Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms

Johannes Winter
Institute for Applied Information Processing and Communications (IAIK)
Graz, University of Technology
Inffeldgasse 16a, 8010 Graz, Austria
Johannes.Winter@iaik.tugraz.at

ABSTRACT

Security is an emerging topic in the field of mobile and embedded platforms. The Trusted Computing Group (TCG) has outlined one possible approach to mobile platform security by recently extending their set of Trusted Computing specifications with Mobile Trusted Modules (MTMs). The MTM specification [13] published by the TCG is a platform independent approach to Trusted Computing explicitly allowing for a wide range of potential implementations. ARM follows a different approach to mobile platform security, by extending platforms with hardware supported ARM TrustZone security [3] mechanisms. This paper outlines an approach to merge TCG-style Trusted Computing concepts with ARM TrustZone technology in order to build an open Linux-based embedded trusted computing platform.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: Protection mechanisms; C.3 [**Computer Systems Organization**]: Special purpose and application based systems

General Terms

Design

Keywords

ARM TrustZone, Linux, Mobile Trusted Computing, Virtualisation

1. INTRODUCTION

This paper outlines parts of an ongoing effort of the Trusted Computing Labs at IAIK to develop building blocks for secure embedded platforms. The key focus of this paper is directed towards an open Linux-based virtualisation framework prototype for ARM TrustZone enabled platforms.

Based on the foundations provided by this virtualisation framework, a design of a mobile Trusted Computing platform supporting a mixture of hardware and software based Mobile Trusted Modules is presented. Within the scope of this paper a software-only approach to Mobile Trusted Modules will be briefly discussed.

The remainder of this paper is structured into five major parts: The current section 1 gives a brief overview of Mobile Trusted Computing and ARM TrustZone. At the end of section 1 references to related work are given. Section 2 introduces a prototype design for a trusted embedded platform and discusses implications and requirements stemming from the design decisions. Section 3 addresses the problem of providing sufficient isolation properties for the prototype platform, by using ARM TrustZone features for virtualisation purposes. The last two sections conclude the paper.

1.1 Mobile Trusted Computing

The TCG specifications for the Trusted Platform Module (TPM) [16], [15] and the accompanying Trusted Software Stack (TSS) [14] are primarily focused on PC-style platforms. A TCG compatible PC-style platform can be assumed to contain a PC-style BIOS together with a single hardware TPM.

When attempting to implement TCG-compatible Trusted Computing systems on mobile and embedded devices a number of issues not addressed by the PC-oriented TPM specifications arise. Typical embedded and mobile platforms greatly differ to PC-platforms with respect to their booting process and to their interpretation of BIOS. When considering mobile phone platforms, it might no longer be sufficient to have a single TPM on the platform due to ownership issues. The TCG has published a set of specifications trying to address the different requirements of mobile and embedded devices in [13]. This specification defines two mobile versions to the TPM, which primarily differ in the supported command set and in the way of handling ownership issues.

It is not within the scope of this paper to give a detailed elaboration of the features and differences of these remotely-owned (MRTM) and locally-owned (MLTM) Mobile Trusted Modules. For an overview of MTMs and their underlying concepts the interested reader should be referred to [18].

1.2 ARM TrustZone

In [3] and [6] ARM introduced a set of hardware-based security extension to ARM processor cores and AMBA on-chip components.

The key foundation of ARM TrustZone is the introduction of a “secure world” and a “non-secure world” operating mode into TrustZone enabled processor cores. This secure world and non-secure world mode split is an orthogonal concept to

STC'08, October 31, 2008, Fairfax, Virginia, USA.

the privileged/unprivileged mode split already found on earlier ARM cores. On a typical ARM TrustZone core, secure world and non-secure world versions of all privileged and unprivileged processor modes coexist. A number of System Control Coprocessor (CP15) registers, including all registers relevant to virtual memory, exist in separate banked secure and non-secure world versions. Security critical processor core status bits (interrupt flags) and System Control Coprocessor registers are either totally inaccessible to non-secure world or access permissions are strictly under the control of secure world. For the purpose of interfacing between secure and non-secure world a special Secure Monitor Mode together with a Secure Monitor Call instruction exists. Depending on the register settings of the processor core, IRQ- and FIQ-type interrupts are routed to Secure Monitor Mode handlers. Apart from the extensions to the processor core itself, the AMBA AXI bus in a TrustZone enabled system carries extra signals to indicate the originating world for any bus cycles. TrustZone aware System-On-Chip (SoC) peripherals can interpret those extra signals to restrict access to secure world only. In conjunction with the ability to reroute external aborts to Secure Monitor Mode handlers, a secure world executive can closely monitor any non-secure world attempts to access secure world peripherals. To summarise an ARM TrustZone CPU core can be seen as two virtual CPU cores with different privileges and a strictly controlled communication interface.

ARM has published its own TrustZone software API specification [5]. Unfortunately this API specification only defines an interface for applications wanting to interact with TrustZone protected "services" through some kind of service manager. The mentioned API specification does not cover most aspects of the backend "service provider API" interface of the service manager. At the time of writing, the author is not aware of any publically available open implementation of the API described in [5]. Consequently the ARM specified TrustZone API will *not* be considered in this paper. Furthermore within this paper, the term ARM TrustZone is only used to refer to publically available hardware documentation primarily covered by [3], [6] and [7].

Together with Trusted Logic, ARM has developed its own closed-source TrustZone software stack, complementing the TrustZone hardware extensions. Details of this software stack are given in various ARM Whitepapers, for example in [3]. This paper focusses on an independent approach, purely based on open-source software components. As a consequence, the term ARM TrustZone is used to refer to the hardware specific aspects of TrustZone technology only.

All design and prototype ideas presented in this paper are being implemented at IAIK on a TrustZone aware prototype ARMv6 processor based on the ARM1176JZF-S core. Detailed technical documentation, including a description of the TrustZone specific features of the ARM1176JZF-S core can be found at [7].

1.3 Related work

IBM has already done significant work regarding the virtualisation of Trusted Platform Modules on normal desktop platforms, by developing vTPMs [11] as an extension to the XEN hypervisor [25]. On x86 platforms, the XEN hypervisor is capable of utilising hardware isolation mechanisms, including Intel's Vanderpool and AMD's Pacifica extensions. There are ongoing efforts to port XEN hypervisor to ARM platforms, e.g. by Samsung [2] or by the "Embedded XEN" SourceForge project [1]. At the time of writing, the author is not aware of published results of working XEN ports to ARM11 platforms with TrustZone support.

Open Kernel Labs has developed an implementation of the L4 microkernel [19] with support for ARMv5 and ARMv6 based platforms. Their L4 implementation utilises hardware isolation mechanisms available on the ARMv5 and ARMv6 platforms. At the time of writing, the OKL4 source tree [20] contains rudimentary support for TrustZone specific features found an ARMv6 platforms. At the time of writing the author is not aware of any ARM TrustZone specific support code in the mainstream Linux source tree.

A number of virtualisation style approaches have been integrated into mainstream Linux kernel [23] sources: User-Mode-Linux (UML) is an approach, which allows an adapted Linux "guest" kernel to run as unprivileged process under the control of a regular "host" Linux kernel. KVM is an alternative approach on x86 kernels with hardware virtualisation extensions. The KVM device driver allows userspace applications, to act as hypervisors, taking advantage of the processor's hardware virtualisation extensions.

Perhaps the most prominent example of an application using the KVM interface is the x86 version of the platform emulator QEMU [8].

The authors of [26] investigated methods of extending mandatory access control mechanisms provided by SELinux with mobile trusted computing concepts. They especially focus on software (os-kernel) based domain isolation, by strictly controlling intra-domain channels and domain permissions.

"Seccomp" [4] is a small system-call monitor addition to the Linux kernel. It allows processes to voluntarily relinquish their ability to execute most system calls.

At ETH Zürich an open-source software emulator resembling a Trusted Platform Module [22] has been developed by Mario Strasser. Based on this TPM emulator, NOKIA researches have published a protoype of a Mobile Trusted Module emulator [17]. The vTPMs used in the XEN hypervisor are based on an adapted version of the TPM emulator too.

The authors of [21] describe ideas for the deployment of software-based Mobile Trusted Modules on virtualised platforms.

A detailed discussion of the ARM TrustZone features, including an description of the closed-source TrustZone software stack developed by ARM and and Trusted Logic is given in [24]. Parts of the API interface described in the mentioned paper are openly available in [5].

Great emphasis has been placed upon building all components of this prototype design from open-source software components and tools only. As a consequence the software prototype described in this paper contains open-source software based replacements for all components, even if closed-source commerical alternatives exist.

2. PROTOTYPE MOBILE TRUSTED PLATFORM DESIGN

Two-kernel platform design approaches are a natural fit to the TrustZone concept, as already encouraged in [24]. However for implementing a platform design following the spirit of the Mobile Reference Architecture envisioned by

the TCG in [12], just having two separate operating system worlds seems to be insufficient. In principle the TCG's Mobile Reference Architecture decomposes the platform into a set of isolated trusted engines owned by different entities. Each of those trusted engines typically has an associated MTM and well defined interfaces for communication with other trusted engines. As shown in [26], mandatory access control mechanism as provided by SELinux can be used as robust foundation for implementing the TCG trusted engine model.

This section introduces a prototype platform design, which merges the intrinsic virtualisation capabilities of ARM TrustZone with software isolation in the spirit of [26]. Figure 1 depicts an overview of the prototype platform software design currently being implemented at IAIK.

Any software running in the secure-world partition of the platform can rely on isolation guarantees offered by TrustZone hardware features.

- Secure-world peripherals can not be accessed by any non-secure world software.
- Non-secure world software is not able to access secure-world memory without authorisation by secure-world.

On TrustZone hardware platform implementations with hard-wired secure/ non-secure memory access policies, software can even rely on these policies as hardware anchor of trust.

At this point it should be mentioned, that any ARMv6 TrustZone implementation with TCMs implicitly contains at least one memory region, namely the TCM, which is capable of reconfiguring the secure/non-secure world memory access policies at runtime.

2.1 Software components of the prototype platform design

Using TrustZone features the prototype in figure 1 creates two strongly isolated system partitions. The secure world partition and its operating system kernel are in ultimate control of the whole platform. For the IAIK prototype implementation, an adapted version of the Linux 2.6.24 kernel has been chosen as basis for the secure world operating system. This secure world Linux kernel contains a number of TrustZone specific extensions, most notably it provides a special user-space interface, allowing regular secure world user-space processes to act as "hypervisor" for the non-secure world partition. A more detailed description of this user-space interface follows in section 3. For the discussion of the prototype architecture it is sufficient to know, that this TrustZone based virtualisation framework takes care of low-level details like dispatching secure monitor calls and that the framework enforces restrictions on the resource usage of the non-secure world guest VM.

Secure boot loader

A secure boot process is of ultimate importance to mobile trusted computing. It provides the basis for establishing trust on mobile and embedded devices, especially in face of mobile devices with complicated multi-ownership situations. The existence of MRTMs on a particular type of mobile or embedded device automatically implies a requirement for some kind of secure boot process.

On mobile devices where hardware MTMs are available, a secure boot process can be implemented by relying on their capabilities. However the TCG specifications do not require any particular method for implementing MTMs, more specifically the specifications explicitly allow software MTM implementations.

Platforms which only contain software MTMs cannot take advantage of having MTMs as hardware roots of trust. Such platforms have to rely on other kinds of hardware roots of trust in order to allow implementation of a secure boot process.

At the moment the IAIK prototype platform assumes that no hardware MTMs are available for implementing secure boot. Additionally the IAIK prototype platform assumes, that software MTMs are only available after the secure-world Linux kernel has invoked the secure user-space init process.

Based on these assumptions, the secure boot process of the IAIK prototype platform relies on a secure boot loader to be present on the platform. The task of this boot loader is to authenticate the secure-world Linux kernel image together with any kernel parameters, before handing over control to the Linux kernel. At the moment, the IAIK prototype platform uses a slightly adapted version of "Das U-boot" [9] as secure boot loader. The modified u-boot version is capable of measuring the Linux kernel image, its initial ramdisk and the kernel command line. Before control is handed over to the operating system kernel, these measurement values are compared to a Reference Integrity Metric (RIM) certificate[1] attached to the kernel image. The boot process only continues if the kernel's RIM certificate can be successfully validated. The measurement values obtained by u-boot are handed over to the secure-world Linux kernel and are available after the Linux kernel hands over control to the user-space init process.

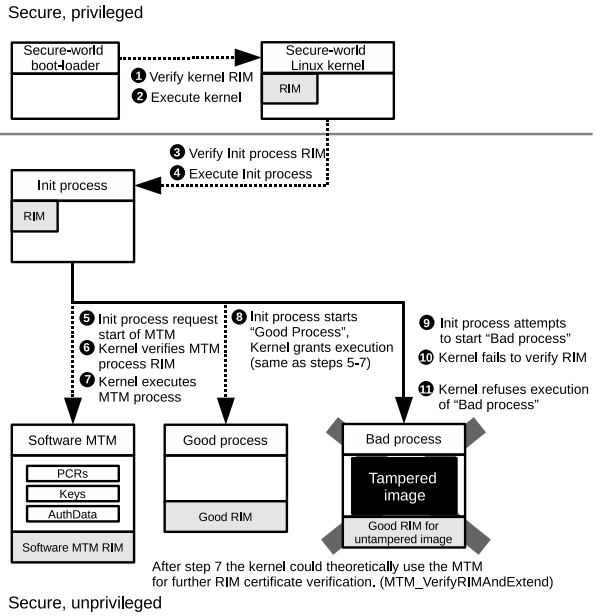

Figure 2: Secure boot on the IAIK prototype platform

[1]The RIM certificates used by the boot loader and secure world kernel are not fully equivalent to the RIM certificates described in the TCG MTM specification.

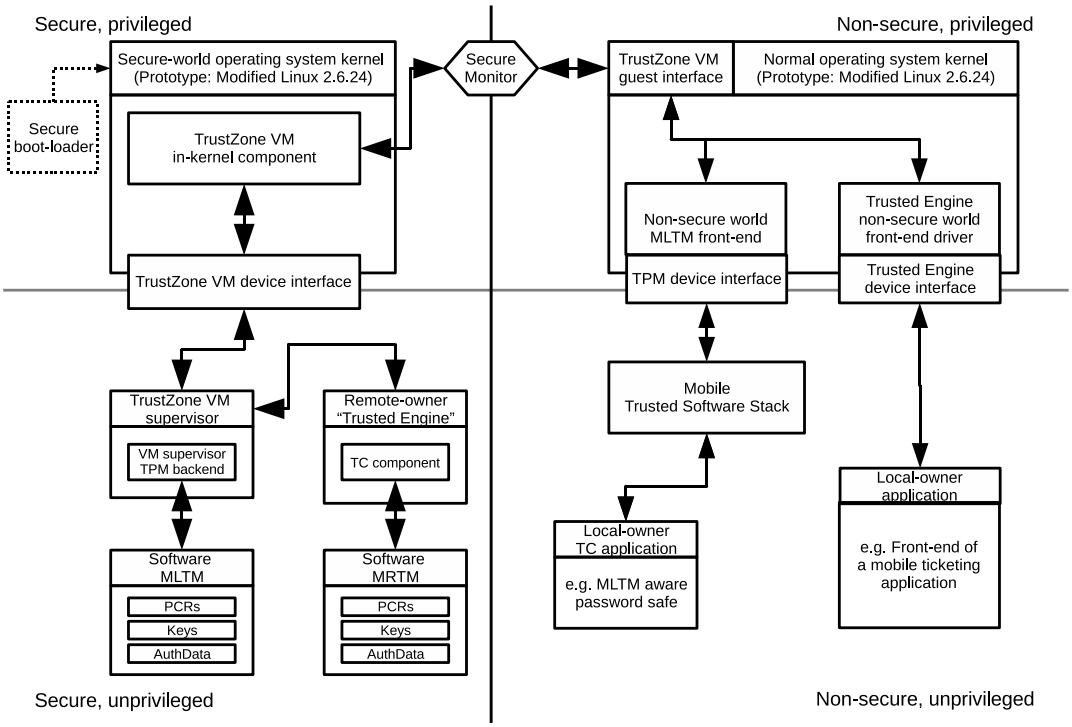

Figure 1: Prototype mobile trusted platform components

The secure world Linux-kernel has been adapted to support a similar mechanism for verifying any user-space process images and kernel modules loaded within secure-world. Any secure-world process image and kernel module must carry an embedded RIM certificate. When the secure-world kernel is about to load a process image or a kernel module, it verifies the RIM certificate. On RIM verification failure, the kernel refuses to load the offending image or module.

Figure 2 gives an overview of the secure boot approach implemented on the IAIK prototype platform. Once the software MTM is up and running it can be used by the secure-world Linux kernel for TCG-style RIM certificate verification.

Software-based Mobile Trusted Modules

The IAIK prototype platform contains a software *Mobile Local-Owner Trusted Module (MLTM)* and asoftware *Mobile Remote-Owner Trusted Module*. Both software MTMs are based on adpated versions of the open-source TPM emulator [22] and the MTM emulator [17]. For the IAIK prototype both emulators have been modified to support a TPM-emulator hosting API interface, which allows multiple instances of the emulators to be directly embedded into other applications. As shown in figure 1, the MLTM is connected to the VM supervisor application. The MLTM is started as a child process of the VM supervisor process, taking advantage of shared memory communication mechanisms. With this design it is easy to tie (virtual) platform reset, from non-secure world's point of view, to the MLTM reset.

Since the supervisor process is in full control of the initial non-secure world executable image, secure boot can be implemented easily for the *non-secure* world partition, using the software MTMs available in secure world.

The software MRTM found on the prototype platform is one of the first software components started during startup of the secure-world partition. After this MRTM is operational, a TCG-style secure-boot process can be implemented for the secure-world partition. In the final stages of this secure-world process a prototypical Trusted Engine process for the device manufacturer is started up and a connection between this process and the software MRTM is established. Similarly to the VM supervisor process, the Trusted Engine process can take advantage of any software isolation mechanism available inside the secure-world system partition, in order to archive a maximum degree of isolation towards other secure-world processes.

Trusted Engines

The TCG Mobile Reference Architecture published in [12] is based on the foundation of "Trusted Engines". These Trusted Engines are described as isolated computation environments which typically have their own private MRTMs or MLTMs. Inter-engine communication is only possible by means of well-defined interfaces exposed by the individual engines. Furthermore [12] suggests that Trusted Engines are organised hierarchically with respect to their inter-engine communication interfaces. For example, on a mobile phone platform with a manufacturer, network operator and user engine, there might be no direct interface between the manufacturer and user engine. Instead, the user engine would have to use the interfaces offered by the network operator engine in order to indirectly talk to the manufacturer engine. [12] does not prescribe any particular way of implementing Trusted Engines, given that a sufficient level of isolation among the Trusted Engines of a platform can be guaranteed.

Figure 1 shows a possible way of realising a Trusted En-

gine as isolated user-space process running within the secure-world partition of the prototype platform. As shown in [26], SELinux mechanisms available within the secure world kernel can be used to isolate this kind of Trusted Engine from other secure-world user-space processes. The Trusted Engines themselves can be implemented as byte-code interpreters as encouraged in [24].

TrustZone VM supervisor

The *TrustZone VM supervisor* is the fundamental component needed to support a non-secure world partition. This application utilises the TrustZone user-space interface exposed by the secure-world Linux kernel to act as hypervisor for the non-secure world.

Any secure monitor calls invoked by the non-secure world guest VM are routed to the user-space VM supervisor after minimal processing inside the secure-world Linux kernel. Minimising the amount of secure-world in-kernel processing, reduces the secure-world privileged mode attack surface visible to a potential adversary coming from the non-secure world partition. The attack vector for such an adversary is shifted towards the VM supervisor application running in secure unprivileged user-mode. Thus the supervisor application has to take any possible precautions against attackers trying to exploit the non-secure world interface of the VM supervisor application. The virtualisation framework described in section 3 provides mechanisms to aid the VM supervisor application with respect to defense against potential intruders coming from non-secure world. Most notably the virtualisation framework offers a secure-world/non-secure-world memory sharing mechanism which allows memory mappings to be inherited to child processes of the VM supervisor process. The size and address covered by these memory mappings is defined once at creation time of the mapping object and cannot be changed afterwards. As a consequence the VM supervisor process can be kept relatively small and simple while the actual burden of handling the bulk of workload associated with non-secure world requests can be delegated to isolated subprocesses of the VM supervisor.

To guarantee the isolation between the VM supervisor main process and its worker subprocesses, any privilege separation and access control mechanisms found inside the secure-world Linux kernel can be leveraged. This especially includes, but is not limited to, mandatory access control enforced by SELinux domains or system call usage restrictions as supported by seccomp.

3. VIRTUALISATION WITH ARM TRUSTZONE

This section outlines the virtualisation framework being developed as part of the IAIK trusted platform prototype. A primary design goal of this ARM TrustZone based virtualisation framework is the minimisation of privileged and unprivileged secure-world code required to perform virtual machine supervisor tasks.

The framework features implementation of supervisors for non-secure world guests as ordinary secure-world user-space processes. Interaction with the guests is accomplished by using the user-space interface exposed by the framework. Only a small set of critical hypercalls has to be implemented within the secure-world host kernel.

3.1 Secure and non-secure world partitioning

Depending on the exact hardware implementation of the ARM TrustZone aware SoC platform, the partitioning of peripherals and memory between secure and non-secure world can be hard-wired in silicon or can be reconfigurable by means of special platform dependent mechanisms.

The prototyping chip used at IAIK supports a reconfigurable control mechanism for assigning non-secure world access permission to peripherals and parts of internal and external memory during runtime.

Platforms with dynamic secure/non-secure world repartitioning support offer a high degree of flexibility with respect to handling non-secure world guests. Depending on the current software configuration of a non-secure world guest as reported by its associated MTM, hardware resources could be made accessible to that guest selectively. For example, "reactive" PCRs which enable or disable hardware features based on extend operations can be implemented particularly well on such platforms.

Another interesting application of dynamic secure/non-secure world repartitioning could be the concurrent exectuion of multiple strongly isolated non-secure world virtual machines. The overhead required for changing the partitioning configuration might be too high for concurrent execution of the non-secure world guests in a traditional multi-threading style approach. Nevertheless strongly isolated non-secure world guest VMs could be suspended and resumed relatively fast upon user request.

The IAIK prototype virtualisation framework takes care of repartitioning issues, by enforcing hypervisors to allocate their guest VM resources using special API calls even if no hardware repartitioning support is available. Regardless of the underlying hardware repartitioning capabilities, the IAIK prototype virtualisation framework supports concurrent execution of different non-secure world guest VMs.

3.2 Interrupts

The interrupt handling capabilities provided by TrustZone enabled ARM processor cores are quite flexible and allow for a number of different approaches to secure and non-secure world interrupt handling. As explained in [7] two special TrustZone relevant properties of IRQ- and FIQ-type interrupts can be configured by secure-world privileged executives:

- Non-secure world access permissions to the global interrupt disable bit for FIQ-type interrupts can be configured to dissalow non-secure world modifications.
- The destination for handling IRQ- and FIQ-type interrupts can be either set to the currently executing worlds' regular vectors or to special Secure Monitor Mode vectors.

As explicitly pointed out in [7] a configuration where non-secure world is not permitted to modify the FIQ disable bit combined with a Secure Monitor Mode FIQ handler vector can be used to generate "deterministic" secure interrupts.

A straightforward interrupt handling system on an ARM TrustZone core could use IRQ-type interrupts for non-secure world only, while FIQ-type interrupts could be used exclusively by secure-world. This interrupt handling strategy is applicable to software environments, which assume a single secure world operating system and a single non-secure world operating system.

Note however, that the virtualisation framework discussed in this section is not limited to a single non-secure world executive. As a consequence, IRQ-type interrupts can be targeted towards distinct, isolated non-secure world compartments running in parallel.

In order to avoid unintended cross-compartment interference among the non-secure world compartments, interrupt handling must be under total control of the secure-world kernel. A non-secure world compartment must not be able to mask or disable any interrupt sources which are allocated to other non-secure world compartments running in parallel.

To guarantee these properties in the IAIK prototype platform, non-secure world compartments are never granted direct access to either the hardware interrupt controller or to the interrupt mask bits. Instead, the compartments are provided with a virtual interrupt controller managed by the secure-world kernel, as shown in figure 3. From the non-secure world software point of view, this virtual interrupt controller consists of:

- global virtual interrupt controller status flags
- per-source virtual interrupt pending and mask flags
- secure monitor calls to interact with the secure-world part of the virtual interrupt controller

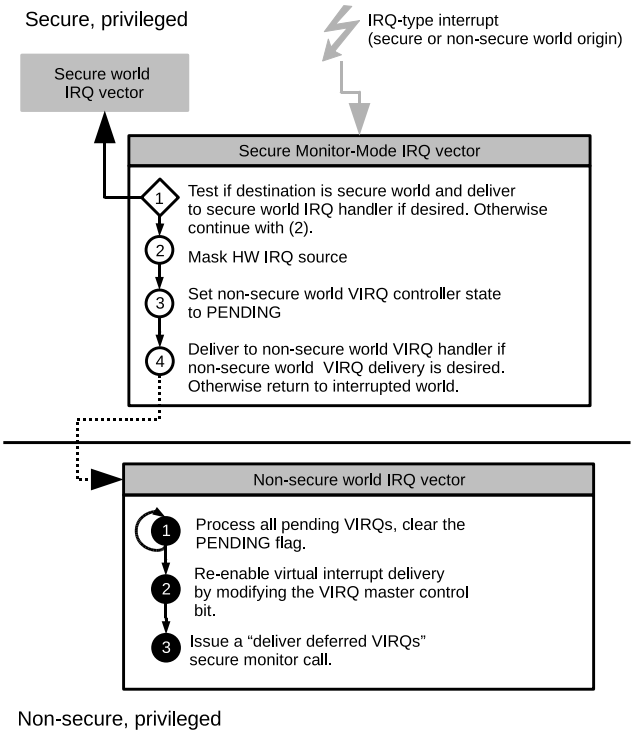

Figure 3: Virtual interrupt delivery

Secure monitor calls are not needed for most interactions with the virtual interrupt controller. For example, masking a virtual interrupt source or disabling all virtual interrupts can be accomplished by the non-secure world operating system by writing to the appropriate status bits in the virtual interrupt controller's memory region. Reenabling individual virtual interrupt sources or the whole virtual interrupt controller works a bit different. The secure world kernel can not automatically detect when the non-secure world operating system writes to the virtual interrupt controller's memory region.

The lack of automatic notification of the secure world kernel is not a problem when disabling virtual interrupts. Actually the secure world part of the virtual interrupt controller has to check the "disabled" state of a virtual interrupt, when the corresponding hardware interrupt source actually fires. If the virtual interrupt target is disabled at this time, the secure world part masks the hardware interrupt source and records the virtual interrupt for deferred delivery.

When reenabling virtual interrupt sources, the non-secure world operating system has to explicitely notify the secure world part by means of a secure monitor call. Without such an explicit notification, reliable interrupt delivery to the non-secure world guest can not be guranteed.

Support for virtual interrupt sources has been fully integrated into the guest VM resource management of the virtualisation framework prototype. As a consequence, the secure-world part of the virtual interrupt subsystem has to be aware of hardware interrupts routed to different non-secure world guest VMs than the currently active guest VM. At the moment this issue is handled by queueing the interrupt for the destination guest VM, without suspending the active guest VM.

3.3 Userspace supervisor interface

For good reasons, there is no direct way for secure-world userspace applications to trigger a switch to non-secure world without going through a system call to the secure world kernel. An even stronger property holds for non-secure world executives. Their only way[2] to cause a switch to secure-world is to execute the secure monitor call instruction. Due to its reliance on privileged instructions and operations the low-level core functionality of the Linux TrustZone virtualisation extensions have to reside within the secure-world kernel.

Nevertheless it is possible to move a large amount of code required to supervise a non-secure world guest VM into secure-world userspace. Figure 4 gives a detailed overview, of the user-space interface to the virtualisation framework currently being developed at IAIK.

In the lower-part of figure 4 a minimal set of operations required to implement a secure-world userspace supervisor process for the non-secure world guest compartment is depicted. The steps performed by the supervisor application in the figure can be summarized as:

Opening the TrustZone VM interface

Any communication between a supervisor application and the userspace supervisor interface is based on traditional Linux file I/O. This design approach follows the UNIX philosophy of "everything is a file" and has inherent advantages over a specialised system call based approach. Exposing the userspace virtualisation interface through a device file, allows any existing file access control mechanisms available inside the Linux kernel to be used.

The first step to be performed by a supervisor application is to open the "/dev/trustzone" device file.

[2]Ignoring external aborts and IRQ/FIQ type interrupts

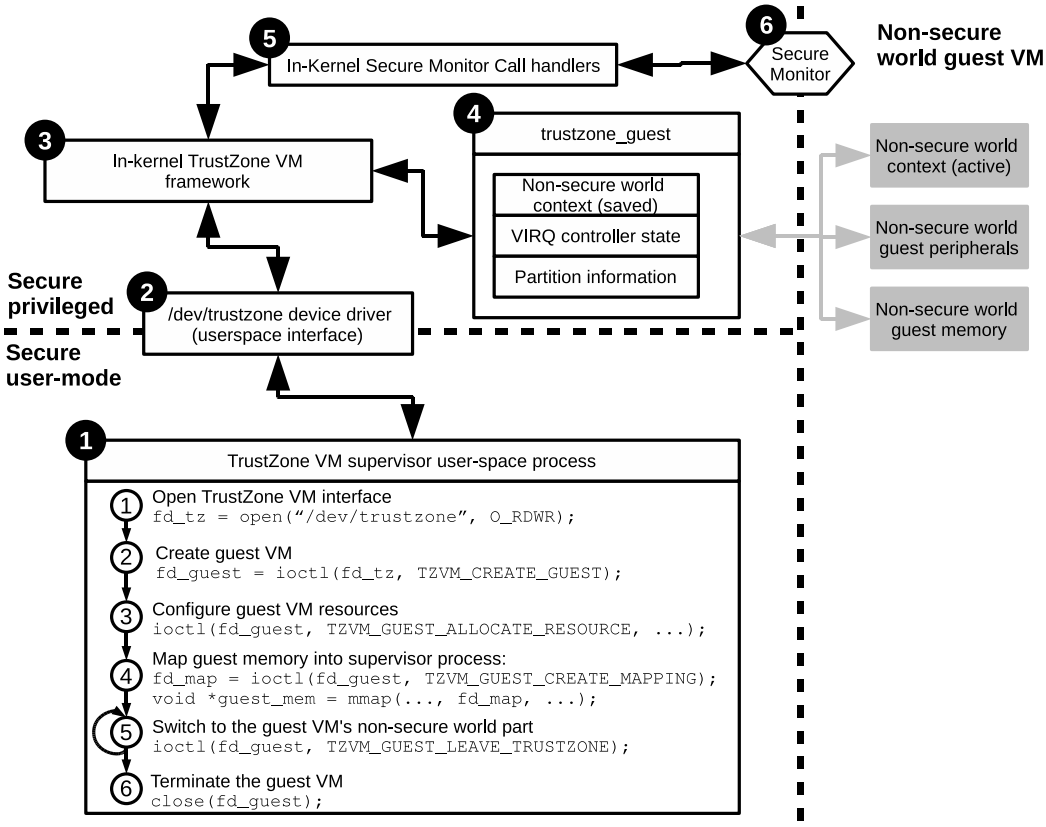

Figure 4: Userspace supervisor interface

Creating a guest VM

The next step a typical supervisor application will take is to create a new guest VM without any associated resources. To perform this step, the supervisor application has to issue an *ioctl(TZVM_ IOCTL _CREATE_VM)* call on the file descriptor for the TrustZone device file opened during the previous step. Upon success, an in-kernel object representing the new guest VM and its state is created. This object is presented to user-space through a new file descriptor returned by the *ioctl(. . .)* call.

Once the file descriptor for the guest VM has been created, the supervisor application can safely close the device descriptor for the TrustZone device.

Configuring guest VM resources

The guest VM created during the preceding step does not yet have any resources associated with it. It is the responsibility of the supervisor application to allocate resources for the guest VM by issuing *ioctl(TZVM_ IOCTL _ALLOCATE_RESOURCES)* calls. It is not mandatory to allocate all resources used by the guest VM before the first switch to non-secure world. At the moment the only way to deallocate guest resources is to destroy the associated guest VM. Currently supported types of guest VM resources include:

- non-secure world memory
- non-secure world peripherals and coprocessors
- virtual interrupt sources

When allocating virtualised interrupts, the supervisor application optionally can specify IRQ-type secure-world hardware interrupt sources to be used as trigger for the virtualised interrupts.

Any resource allocation is checked by the in-kernel component of the TrustZone virtualisation framework. Thus a secure-world supervisor application can not perform any allocations considered invalid by the in kernel framework. If multiple supervisor applications and multiple guest VMs are running concurrently, the in-kernel resource manager takes care of prohibiting any conflicting allocations.

Depending on the precise nature of the SoC platform implementation, the in-kernel resource manager has to perform additional steps, as already discussed in section 3, to make the allocated resources accessible to the non-secure world guest VM.

Making guest VM memory accessible to the supervisor process

The supervisor userspace interface facilitates communication with the guest VM using shared-memory by providing a mechanism for creating mappings of parts of the guest VM's physical memory. A mapping object is simply a file-descriptor usable with the standard Linux *mmap(...)* family of system calls.

Mapping objects are created by issuing an *ioctl(TZVM_ IOCTL _CREATE_MAPPING)* call on the guest VM file descriptor. When creating a mapping, the in-kernel TrustZone

virtualisation framework asserts that the requested mapping range has been allocated previously to the guest VM. The supervisor userspace interface explicitly allows the creation of multiple overlapping mappings for the same guest VM. This feature is particularly useful to supervisor implementations, who want to execute parts of code in separate subprocess contexts using a combination of the *fork(...)/clone(...)* and *execv(...)* system calls. Such hypervisors could inherit the file descriptor belonging to the mappings required for performing a particular task to their subprocesses. The subprocesses themselves could run as totally unprivileged secure-world userspace processes. Isolation towards the supervisor process can be increased by taking advantage of mechanisms similar to the "seccomp" framework available in all recent Linux kernel [4].

Switch to non-secure world and handle requests from non-secure world

There are still a few remaining steps to be done by the secure-world supervisor application, before the non-secure world guest VM can be invoked. Most importantly, the supervisor application has to ensure that the register context of the non-secure world guest VM is set up appropriately. For this reason a number of special supervisor userspace interface *ioctl(...)* calls not shown in figure 4 exist. Using these operations, the supervisor application has full access to parts of the general purpose and system control coprocessor register context relevant to the non-secure world guest VM. If a normal operating system kernel is to be run inside the non-secure world guest VM, the register context setup performed by the supervisor application can emulate the behavior of a normal bootloader. It should be denoted, that the TrustZone virtualisation framework presented in this paper is not limited to emulating a classic boot loader for the non-secure world.

In more advanced setups the supervisor application could do any necessary privileged operations and MMU configuration for the guest VM. In such scenarios, the executive inside the guest VM could be forced to run in unprivileged non-secure user-mode without any possibility to change their page tables, etc. on their own. If this setup is enforced for all guest VMs, the TrustZone virtualisation framework offers the possibility of having multiple strongly isolated guest VMs running concurrently in non-secure world.

Ultimately at some point in time, the supervisor applications has to yield execution flow to the non-secure world guest executive. This task is accomplished by invoking the special *ioctl(TZVM_ IOCTL _LEAVE)* call on the guest VM file descriptor. The in-kernel component of the TrustZone virtualisation framework takes care of saving the secure-world context and loading the non-secure world context. In order to be able to simultaneously access the secure and non-secure versions of banked system control coprocessor registers, the actual world switching code has to run within secure monitor mode. To keep things clean and simple, the implementation being currently developed at IAIK uses a special secure monitor call for realizing the context save/load operations and the actual world switching.

Finally the non-secure world executive is scheduled by performing a return-from-exception instruction from secure monitor mode, with the non-secure bit of the CP15 Secure Configuration Register set. There are several possibilities to reenter secure-world, while the non-secure world executive is running. Interrupts and external aborts have already been discussed before and won't be considered here again.

At this point, the focus should by directed towards Secure Monitor Call instructions issued by the non-secure world executive in order to request services from its secure-world supervisor process.

The secure-world Linux prototype implementation supports two different in-kernel handling mechanisms for secure monitor calls originating in non-secure world. A low-level in-kernel mechanism without the overhead of performing a full world switch is provided for implementing monitor call handlers with low-latency requirements. However these low-level in-kernel monitor call handlers suffer from similar restriction as interrupt handlers, most notably they must not perform any actions which might sleep or cause a secure-world task switch.

For more-complex secure monitor call handlers, which rely on the capability to sleep, the IAIK prototype provides an in-kernel high-level handler mechanism. Using such a high-level handler however incurs the overhead of performing a full world switch on handler entry. If the non-secure world executive is to be invoked after the handler has completed its task, an additional second full world switch back to non-secure world is needed.

```
... // Initialize the guest VM, setup resources, ...
while (!guestvm_terminated) {
  unsigned int reason;
  int result = ioctl(guestvm_fd, TZVM_IOCTL_LEAVE,
                     &reason);
  if (result) {
    ... // Virtualisation framework error
  }

  if (tzvm_status_code(reason) == TZVM_STATUS_SMC) {
    ... // Handle a Secure Monitor Call
  }
...
} ...
```

Figure 5: Main loop of a typical userspace supervisor application (pseudo-code)

When no in-kernel handlers are available for a specific secure monitor call, the TrustZone virtualisation framework returns control to the secure-world supervisor process. For this reason, the *ioctl(TZVM_ IOCTL _LEAVE)* call includes a parameter which indicates the reason for re-entering secure world. From the supervisor process's point of view, the switch back to secure-world simply looks as if the *ioctl(TZVM_ IOCTL _LEAVE)* call has just returned. Consequently a typical supervisor application can be expected to contain a "main loop" similar to the pseudo-code shown in figure 5.

Terminating the guest VM and release its resources

Finally, a userspace supervisor can dispose a guest VM and release any associated resources by simply closing the guest VM file descriptor with the standard *close(...)* Linux system call. Once the last reference to the guest VM file descriptor has been released, the in-kernel virtualisation framework takes care of deallocating any resources and of disposing the guest VM object itself. The guest VM file descriptor exhibits the same cleanup semantics with respect to process termination as any other standard Linux file descriptor. As

a consequence guest VM resources are cleaned up even if the supervisor process crashes or does not explicitly close the guest VM file descriptor before normal termination.

4. FURTHER DIRECTIONS OF WORK

At the time of writing, the Linux implementation of the virtualisation framework presented in this paper is rapidly progressing towards completion. Currently the virtualisation framework is already able to boot an adapted Linux kernel with minimal user-space support as non-secure world guest operating system.

Since the principles inherent to the virtualisation framework architecture are not specific to Linux, future development efforts will be concentrated on preparing the framework for easy porting to other secure-world operating systems. Eventually the feasibility of integrating the TrustZone based framework with the KVM virtualisation framework found in recent Linux kernels will be considered.

The software platform prototype described in this paper has been realised on a prototype hardware implementation of the ARM1176JZF-S processer. At the time of writing the particular prototyping hardware used at IAIK is not (yet) publically available. It is however intended to port the software described in this paper to publically available hardware platforms. A particular interesting candidate target platform, is the BeagleBoard[3] featuring an ARMv7-based OMAP3530 processor from Texas Instruments.

As part of reducing platform and architecture dependencies, development of a user-space library which encapsulates the low-level details of Linux-specific system calls like *ioctl(...)* has already been started. Once this library has been completed, it should be possible to port applications using the virtualisation framework user-space interface easily to other non-Linux operating systems, given that the privileged mode components and the wrapper library have been ported. A particular promising candidate for such porting attempts is the L4 micro-kernel.

The prototype platform described in this paper aims at providing a Trusted Computing environment based on software implementations of Trusted Mobile modules. In the paper several mechanisms usable to provide isolated environments suitable for running software MTMs have been mentioned.

Once hardware MTMs become available, it will be interesting to analyse ways how to integrate these modules on platforms where they coexist side-by-side with software MTMs. As a first step the IAIK prototype platform will consider hardware MTM implementations in the spirit of [10]. As part of development of the IAIK prototype platform, reasearch effort is done towards definition and implementation of a low-level and high-level MTM abstraction APIs. The low-level API interface is intended to provide a uniform mechanism of accessing hard- and software MTMs by the secure boot loader. The high-level API has the same goal for the secure world operating system kernel and for secure-world user-space.

While this paper introduces a new virtualisation framework based on ARM TrustZone, it does not yet cover the security properties of the proposed system in great detail. It will be part of further research to produce a detailed analysis of the exact security properties of the proposed system.

[3]see `http://www.beagleboard.org/`

5. CONCLUSIONS

In this paper a novel approach for integrating the Linux-kernel with ARM TrustZone features has been introduced. At the secure-world end this approach focuses on providing a robust and portable virtualisation framework for handling non-secure world guests. The virtualisation framework attempts to allow supervision of non-secure world guest VMs with minimal secure-world privileges, in particular by reducing the amount of secure privileged code. Results obtained with the IAIK prototype implementation of the virtualisation framework outlined in this paper are quite encouraging. Especially debugging of the user-space supervisor process for the non-secure world VM turned out to be relatively straightforward in comparison to purely kernel-base virtualisation approaches.

On the basis of this virtualisation framework a prototype design for a trusted mobile platform has been introduced. In particular the prototype design attempts to implement Mobile Trusted Modules on a software-only basis, without the need for additoinal special purpose hardware. As a consequence the prototype design has to reuse hardware anchors of trust implicitly found in the ARM TrustZone architecture or the specific platform implementation in order to provide a secure-boot process.

Finally this paper demonstrates the feasibility of designing and implementing embedded trusted computing software platforms, which take advantage of advanced hardware security mechanisms, solely on the basis of well-known open source components.

6. REFERENCES

[1] *Embedded XEN*. Available online at: `http://sourceforge.net/projects/embeddedxen/`.

[2] *Secure Architecture and Implementation of Xen on ARM for Mobile Devices*. Presentation slides available online at: `http://xensource.com/files/xensummit_4/Secure_Xen_ARM_xen-summit-04_07_Suh.pdf`.

[3] T. Alves and D. Felton. *TrustZone: Integrated Hardware and Software Security - Enabling Trusted Computing in Embedded Systems*. Available online at: `http://www.arm.com/pdfs/TZ_Whitepaper.pdf`, July 2004.

[4] A. Arcangeli. seccomp. Import into mainstream Linux kernels: `http://kernel.org/hg/linux-2.6/file/cfe426c10480/kernel/seccomp.c`, 2005.

[5] ARM. Trustzone api specification, June 2006. PRD29-USGC-000089, v2.0.

[6] ARM Ltd. *TrustZone Technology Overview*. Introduction available at: `http://www.arm.com/products/esd/trustzone_home.html`.

[7] ARM Ltd. *ARM1176JZF-S Technical Reference Manual, Revision: r0p7*. Available online at: `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301g/DDI0301G_arm1176jzfs_r0p7_trm.pdf`, 2008.

[8] F. Bellard. Qemu open source processor emulator. Available online at: `http://bellard.org/qemu/`.

[9] W. Denk et al. Das u-boot – the universal boot loader. Available online at: `http://www.denx.de/wiki/UBoot/WebHome`.

[10] K. Dietrich. An integrated architecture for trusted computing for java enabled embedded devices. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 2–6, New York, NY, USA, 2007. ACM.

[11] S. B. et al. *vTPM: Virtualizing the Trusted Platform Module*. Available online at: `http://www.research.ibm.com/`, February 14 2006.

[12] T. C. G. .-M. W. Group. *TCG Mobile Reference Architecture Version 1.0 Revision 1*. Specification available online at: `https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-reference-architecture-1.0.pdf`, 12 June 2007.

[13] T. C. G. .-M. W. Group. *TCG Mobile Trusted Module Sepecification Version 1 rev. 1.0*. Specification available online at: `https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf`, 12 June 2007.

[14] T. C. G. .-T. W. Group. *TCG Software Stack (TSS) Specification Version 1.2 Level 1*. Specification available online at: `https://www.trustedcomputinggroup.org/specs/TSS/TSS_Version_1.2_Level_1_FINAL.pdf`, 6 January 2006. Part1: Commands and Structures.

[15] T. C. G. .-T. W. Group. *TPM Main Part 2 Structures*. Specification available at: `https://www.trustedcomputinggroup.org/specs/TPM/mainP2Structrev103.zip`, 9 July 2007. Specification version 1.2 Level 2 Revision 103.

[16] T. C. G. .-T. W. Group. *TPM Main Part 3 Commands*. Specification available online at: `https://www.trustedcomputinggroup.org/specs/TPM/mainP3Commandsrev103.zip`, 9 July 2007. Specification version 1.2 Level 2 Revision 103.

[17] M. K. Jan-Erik Ekberg. *MTM implementation on the TPM emulator*. Available online at: `http://hemviken.fi/mtm/index.html`.

[18] M. K. Jan-Erik Ekberg. *Mobile Trusted Module (MTM) - an introduction*. Available online at: `http://research.nokia.com/files/NRCTR2007015.pdf`, November 14 2007.

[19] O. K. Labs. *OKL4*. Available only at: `http://www.ok-labs.com/products/okl4`.

[20] O. K. Labs. *OKL4 microkernel source code, release 1.5.2*. Available online at: `http://wiki.ok-labs.com/images/2/20/Okl4_release_1.5.2.tar.gz`.

[21] A. U. Schmidt, N. Kuntze, and M. Kasper. On the deployment of mobile trusted modules, 2007.

[22] M. Strasser. *TPM Emulator*. Software package available at: `http://tpm-emulator.berlios.de/`.

[23] L. Torvalds et al. The linux kernel archives. Available online at: `http://www.kernel.org/`.

[24] P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves. Implementing embedded security on dual-virtual-cpu systems. *IEEE Design and Test of Computers*, 24(6):582–591, 2007.

[25] *XEN Hypervisor*. Available online at: `http://xen.org/`.

[26] X. Zhang, O. Aciicmez, and J.-P. Seifert. A trusted mobile phone reference architecture via secure kernel. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 7–14, New York, NY, USA, 2007. ACM.