in which critical CPS services can be running. The secure environment relies on the ARM TrustZone security extensions. A commodity Android environment is provided in which the user can install additional software components to extend the functionality of the devices. A prototype of the platform is developed and this prototype is evaluated.

Keywords: TrustZone · Genode · Android · CPS · Security
Embedded system

1 Introduction

The Internet of Things and its integration in Cyber-Physical Systems (CPS) is causing a major evolution in our lives. Previously simple isolated systems are becoming more connected and are equipped with more powerful processors to run more complex services. These new levels of connectivity and processing power provide opportunities for the development of new products and services and, potentially also new business models. Currently IoT/CPS devices are typically integrated in a closed system and serve a specific use case. However, third-party developers could use the hardware capabilities (e.g. sensors, actuators and networking connectivity) to realize additional services. Hence, a more open software model could greatly increase the value of CPS devices by integrating them into additional ecosystems. This, however, introduces several new challenges [9] with respect to the data and software managed on these CPS devices and gateways. This is typically solved by enforcing access control to the data via a back-end system. However, in many cases it is desirable that the software running on the

device itself or the gateway can be extended and the data can be directly accessed locally. Hence, a set of core security sensitive services should be run in a secure environment, isolated from the software components provided by third-parties. These core services provide the base functionality of the device. Software developers can use these core services of the device to develop their own applications and integrate the IoT device in new ecosystems. Existing embedded platforms do not provide the required support to develop these new services. The operating systems typically running on embedded devices (e.g. Windows ME or a Linux variant) make it difficult to sufficiently ensure the required level of security. The size of these OSs code base offers a large attack surface and makes it hard to ensure that there are no bugs which can be exploited on these systems, especially with a more open software ecosystem.

This paper proposes a platform that facilitates the development of these specific types of IoT devices and gateways. Our contributions in this paper are the following. First we propose a platform to provide secure services in a CPS, based on TrustZone security technology. The platform provides built-in support for the development of secure CPS services. These services run isolated from the commodity OS that allows the installation of additional third-party applications on the platform. This platform offers isolated execution for sensitive services and a transparent execution for application that are not security-sensitive. Then, a prototype of the platform is developed which is evaluated on various points such as performance, security and ease of use.

The remainder of this paper is structured in 6 parts. Section 2 reviews other approaches to provide a Trusted Execution Environment (TEE). In Sect. 3, background regarding the used technologies TrustZone and Genode is provided. The design of the platform is discussed in Sect. 5. Section 6 handles the realization of the prototype. The evaluation can be found in Sect. 7 and our results are concluded in Sect. 8.

2 Related Work

This section provides an overview of other approaches which provide security in an untrusted environment. There have been many research endeavours that are focused on protecting applications from an untrusted environment. In this section, three different categories are distinguished, namely software-based, custom hardware and TrustZone-based solutions.

The first category uses hypervisors and software-based protection to offer an isolated execution environment. AppGuard [17] takes advantage of hardware virtualization support such as Intel VT-x and AMD-V and a Trusted Platform Module (TPM) to secure user applications from the OS. It requires no modifications to be made to the application or the untrusted OS. The dependency on a TPM and a virtualization extension make it impractical to use in embedded devices. Other approaches such as InkTag [5] aim to ensure that the OS is functioning correctly. This is done with a hypervisor and a new technique called *paraverification* in which the OS helps verifying its own behaviour. InkTag incurs

a high performance cost since it encrypts and decrypts user processes based on context switches which is not ideal in an industrial setting.

The second category of security solutions consists of custom hardware solutions. Sancus [11, 16] is a hardware platform that is specifically designed to run software of mutually distrusting parties on a single device, while each party has a strong assurance that its software runs untampered. The security offered is completely hardware-based. This solution is highly geared towards networked microprocessor-based devices. We believe there is a place for more powerful embedded devices with a general purpose OS in a CPS setting.

The final category uses TrustZone as a trusted computing technology. SPROBES [4] is focused on the correct execution of an untrusted OS. It enforces that the kernel can only run code from approved code pages. There are cases that require a more in-depth security for their applications. For example in some cases the code that is executed should be kept secret. Correct functioning of the OS does not guarantee code confidentiality. Another TrustZone based solution is the Trusted Language Runtime (TLR) [12, 13]. TLR protects the security sensitive parts of .NET applications by isolating it from the OS and other applications. It is mainly geared towards mobile applications and especially Windows Phone systems. It also protects a specific part of a single application while we wish to offer services that are available to multiple untrusted applications.

3 Preliminaries

3.1 ARM TrustZone

ARM TrustZone is a set of hardware security extensions implemented on most recent ARM processors. The security extensions allow the device to run in two different processor modes, a Normal and a Secure mode. Each mode represents a virtual processor, providing two isolated execution environments. As the name suggests, the Secure mode is typically used to run security critical software components, isolated from less trusted code running in the Normal mode. The division between Secure and Normal mode is not limited to the CPU but is propagated over the system bus to peripheral devices and memory controllers, providing system-wide security. Access to specific peripherals can be restricted to Secure-mode software. A process running in the Normal mode is unable to access the Secure-mode memory regions or use Secure-mode peripherals. It can also be used to trap specific interrupts to the Secure world. Code running in the Secure mode is capable of reaching Normal-mode resources. The collection of resources accessible to software running in Normal/Secure mode is called the Normal/Secure world. The creation of two virtual worlds on a single core is achieved with a new processor bit, the NS-bit. The world in which the processor is running is determined by this bit. The system bus and memory controllers use the NS-bit to check whether a specific resource request is permitted in the context of the active world.

Since the processor can only execute in one security mode at a time, a way to switch worlds is required to run software in both security modes. A dedicated

processor instruction, the Secure Monitor Call (smc) can be used to activate the Secure world from the Normal world. The instruction changes the processor's operating mode and starts executing the Secure Monitor. The Secure Monitor is part of the Secure world and responsible for controlling communication between both worlds. Based on arguments provided with the Secure Monitor Call, the Monitor can invoke specific software components in the Secure world. The secure world can directly change the operating mode of the processor after it has finished executing the required software components and, subsequently, pass control back to the Normal world.

3.2 Genode

Genode [2] is a framework for the development of highly secure micro Operating Systems. It has been selected as the OS to run in Secure world of the platform presented in this paper. As a microkernel, Genode avoids pitfalls of regular kernels by stepping away from a monolithic kernel structure. The various services that are usually managed by the central kernel, such as device drivers and the file system, run in separate sandboxes. Genode extends the microkernel idea by de-composing the system policy as well as the operating system code. This is achieved by imposing a organizational structure on each part of the system.

A Genode application can be allowed to contact other applications with several Genode interfaces. These are a Remote Procedure Call (RPC) interface and an asynchronous notification interface. These methods are sufficient for calling functions from another service but are not suited for sending large chunks of data. The RPC messages are bound to a small size and the asynchronous notifications have no additional payload. If large amounts of data need to be propagated, Genode allows processes to share some of their assigned memory.

4 Problem Statement

With this paper we wish to propose a solution for the increased security needs in CPS and IoT environments. Recently these systems have become more connected to other networks and are using more powerful embedded devices. Since these systems were typically isolated they are lacking in terms of security. Our proposal consists of a platform that hosts secure services in a secure environment while still allowing untrusted third party applications on the device. The untrusted applications run in a familiar environment that is easily extensible. Developers of secure services get tools to create and add secure services on the platform, and a way to contact their services from the untrusted world is available. The secure world is transparent for the untrusted applications so if an application does not utilize a service, it is unaffected by their existence.

4.1 Requirements

Secure Execution: A trusted execution environment is offered in which services can run isolated from the untrusted rich OS. This environment guarantees the integrity and privacy of the service.

Secure Boot: Secure boot is used to guarantee the authenticity of the secure world and secure services upon startup.

Secure Service Development: The platform provides a build environment for the development and deployment of secure services. Secure services typically use cryptography to support authentication or confidentiality of data. This should, hence, be included in the build environment. Most services will also communicate with some kind of remote endpoint, e.g. a server or a sensor. Thus, the build environment should offer a way to set up a secure tunnel with these remote endpoints.

Rich Normal World: There should be a simple way for new applications to be installed and used in the normal world. The normal world should be able to efficiently run user applications. Application developers should be able to easily develop applications to run in the normal world and use secure world services.

4.2 Attacker Model

The attacker model used for this paper assumes the following capabilities for the attacker. First, the attacker is capable of controlling all the software running in the normal world. This is due to the normal world OSs large code base which makes it prone to exploits. Any applications running in the secure world and the secure world implementation itself are assumed to be trustworthy.

Second, any communication channels to and from the device are assumed to be controlled by the attacker. This allows the attacker to perform Man-in-the-Middle attacks, sniff the network and modify any traffic on the channel.

Further, the attacker is assumed to be unable to break cryptographic primitives, but can perform protocol-level attacks. This follows the Dolev-Yao [1] attacker model.

Finally, TrustZone's attacker model is used for hardware attacks. This means that the attacker is capable of performing low level hardware attacks such as using debugging tools but advanced attacks such as usage of an electron microscope are out of scope.

5 Design

The platform relies on the TrustZone security technology to realise a secure world in which security sensitive services are running, and a flexible normal world system which is managed by the user. This section first presents the architecture of the platform which includes the software stack, the inter-world communication, and the boot process. Subsequently, the software development support provided by the platform is discussed.

5.1 Platform Architecture

Software Stack. The platform has two separate software stacks running on it. The normal world environment runs the Android operating system. Android provides a familiar environment for users in which new applications, developed by third party developers, can be easily installed. It also offers a rich UI which has been geared towards touchscreens. The secure world runs the Genode OS. The secure services running on the platform are separate Genode applications. Each service implements a single Genode RPC function. By calling this function, the service is executed. If large chunks of data need to be passed to a service Genode's memory sharing technique, described in Sect. 3, is used. Apart from the secure services, the secure monitor is also running as a Genode application in the secure world. The secure monitor manages a list of references to the different services running in the secure environment. Each service in the list is linked to a unique 8-byte identifier. When the monitor is called with one of these identifiers, the corresponding service will be contacted. Figure 1 shows a graphical representation of the software on the platform.

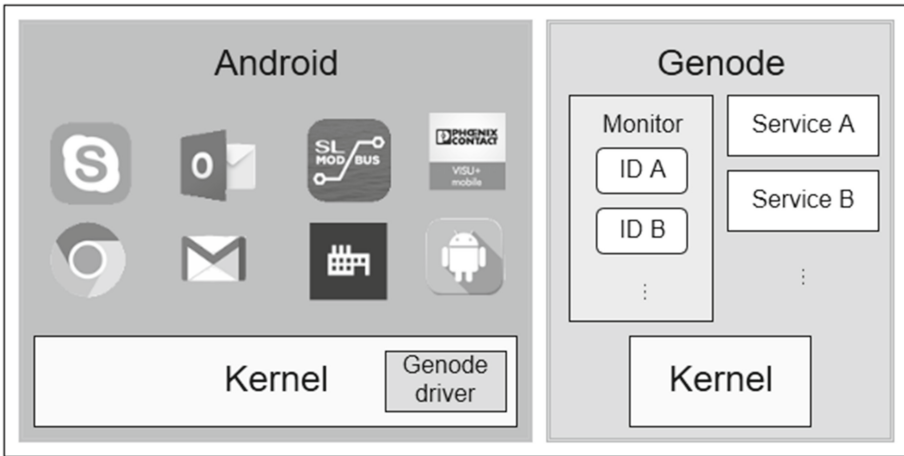


Fig. 1. The software on the platform

World Switch and Inter-world Communication. The secure world behaves as a slave to the normal world. When a normal world application requires data from a secure world service, it issues a request to the secure world. To enable these requests to the secure world, a new driver was added to the Android platform. The driver is responsible for executing the `smc` command which triggers the world switch. Furthermore, the driver manages the data structures that are passed to and from the secure world. A virtual device interface is provided to higher layers of the Android stack. This interface requires the service identifier and one data structure containing all the input for the requested service.

Once the `smc` command is executed, the TrustZone hardware passes control to the secure monitor service. When the secure monitor receives control, it reads the identifier of the requested service and calls the RPC function of that service. The data structure is also passed to the service. Once a service has finished the requested function, execution is returned to the monitor along with the generated result. The monitor then writes the response to the Android driver's memory and switches back to the normal world. Finally the driver responds to the calling function with the data written by the secure world.

Platform Boot. A TrustZone enabled board starts in TrustZone's secure mode. This ensures that the non-secure software is incapable of interfering with the execution of software when the device is booting. The boot process in our platform starts with a secure world bootloader. This bootloader loads the Genode image into memory and checks the integrity of the image using a hash value signed by a key that is saved in the board's hardware. If the authentication is successful, Genode boots as the secure OS. Once the TrustZone configuration is complete and the secure services have been started, Genode operates as bootloader for the normal world. It copies the Android image into memory and prepares the CPU registers accordingly. When the setup has been completed, the first world switch occurs and Android starts operating.

5.2 Software Development Support

This section further discusses the support provided by the platform to develop both applications that use services provided in the secure world and the secure services itself.

Android Application Development. Developers can use the regular development environment provided by Google to develop their Android applications. A Java library is provided to enable communication with secure services. The library offers a generic communication API that allows applications to specify the identifier of the secure service which should be contacted and any additional data for the service. In order to access the previously discussed Android driver, the library uses JNI. Developers of secure services can build on top of this library to provide a service-specific interface on top of the generic communication library. This service-specific library can automatically fill in the service identifier and serialize the different parameters for the service.

Secure Service Development. For the development of secure services, the Genode framework and external C/C++ libraries can be used. The Genode framework itself allows programs to be developed in C and C++. Genode provides a `libc` implementation based on FreeBSD's `libc` which is fairly complete. Genode's `libc` implementation can be extended using plugins to provide additional functionality.

External libraries can be ported to the Genode framework to provide advanced or specialised functions. Porting is a matter of providing the necessary

functionality, if necessary through patching the library, and ensuring libraries the new library depends on, such as `libc`, are available. In our platform a port of the OpenSSL 1.0.1u library is provided since many secure services require cryptographic operations. This OpenSSL library is used as the base for another library which helps setting up secure tunnels to remote endpoints. Secure services can use either of these libraries to use the functions they offer.

6 Prototype

A prototype of the platform has been developed using the i.MX6 SABRE Lite board. This section discusses the hardware-specific TrustZone modifications of the Secure and Normal world software. Subsequently, the use of the secure boot features of the i.MX6 SABRE Lite in the platform is discussed. The code used for this prototype can be found at <https://bitbucket.org/vincent-raes/>.

6.1 Secure World Software

The Genode OS has been modified to make use of TrustZone’s security capabilities. To achieve this, drivers were written for the various TrustZone peripherals. Every driver activates one part of TrustZone’s security measures. The security measures for memory, peripherals and interrupts each have a separate driver. Our platform has the following TrustZone peripherals: the Central Security Unit (CSU), the TrustZone Address Space Controller (TZASC) and the Generic Interrupt Controller (GIC) [3].

The CSU is used to determine what peripherals can be accessed by each world. It assigns various security levels to the board’s peripherals. These security levels vary from allowing secure and normal world processes to reach the peripheral to extremely strict policies in which only privileged processes in the secure world are capable of accessing the device. Since our platform aims to be as transparent towards Android as possible with few modifications to the Android kernel, only the peripherals that manage TrustZone settings are set to be exclusive for the secure world.

The TZASC is the default ARM device to split memory between the secure and normal world. It divides the available memory into various zones which can be assigned different security levels. These security levels are similar to those the CSU offers for peripherals. We require secure memory for our Genode runtime so a section of the available memory is set to secure world only. The size of this section is determined by the build configuration.

Finally the GIC is used to assign interrupts to either world. This is accomplished by using the two separate interrupt types ARM offers: the IRQ and the Fast IRQ (FIQ). It is possible to configure a type of interrupt so it is linked to one world. The GIC assigns an interrupt type to the interrupts available on the platform based on their identifier. Due to our goal of transparency for the normal world, there are no secure world only interrupts.

6.2 Building the Software

The boot image used on the platform is built in two stages. The first stage consists of building the Android system. Android has been patched so it can run side-by-side with the secure world. The patch disables access to any resources that are reserved for the secure world. This is achieved by modifying the build configuration file and the Device Tree Blob (DTB). Additionally, the patch adds our new driver to the Android build. This driver reserves 1024 KB which is used to communicate with the secure world.

The second stage is building the Genode system. Genode has been configured to only occupy limited resources on the hardware. Unlike a commodity OS such as Android, Genode requires knowledge of the system when it is being built. The applications that will run on the system are determined prior to the build process, since new applications cannot be installed or updated when the system is running.

The configuration for both build processes is performed in one global config file. This includes options such as the available memory for each world and what secure services will run on the system. A script is provided which uses the options in this file and builds the system accordingly. The script is used to launch the build scripts for Android and Genode. The Android build process results in the Android kernel and userland. The kernel image is reused during Genode's build process where it is packaged with the Genode kernel. The Genode output consists of a single boot image in which the kernel, the secure services, and the Android kernel are packaged. In order to determine what secure services need to be built and linked to the monitor, the global configuration file is used. Based on the selected services, placeholders found in the Genode code and scripts will be modified to dynamically build and link the required services.

6.3 System Boot

The boot process consists of several steps. The first step uses the boot ROM library and the High Assurance Boot (HAB) library to provide Encrypted boot [14] for the bootloader. The availability of encrypted boot and these software libraries is dependent on the used hardware platform. The software libraries are unchangeable and can be seen as trusted software components. Encrypted boot allows a device to be configured so only authenticated code which has been encrypted can run on the device.

Figure 2 shows the boot process. When Encrypted boot is enabled, the boot ROM loads the bootloader image to memory. This image contains the encrypted bootloader and digital signature data and public key certificate data. The additional data is collectively called the CSF data. The image is generated off-line using a dedicated tool. Encrypted boot also adds the key which has been used to encrypt the code to this image. The stored key has been encrypted with a hardware key unique for every CPU core. This results in an encrypted boot image which can only be executed by one board since the stored key is unique per CPU.

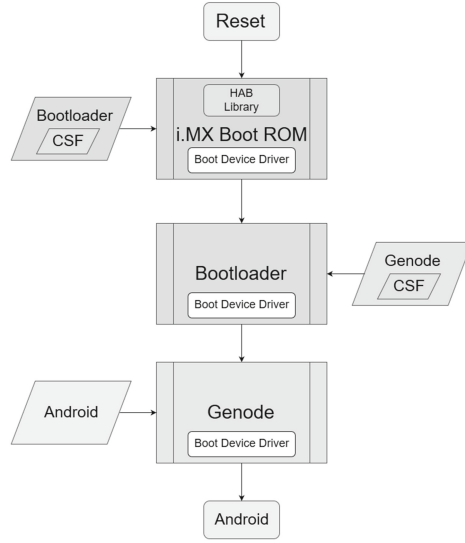


Fig. 2. The secure boot process

With the image completely loaded into memory, the HAB library takes over. This library verifies the signatures and decrypts the image. The loaded code can only be executed if the verification succeeds.

The second step starts once the bootloader has passed verification and has started. The bootloader loads the Genode image to memory. The Genode image contains the encrypted Genode kernel and the additional data used for verification. The bootloader can then issue a command which passes control to the HAB library to verify and decrypt the Genode image.

In the third step Genode acts as the bootloader for Android. Genode loads Android into memory and prepares it for booting by setting the DTB, correcting the CPU registers and setting the Instruction Pointer to the start of the Android kernel. Once this is done, control over the device is passed to the Normal World and Android boots as the final boot step.

7 Evaluation

7.1 Requirements Review

Secure Execution. In order to evaluate the security of the platform we focus on the attack vectors available to adversaries. The biggest attack vector is the Secure Monitor Call. The interface offered by the call is rather narrow. The smc interface is only used to contact secure services. A secure service has limited influence in the secure world due to Genode’s security mechanisms.

Secure Boot. Secure boot in the platform is achieved using both board specific technology and the TrustZone model. The Encrypted boot technology is used

to ensure the authenticity of the secure world bootloader, which in turn verifies the authenticity of the secure world image. This ensures the trustworthiness of the critical steps in boot process.

Secure Service Development. Writing a secure service is a relatively simple matter. Services are programs written in C++ in which at least one RPC call is made available. Based on the input a service receives with this RPC call, various functions can be executed. Genode’s configuration and code needs to be appropriately modified for every service made available but the global build script is capable of automating this process so developers can focus solely on the functionality of their service.

To provide an idea regarding the performance of the cryptographic library, several test have been executed on the prototype. Table 1 show the results of timing measurements of symmetric (i.e. AES) and asymmetric (i.e. RSA) cryptographic operations on both the Android platform and on the secure world. The measurements were made with the Performance Monitor Unit. This is a reliable counter accessibly by both the normal and secure world. The RSA implementation used in Genode is clearly very inefficient compared to the implementation Android uses. Genode is on average 40 times slower in a single operation. The results of the AES implementation on the other hand show a slight delay but the difference isn’t quite as large as with RSA. The fact that Genode is slower has multiple reasons. First, OpenSSL on Genode does not support using assembly optimization for the used platform. Further, Genode does not take advantage of the cryptographic accelerator the hardware offers.

Table 1. Time for cryptographic operations

Type of operation	Genode (ms)	Android (ms)
RSA public	53,6	1,4
RSA private	1774,1	41,1
AES encrypt	76,0	63,9
AES decrypt	75,3	49,9

Rich Normal World. Normal world applications are developed as regular Android applications. On the prototype, applications can be installed using the .apk files. On commercial platforms, an app store application can be provided. A Java library is provided to enable communication with secure World services. To test the performance impact of the secure world on the commodity OS, two tests have been run.

First, in order to test the overhead caused by activating the TrustZone security extensions, Android benchmarking tools have been used to compare the performance of the platform with and without TrustZone enabled. The benchmarking tools used are Antutu¹ and Geekbench². These tests were performed

¹ antutu.com/en/index.shtml.

² geekbench.com.

with the same Android version. The only difference on the platform was the activation of the TrustZone devices in one case. The results for these benchmarks can be found in Figs. 3 and 4. It is clear that activating TrustZone’s hardware security causes no noticeable overhead to the platform.

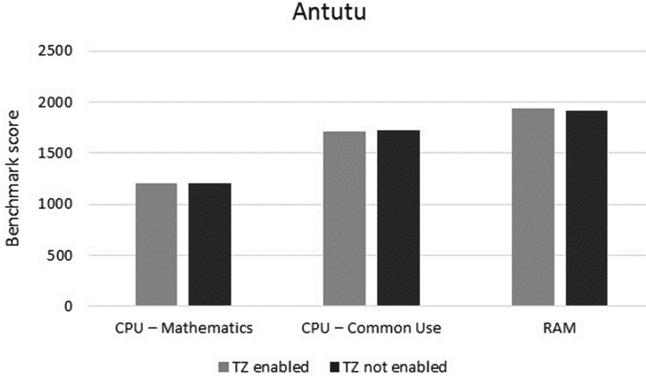


Fig. 3. Antutu benchmark results

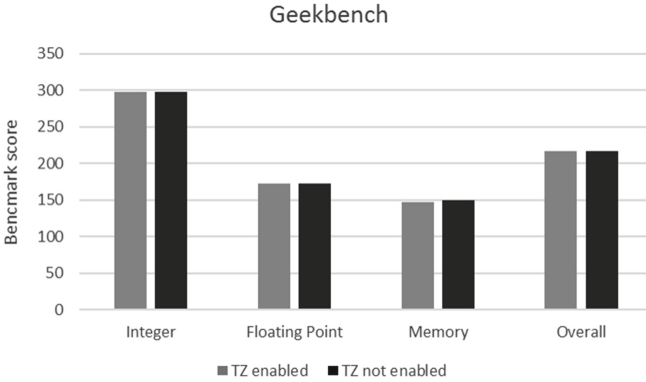


Fig. 4. Geekbench benchmark results

Second, the delay introduced by a world switch is measured using the Performance Monitor Unit. The counter is measured just before the world switch command is given and again when the activated world resumes execution. Table 2 shows the time it takes to switch worlds in either direction. The overhead introduced by the world switch is acceptable for most services. We notice a difference in the overhead depending on the direction of the world switch. This is due to how the switch is called in each world. In the normal world the smc is called directly so the timer can be started closely to the switch. In the secure world

there is more variety in the time until the switch is executed from when it is called due to how this call happens. The monitor requests the Genode kernel to make the switch but this is handled by the kernel's scheduler, so the workload of the machine has an influence on the overhead.

Table 2. Delay caused by world switch

Direction of world switch	Time in ms
From normal to secure world	0,069
From secure to normal world	0,411

7.2 Validation

This section shortly presents several use cases that illustrate the added value of the platform in the context of IoT and CPS. Future work will further explore the opportunities of the platform in these use cases.

Proprietary Local Data Processing: In the setting of Industrial Internet of Things, companies are developing services for the monitoring and management of industrial control systems. The algorithms used for this analysis are typically proprietary. Hence, to prevent competitors from obtaining these algorithms, they are currently mainly running on back-end systems. However, in some settings where analysis need to be performed on remote sites where Internet connectivity is not available or always guaranteed, the algorithms need to run in the field. Hence, a form of hardware-based confidentiality of the algorithm running in a secure environment is desirable, while allowing less trusted code on the platform to gather the data for the analysis and present the results to the user.

Leasing Machines: In the context of sustainable development and more durable customer relations, some companies are moving away from selling products (e.g. light bulbs, trains) and are selling services instead (e.g. lumination, locomotion). Companies are triggered to design products that have a longer lifespan and are easier to maintain. This can also lead to new business models in which a single device can be used by multiple customers. Hence, a set of trustworthy software components running on the device is required to handle these new business models and monitor the usage of the device. Less trusted software components could be loaded on the device to provide the interface and optionally other less critical services to the users.

Onboard Computer: In the automotive sector an increasing number of cars have an onboard computer. They have multiple internal networks and external networking capabilities. Some of the services running on these onboard computers such as park assist and engine start are critical, other services such as maps and infotainment are less critical. Further, in the transportation industry, the driving patterns need to be carefully monitored. Hence, a platform that supports

running security critical services while allowing an extensible software platform for non-critical software would provide great added value.

Home Care: In the health care sector, research and industry are intensively focusing on home care to manage the cost of the health care system. These systems allow the patient to be remotely monitored at their home. The medical gateway connects the monitoring devices to the Internet. The gateway can run critical medical services such as fall detection and heart-rate monitoring. However, an increasing number of health apps is being developed for consumers. Hence, a gateway on which these consumer apps could run next to more critical medical services can provide interesting opportunities.

7.3 Discussion

Extensions. The current setup has secure services running in a master-slave setting which limits the functionality of our platform. An interesting addition would be to have services activate periodically independent of the normal world. An extension which would allow this behaviour is to make use of the TrustZone watchdog timer. This is a timer exclusive to the secure world and which can be used to activate the secure world after a certain time regardless of the normal world's state. This extension would allow many new use cases to be serviced by our platform and is high on our priority list.

Another matter is installing new applications or updating existing software in the secure world. Currently, Genode does not allow on-the-fly installation of new services or the updating of existing software. When a software update needs to happen, the entire boot image needs to be rebuilt with the new code and settings and then redeployed. The creators of Genode, however, are striving towards supporting this functionality in later versions of the Genode OS. Once this has been realized, a secure service update and deployment scheme can be added to the platform.

Finally there is the matter of secure services accessing hardware resources. Since the secure world has access to all peripherals, the main issue herein lies with the security. Accessing peripherals through the normal world is a security vulnerability so the secure world needs drivers of its own to access these. Adding drivers increases Genode's TCB and thus increases the possibility of bugs and exploits.

Threat Model. Several TrustZone-based platforms have already been successfully attacked (e.g. Huawei HiSilicon TEE [15] and Qualcomm's Secure Execution Environment (QSEE) [8]). The attackers were able to extract security critical data from the secure environment. The attackers managed to control the Normal World OS so they could forge malicious commands for the Secure World. They then exploited vulnerabilities in the available commands to write arbitrary data to memory locations in the Secure World. Our platform provides a simple and strict communication API between both worlds. Further, an important aspect is the absence of bugs in the secure world. The secure world minimizes the trusted computing base of the secure services. For the current prototype, the secure

world software consists of approximately 24,000 lines of code. This suggests that techniques to verify the absence of specific vulnerabilities [6] such as buffer overflows could be used on the secure world code. To ensure that Genode correctly implements the isolation between the different secure services, techniques that verify that software is consistent with its specification could be used. For instance, the seL4 microkernel, consisting of 8,700 lines of C code and 600 lines of assembler, has been formally proven to be consistent with its specification and free from programmer-induced implementation errors [7]. The proof is constructed and checked in Isabelle/HOL [10], a proof assistant for higher-order logic.

8 Conclusion

This paper presents the design, implementation and evaluation of a platform which enables the development of secure rich CPS devices and gateways. This solution consists of a functional micro OS that provides a TEE for certain services. Confidentiality and integrity of the application code is offered in the isolated execution environment. These are achieved using ARM TrustZone and the Genode microkernel. A prototype of the platform is developed using the Freescale SABRELite i.MX6 development board. The evaluation of this prototype shows that it achieves a good level of security at an in general small performance cost.

References

1. Dolev, D., Yao, A.C.: On the security of public key protocols. *Trans. Inf. Theory* **29**(2), 198–208 (1983)
2. Feske, N.: Genode Operating System Framework 15.05
3. Freescale Semiconductor Inc.: i.MX6 Processor Reference Manual (2013)
4. Ge, X., Vijayakumar, H., Jaeger, T.: SPROBES: enforcing kernel code integrity on the trustzone architecture. In: *Proceedings of the Mobile Security Technologies 2014 Workshop* (2014)
5. Hofmann, O.S., Kim, S., Dunn, A.M., Lee, M.Z., Witchel, E.: InkTag: secure applications on an untrusted operating system. *ASPLOS* **2013**, 253–264 (2013)
6. Jacobs, B., Smans, J., Piessens, F.: A quick tour of the verifast program verifier. In: Ueda, K. (ed.) *APLAS 2010*. LNCS, vol. 6461, pp. 304–311. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17164-2_21
7. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an OS kernel. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP 2009*, pp. 207–220. ACM, New York (2009)
8. Laginimaine: Extracting Qualcomm’s Keymaster Keys (2016). <http://bits-please.blogspot.be/2016/06/extracting-qualcomms-keymaster-keys.html>
9. Mayer, C.P.: Security and privacy challenges in the internet of things. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol. ECEASST* **17**, 1–12 (2009)
10. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: Proof Assistant for Higher-order Logic, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>

11. Noorman, J., Preneel, B., Agten, P., Daniels, W., Strackx, R., Huygens, C., Piessens, F., Van Herreweghe, A., Verbauwhede, I.: Sancus: low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In: 22nd USENIX Security (2013). K U Leuven
12. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications (i)
13. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Trusted language runtime (TLR): enabling trusted applications on smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile), pp. 21–26 (2011)
14. Freescale Semiconductor: Secure Boot on i.MX50, i.MX53, and i.MX 6 Series using HABv4, pp. 1–22 (2012)
15. Shen, D.: Exploiting Trustzone on Android. Black Hat (2015)
16. Strackx, R., Noorman, J., Verbauwhede, I., Preneel, B., Piessens, F.: Protected software module architectures. In: Reimer, H., Pohlmann, N., Schneider, W. (eds.) ISSE 2013 Securing Electronic Business Processes, pp. 241–251. Springer, Wiesbaden (2013). https://doi.org/10.1007/978-3-658-03371-2_21
17. Zha, Z., Li, M., Zang, W., Yu, M. and Chen, S.: AppGuard: a hardware virtualization based approach on protecting user applications from untrusted commodity operating system. In: 2015 International Conference on Computing, Networking and Communications, ICNC 2015, pp. 685–689 (2015)

Computer Security

ESORICS 2017 International Workshops, CyberICPS

2017 and SECPRE 2017, Oslo, Norway, September

14-15, 2017, Revised Selected Papers

Katsikas, S.K.; Cuppens, F.; Cuppens-Boulahia, N.;

Lambrinoudakis, C.; Kalloniatis, C.; Mylopoulos, J.; Antón,

A.; Gritzalis, S. (Eds.)

2018, XII, 281 p. 76 illus., Softcover

ISBN: 978-3-319-72816-2