The Untapped Potential of Trusted Execution Environments on Mobile Devices

Jan-Erik Ekberg[1], Kari Kostiainen[2], and N. Asokan[3]

[1] *Trustonic. E-mail: jan-erik.ekberg@trustonic.com*
[2] *ETH Zurich. E-mail: kari.kostiainen@inf.ethz.ch*
[3] *University of Helsinki and Aalto University. E-mail: asokan@acm.org*

Abstract

Nearly every smartphone, and even the occasional feature phone, today contains a hardware-based trusted execution environment (TEE). Smartphones with TEEs first appeared almost a decade ago, but their use has been limited – application developers have not had the means to make use of TEEs to improve the security (and usability) of their applications. In this article, we discuss why TEEs are so widely deployed in mobile devices, and what kind of capabilities they support. We then describe Nokia Research Center's On-board Credentials (ObC) system which opens up the device TEE to application developers and some example applications that make use of ObC. We conclude by briefly outlining recent developments in standardizing TEE functionality.

Keywords: trusted execution environments; mobile devices

1 Introduction

A *trusted execution environment* (TEE) is a secure, integrity-protected processing environment, consisting of processing, memory and storage capabilities. It is *isolated* from the "normal" processing environment, sometimes called the *rich execution environment* (REE), where the device operating system and applications run. As mobile operating systems grow in size and complexity, they are increasingly susceptible to software vulnerabilities. TEEs were designed to provide protection against attacks that exploit such vulnerabilities. TEEs make it possible to design REE applications and services that remain secure even in the face of OS compromise by partitioning them so that sensitive operations are restricted to the TEE and sensitive data, like cryptographic keys, never leave the TEE.

In our daily lives, we encounter more and more services that use dedicated hardware tokens to improve their security: examples include one-time code tokens for two-factor authentication, wireless tokens for opening doors in buildings or cars, tickets for public transport. Mobile devices equipped with TEEs have the potential for replacing these many tokens thereby improving the usability for users while also reducing the cost for the service providers without hampering security.

Chances are that the mobile device in your pocket sports a hardware-based TEE. Chances are, too, that you have not come across too many applications that actually make use of TEE functionality. In this article, we explain why this situation came to pass and what the future may hold.

Security in mobile world had a very different trajectory compared to the world of personal computers [12]. The various stakeholders had strict security requirements, some of which date back to two decades ago, right at the beginning of the explosion of personal mobile communications. Some examples of such requirements are:

- standardization requirements recommending that the device identifier (also known as International Mobile Equipment identifier or IMEI) should resist "manipulation and change, by any means (e.g., physical, electrical and software)" as phrased in 3GPP specification [1],

- regulatory requirements like ensuring secure storage for radio frequency parameters calibrated during manufacture,
- business requirements like *subsidy lock* (ensuring that subsidized mobile phones given to a subscriber as part of a contract with a mobile operator cannot be used by a subscriber of a different mobile operator) and secure implementations of digital rights management schemes, and
- end user perceptions which had grown accustomed to the reliability of early feature phones (e.g., no blue screen of death).

These requirements incentivized mobile device manufacturers, chip vendors and platform providers to design and deploy hardware and platform security mechanisms for mobile platforms from early on. Hardware-based TEEs were seen as essential building blocks in meeting these requirements. The first mobile phones with hardware-based TEEs appeared almost a decade ago in the form of Nokia phones using processors from Texas Instruments [17]. One way to realize a TEE is by implementing a secure processor mode. The primary example of such an implementation is ARM TrustZone [3]. ARM processors capable of TrustZone are deployed in the overwhelming majority of smartphones and tablets today. Almost every smartphone and tablet today contains a TEE like ARM TrustZone (as well as software platform security mechanisms [12]).

Despite such a large-scale deployment, the use of TEE functionality has been largely restricted to its original intended uses. There has been no widely available means for application developers to benefit from existing TEE functionality. Fortunately, with emerging standardization this situation is about to change.

In the rest of this article, we discuss the security features provided by TEEs and describe *On-board Credentials* (ObC), a system that we developed at Nokia for safely opening up access to TEE functionality for application developers. We discuss on-going TEE standardization activities and conclude by providing an outlook for the future of TEEs.

2 What makes a TEE?

A typical TEE provides platform boot integrity, secure storage, device identification, isolated execution, and device authentication capabilities. Figure 1 illustrates these security mechanisms, and in the following text we gradually introduce concepts shown in Figure 1.

2.1 Security mechanisms

The mobile device has a *hardware trusted computing base* (TCB) consisting of hardware and firmware components that need to be trusted unconditionally. Mobile platform boot integrity can be verified using either *secure boot* or *authenticated boot*. In secure boot, the device start-up process is stopped, if any modification of the launched platform components is detected. A common approach to implement secure boot is to use code signing combined with making the beginning of the **boot sequence** immutable by storing it within the TCB (e.g., on ROM of the mobile device processor chip) during manufacturing; the processor must unconditionally start executing from this memory location. **Boot code certificates** that contain hashes of booted code, and are signed with respect to a device **trust root**, such as the device manufacturer public key that is immutable on the device, can be used to verify the integrity of the booted components. The TCB must be enhanced with **cryptographic mechanisms** that validate the signature of the system component launched first (e.g., the boot loader) that can in turn verify the next component launched (e.g., the OS kernel) and so on. If any of these validation steps fail, the boot process aborts. Integrity of the cryptographic mechanisms can be ensured by storing the needed algorithms on ROM. Secure boot with code signing does not necessarily imply that only a single platform version is allowed to be started – there may be several certified alternatives.

In authenticated boot, the started platform components are measured, but not verified with respect to certified reference values. During the boot, measurements of launched software components, and their configuration data, are stored on integrity-protected **volatile memory**. The boot loader measures the first component launched which in turn will measure the next one and so on. These measurements represent the

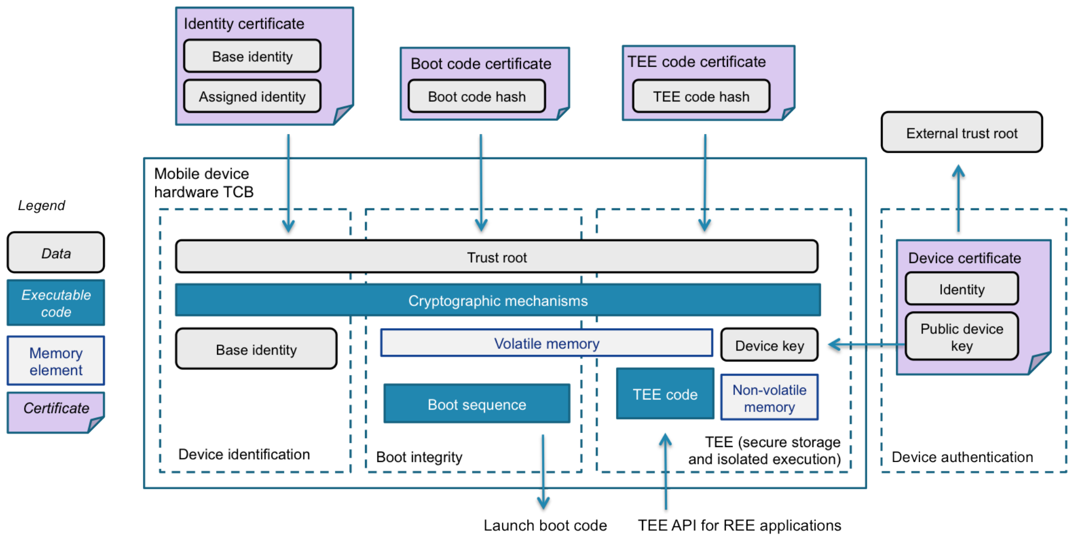

Figure 1: An overview of common hardware-security mechanisms in mobile devices. The trusted execution environment (TEE) consists of secure storage and isolated execution mechanisms. REE applications access these security services through TEE API.

state of the platform components after the boot, and can be used to control access of booted system software to hardware-protected device resources or for remote attestation (see below).

Implementation of *secure storage* requires at least one confidential, device-specific key that can be accessed only by authorized code within the TCB. Such a **device key** may be initialized during manufacturing and stored in a protected memory area on the processor chip. In addition, secure storage also requires trusted implementations of necessary cryptographic mechanisms: e.g., an authenticated encryption algorithm. Data rollback protection, the ability to detect old versions of protected objects in the local persistent storage, requires inclusion of some writable **non-volatile memory** (e.g., a monotonic counter) that persists its state across device boots.

The secure storage mechanism combined with a set of pre-defined cryptographic algorithms can be used to expose the functionality of such algorithms to the REE with the guarantee that the cryptographic keys never leave the hardware TCB. While pre-defined common cryptographic operations are sufficient for many services, certain REE applications require execution of application-specific algorithms in isolation from the mobile OS and the rest of the REE (*isolated execution*). Proprietary one-time password algorithms for online banking constitutes one such example. To extend the secure storage mechanism for isolated execution of arbitrary code, the device hardware configuration must provide an interface (**TEE API**) through which the executable code (**TEE code**) can be loaded for execution using the protected volatile memory. A **TEE code certificate** that contains a hash of the TEE code, and is signed with respect to the device trust root, can authorize code execution within the TEE and authorize TEE code to access the device key. Furthermore, the access that any TEE code has to the device key may be controlled based on the platform state that was measured and saved on volatile, integrity-protected memory during an authenticated boot process.

The hardware TCB instance typically has a unique immutable **base identity** which may be a serial number from a managed namespace or a statistically unique identifier initialized randomly at manufacture. A combination of a trust root and the base identity allows flexible *device identification*; an **identity certificate** that is signed with respect to the aforementioned trust root binds an assigned identity to the base identity. IMEI and link-layer identities like Bluetooth and WiFi addresses are examples of device identities. A **device certificate** that is signed by the device manufacturer can bind any assigned identity to the public part of the device key. Signatures using the device key provide *device authentication*, while signed statements over volatile memory that contains measurements from an authenticated boot enable device state reporting

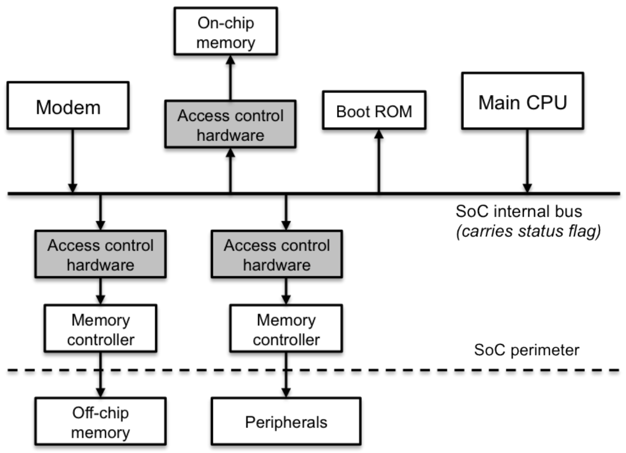

Figure 2: Example hardware configuration for a TrustZone-enabled mobile device. The access control between hardware elements is implemented using a *status flag* that determines the processor mode: secure world or normal world. The system-on-chip (SoC) communication bus carries the status flag, and dedicated access control hardware (shaded boxes) are added to the device hardware configuration for access control enforcement.

and verification (*remote attestation*). The hardware TCB may also contain a hardware-based source of randomness.

2.2 Hardware-security architectures

ARM TrustZone [3] and TI M-Shield [17] are hardware architectures that realize these security mechanisms in current mobile devices. In a common mobile device hardware configuration, the device main processor, small amounts of ROM and RAM, some peripheral and interrupt controllers, and debug and trace ports are integrated to a single chip (*system-on-chip* or SoC). These *on-chip* components are connected with an internal bus. The rest of the mobile device components, such as the system main runtime memory, flash memory elements, the display, and antennas are typically implemented as external components. Such *off-chip* hardware elements are connected with an external device bus. Figure 2 illustrates such a hardware configuration.

The processor can operate in one of two modes: *normal world* or *secure world*. The processor boots into the secure world which sets up the necessary environment before switching to the normal world. Execution can switch back to the secure world when a special command is executed in the normal world. The secure world is intended for the TEE while the normal world runs the REE.

The designer of a mobile device hardware configuration defines the hardware components that are accessible in these two modes. In a typical configuration, access to on-chip elements is restricted to the secure world only, while the device main memory and user input and output interfaces are accessible in both modes. The access control is implemented based on a status flag indicating the current world which is made visible over the internal and external bus communication. Access control hardware that enforces the status flag is added to mobile device configuration (see Figure 2). The switch from the normal world to the secure world is only possible via a controlled transition from the less privileged mode to the more privileged mode. The on-chip ROM is populated during device manufacturing to contain the base identity, device key, trust root, and the cryptographic algorithms. The on-chip RAM is used for isolated execution at runtime.

TrustZone software architecture is illustrated in Figure 3. The mobile OS in REE accesses the TEE services through a TrustZone library and a hardware driver. Within the TEE, *trusted applications* are executed on top of a minimal runtime environment, called *trusted OS*. The trusted OS provides a TEE internal API using which trusted applications can communicate with REE applications and access cryptographic

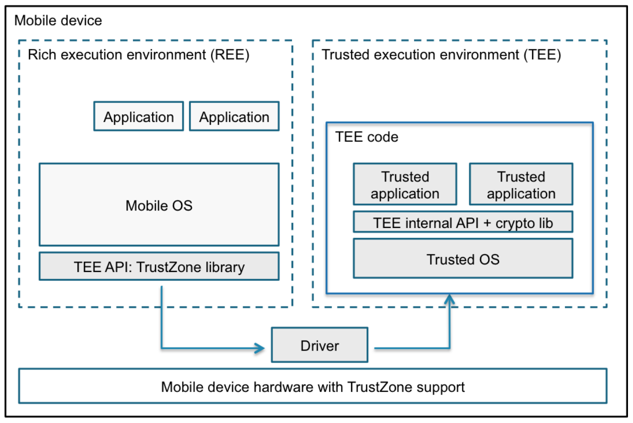
Mobile device
Rich execution environment (REE)
Application
Application
Mobile OS
TEE API: TrustZone library
Trusted execution environment (TEE)
TEE code
Trusted application
Trusted application
TEE internal API + crypto lib
Trusted OS
Driver
Mobile device hardware with TrustZone support

Figure 3: Software architecture for TrustZone-enabled mobile devices. Trusted applications are executed within the TEE that is isolated from the REE running the mobile OS and applications. REE applications access TEE security services through a device driver.

operations and secure storage functionality. The trusted OS can enforce access control on trusted applications that attempt to access the device key. The TrustZone architecture *does not* define how REE applications access TrustZone services.

Boot time trustworthiness of REE-side system software components, including the TrustZone library, can be verified with hardware-assisted secure boot. Even if the adversary would manage to compromise these software components at runtime, he would not gain access to TEE-protected secrets; at most the adversary could cause denial-of-service for TEE-provided security services.

2.3 Other types of TEEs

The most common approach in today's mobile devices is to realize the TEE using processor security features like isolated memories and a secure processing mode, as discussed above. There are also other ways to realize a TEE:

Virtualization: A securely booted virtual machine monitor (VMM) allows many virtualized "guest" OSs to run concurrently on the device. The isolation of the guests is arranged by virtue of the VMM running at a higher protection domain than the OSs, and mastering the memory management unit on behalf of the OSs. Approaches like Overshadow [5] can even extend similar protection to individual applications inside a potentially harmful guest OS. Solutions based on virtualization typically rely on software to guarantee integrity and isolation, but are otherwise architecturally similar to hardware-based TEEs.

Dynamic Roots of Trust: In x86-based mobile devices like laptops, TEEs can be realized using the Dynamic Roots of Trust (DRTM) specified by the Trusted Computing Group (TCG) (`www.trustedcomputinggroup.org`) as demonstrated by Flicker [13]. A DRTM-enabled processor acting in concert with a Trusted Platform Module (TPM) can launch and run small pieces of trusted software, isolated from the OS. Such TEEs are isolated by hardware but active only for short periods at a time.

Embedded secure elements: Some commercial mobile phone models, intended for payment applications that have high security requirements, include embedded smart cards on their motherboards. These TEEs often use JavaCard as their programming environment. The cards provide full hardware isolation. Embedded smart cards, however, rarely provide open interfaces and support for more general 3rd-party programming.

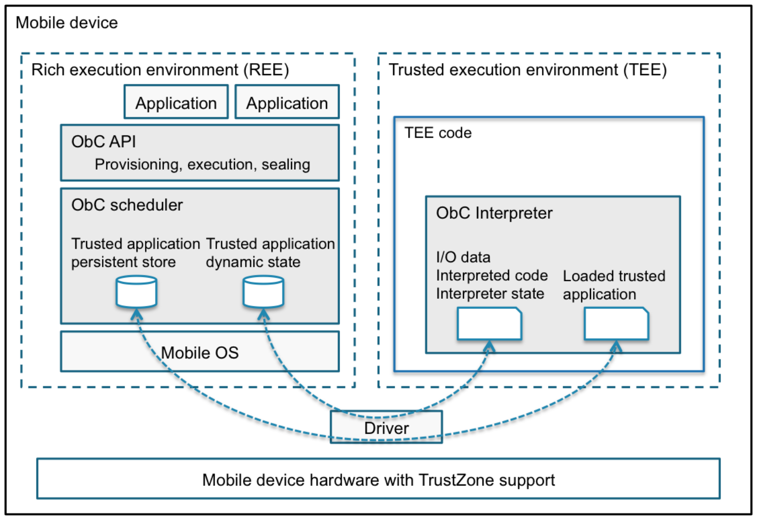
Mobile device
Rich execution environment (REE)
Application
Application
ObC API
Provisioning, execution, sealing
ObC scheduler
Trusted application persistent store
Trusted application dynamic state
Mobile OS
Trusted execution environment (TEE)
TEE code
ObC Interpreter
I/O data
Interpreted code
Interpreter state
Loaded trusted application
Driver
Mobile device hardware with TrustZone support

Figure 4: On-board Credentials (ObC) architecture. Shaded boxes illustrate ObC components. Trusted applications are executed within the TEE by the *ObC Interpreter*. The *ObC scheduler* maintains trusted application persistent storage and handles execution scheduling. REE applications use ObC services through the *ObC API*.

3 TEE access for developers

Although TrustZone and M-Shield architectures have been deployed to many mobile devices for almost a decade, the usage of hardware-security mechanisms by third-party developers has been limited. Traditionally, mobile device manufacturers have leveraged TEE capabilities only for their *internal* use cases, such as subsidy lock protection and secure boot.

Some mobile platforms provide hardware-assisted security APIs. Java ME application framework that is widely used in feature phones defines JSR 177 API [14] . The JSR 177 provides a low-level communication interface to a smartcard-like secure element and a more high-level cryptographic API. The implementation may be provided by a mobile phone secure element such as SIM card. Recent versions of the Android platform expose hardware-assisted cryptographic APIs [8] in the form of standardized PKCS11 [16] interface, while in iOS similar functionality is provided through a proprietary key chain API [2].

The high-level hardware-security APIs described above have been modeled on the usage paradigms of hardware security modules (HSMs), cryptographic tokens and smart cards. They allow the creation of hardware-protected keys, and common cryptographic operations, such as encryption and signatures, on them or assume. To take advantage of programmability of mobile TEEs, a different kind of API abstraction is needed. The API should address provisioning of trusted applications and secrets to the device, authorization of trusted applications to access provisioned secrets and device keys, and control which REE application can execute trusted applications. None of the current standardized, or de-facto proprietary, hardware-security APIs provide such functionality. This is what motivated us to to start the On-board Credentials project.

4 On-board Credentials

On-board Credentials (ObC) is a TEE architecture that we developed at Nokia Research Center. ObC is now available in Nokia Windows Phone 8 and Symbian phones. The ObC architecture is illustrated in Figure 4 and further elaborated in [11].

In the ObC architecture, the needed isolation properties for trusted applications are achieved using the *ObC interpreter*, a small virtual machine (VM). Although this interpreter is not a complete operating system, it provides the necessary runtime environment for execution of trusted applications that originate from untrusted developers. The VM-based design was driven by the constraints of the current mobile devices such as the scarcity of isolated on-chip memory. Future generation mobile devices are likely to be less constrained in terms of TEE resources and may support secure virtual memory. Thus, adoption of TEE platforms that resemble full-fledged embedded operating systems may become practical in the future.

Development of ObC trusted applications can be done either in BASIC or using a bytecode assembler. The ObC VM is implemented as a set of interacting TrustZone code components — depending on the underlying TEE hardware and software architecture these components may permanently reside inside the TEE or be loaded on-demand. In some environments they constitute the only available "trusted OS" in the device whereas in other deployments the ObC interpreter components are themselves trusted applications; these are cases where ObC is used to augment the native trusted OS provisioning capability or isolation guarantees rather than to provide implementation minimality. In the rest of this section, we use the term trusted application to denote bytecode running on the ObC VM rather than the ObC VM implementation option described above.

The scheduling between the ObC TrustZone components, as well as managing the encrypted state of the currently interpreted trusted application, is left to the *ObC scheduler* that runs in REE. Applications in REE access the ObC system through the *ObC API* (see Figure 4).

4.1 Scheduling

To execute a trusted application, the scheduler loads the locally encrypted representation of it, as well as possible inputs and stored data sealed by previous invocations of the same application instance.

The ObC interpreter will then execute the trusted application bytecode. Some execution events including a bytecode that request an input parameter or a bytecode that needs to invoke a library function, will cause the interpreter to collect its runtime state (the virtual machine program counter, local variables and stack), encrypt it, and return to the REE for scheduling. The interpreter also performs on-demand code page loading of the bytecode itself which is another cause for scheduling out from the TEE. Back at the REE, the ObC scheduler will re-invoke the same or different trusted application, using the same or different TrustZone component, attaching possible temporarily stored data or interpreter state. In this manner the bytecode execution will continue until the trusted application completes.

This scheduling mechanism causes a significant execution overhead, due to often numerous context switches between TEE and REE, and corresponding encryption and decryption. However, since the ObC system runs on the main processor of the mobile device, the overall performance is comparable to solutions without such scheduling on slower security chips, such as smart cards.

4.2 Provisioning

The ObC platform supports an open provisioning model in which any developer can, with the permission of the user of a device, deploy trusted applications without having to ask permission from a centralized authority, such as the device manufacturer or the OS provider. A device-specific and manufacturer-certified public key provides the basis for remote provisioning of trusted applications (and the secrets they need to operate on) to devices already in the field. In addition, service providers need to handle user authentication for provisioning. The certified device (public) key is used to transport a provisioner-specific secret key that defines a security domain for trusted applications and secrets that are later provisioned to that domain. Isolation between security domains inside the TEE is guaranteed by interleaving execution of different security domains in time, and implementing local storage with distinct encryption keys for each domain.

4.3 Application development

To develop a complete service, a service provider needs to deploy two components to a mobile device: a trusted application that handles the service-specific security logic within the TEE, and an REE application that triggers the trusted application execution and potentially provides a user interface.

For trusted application developers, the ObC interpreter provides an API including cryptographic functions, standard string and array manipulation, sealing and I/O. Listing 1 shows an extract from a trusted application written in ObC BASIC for MirrorLink attestation (see next section).

For REE application developers, the ObC API includes functions for (1) provisioning of new trusted applications and secrets that these applications should operate on, (2) executing previously provisioned trusted applications with the needed input parameters, and (3) local encryption, or sealing, of REE application data that can be done without having to develop a custom trusted application.

Listing 1: A fragment of MirrorLink attestation trusted application. The fragment illustrates TPM platform configuration register (PCR) extending and signing (Quote operation).

```
rem --- Subroutines written in assembler or BASIC can be linked in
#include ``sha1_standard.evoh''
#include ``io_codes.evoh''
#include ``program_io.evoh''
...

  rem --- Extend operation
  rem --------------------------------
  if mode == MODE_EXTEND
    read_array (IO_SEALED_RW, 2, pcr_10)     rem --- Read earlier produced sealed input data
    read_array (IO_PLAIN_RW, 3, value)       rem --- Read plaintext input data

    append_array(pcr_10, value)              rem --- Concatenate arrays
    sha1(digest, pcr_10)                     rem --- PCRval' = H(PCRval | New)

    write_array (IO_PLAIN_RW, 1, digest)     rem --- Write plaintext output
    write_array (IO_SEALED_RW, 2, digest)    rem --- Write encrypted data for storage and future use
  end

  rem --- Quote operation
  rem --------------------------------
  if mode == MODE_QUOTE
    read_array (IO_SEALED_RW, 2, pcr_10)
    read_array (IO_PLAIN_RW, 3, ext_nonce)

    rem --- Create TPM_PCR_COMPOSITE (the default type is uint16)
    pcr_composite[0] = 0x0002               rem --- sizeOfSelect=2
    pcr_composite[1] = 0x0004               rem --- PCR 10 selected (00 04)
    pcr_composite[2] = 0x0000               rem --- PCR selection size 20 (uint32)
    pcr_composite[3] = 0x0014
    append_array(pcr_composite, pcr_10)
    sha1(composite_hash, pcr_composite)

    rem --- Create TPM_QUOTE_INFO
    quote_info [0] = 0x0101                 rem --- version (major and minor)
    quote_info [1] = 0x0000                 rem --- version (revMajor and revMinor)
    quote_info [2] = 0x5155                 rem --- fixed ('Q' and 'U')
    quote_info [3] = 0x4F54                 rem --- fixed ('O' and 'T')
    append_array( quote_info, composite_hash)
    append_array( quote_info, ext_nonce)
    write_array (IO_PLAIN_RW, 1, pcr_composite)

    rem --- Hash QUOTE_INFO for MirrorLink PA signing
    sha1(quote_hash, quote_info)
    write_array (IO_PLAIN_RW, 2, quote_hash)
    ...
```

4.4 Example Applications

In this section we describe two applications that we have developed and deployed using the On-board Credentials system.

Public transport ticketing. Our first application is public transport ticketing with NFC enabled mobile phones. We have designed and implemented trusted applications, and matching REE applications, for gated and non-gated ticketing scenarios. The protocol for our gated environment is a straight-forward challenge-response protocol for identity verification. In non-gated transport, the travel tickets are not verified at the station gates, but instead travelers are requested to perform ticketing on their own accord but are subject to random ticket inspections. In such a model, some evident options for fraud emerge. For example, a traveller could stop his phone from reporting evidence for trips during which he was never inspected.

To address this threat, we developed an ObC trusted application which implements an authenticated counter value that is bound to identity verification signatures. In ticket inspection and evidence reporting, we require that signatures are accounted for in backend logs and bind the use of the counter (device-local enforcement) to an operational window remotely controlled by the service backend to match the risk model and the usability considerations of the target public transport system [7].

A traditional cryptographic API (e.g., PKCS #11 or TPM interface) would not give the needed flexibility for the non-gated scenario. By having a programmable TEE environment, we were able to implement such an augmented cryptographic primitive with little effort and deploy it to devices already in the field. The non-gated ticketing application has been used in a trial of more than 100 users in New York transit system.

MirrorLink attestation. Our second application is MirrorLink (`www.mirrorlink.com`), a system that enables usage of smartphone provided content and services in automotive environments. In order to enforce driver distraction regulations, the car *head-unit* (the main console of the car entertainment and navigation system) must verify the trustworthiness of the data it receives from the mobile device. We have designed an attestation protocol that builds on standardized functionality and data structures defined in the Mobile Trusted Module (MTM) specification of TCG. A set of Platform Configuration Registers (PCRs) aggregate measurements that represent the mobile device software configuration. A signature using a device-manufacturer certified key over the content of such registers provides the verifier (i.e., the car head-unit) assurance on the trustworthiness of the software running on the mobile device. Content attestation is built on top of device and software attestation [10]. We have implemented the relevant subset of the MTM specification as a trusted application, and Listing 1 shows a fragment of this implementation.

This example illustrates a different use of programmable TEEs. There are many legacy security standards, ranging from one-time token algorithms to APIs like MTM that may originally have been intended for implementation with dedicated hardware tokens or resources. In devices with programmable TEEs, it becomes possible to easily deploy these interfaces to already shipping customer devices if the security properties of the TEE satisfies the compliance requirements of the standard being implemented. For example, the Nokia ObC implementation of the MirrorLink attestation has been security audited for approval by Car Connectivity Consortium. Evaluation with respect to more generic auditing systems, such as Common Criteria, is left for future.

5 Emerging standardization

To date, there is no standard interface that would define how REE applications access security services of a TEE. The ARM TrustZone architecture can be considered a de-facto starting point hardware architecture for mobile devices. NIST recently made public a draft guideline for hardware rooted security in mobile devices [4]. NIST identifies secure storage, (code) integrity, reporting and provisioning, policy enforcement and OS component measurement as security services that should be rooted in device hardware. From these roots, TEE isolation properties, application protected storage and overall integrity guarantees can be constructed.

The Global Platform consortium (`www.globalplatform.org`) intends to produce a full set of standards for TEE interfaces, including trusted application provisioning and use, API interfaces within the TEE and I/O specification for trusted application communication with external interfaces such as trustworthy user interfaces. Many of these specifications have already been in public review: An OS driver interface (TEE Client API) provides an interface to activate and run trusted applications, and a trusted application system interface (TEE internal API) provides commonly available programming interfaces such as memory allocation primitives and cryptographic primitives for trusted applications. Thus far, Global Platform has not published how REE applications access TEE functionality.

TPMs, specified by TCG, define security interfaces for non-volatile storage, key generation and use, and binding locally encrypted data to the measured software configuration of the device (sealing). The recently published TPM 2.0 specification [19] adds a fine-grained policy authorization model for object access. The TPM is typically implemented as a discrete chip, but in a mobile context a trusted application implementation is more likely deployment. TPM interfaces do not specify installation of trusted applications into TEE. TCG also has a mobile working group; a recently published use case document [18] indicates that TPM Mobile will provide standardization support for OS application interfaces as well as trusted application use and possibly provisioning. These specifications are complementary to the Global Platform standards.

6 Outlook

Security through isolation has a long history in computing systems. In mobile devices, hardware-based TEEs have been available for over a decade, but as we have seen, the ability for app developers to make use of them has been limited. Our ObC system was a first step in this direction.

Recent standardization efforts in Global Platform and Trusted Computing Group is accompanied by increased interest in the industry to make standardized TEE functionality widely available. However, as of today, no standardized API exists that would provide the TEE functionality exemplified by the ObC system, including open, trusted application provisioning and execution by REE applications. A recent white paper by Global Platform [9] indicated a vision for moving away from the centralized “issuer-centric” provisioning model to a more open “consumer-centric” model. The concrete provisioning specification is not yet public. Therefore it is too early to say what its level of openness would be.

In addition, several important considerations are likely to be missing from the picture. In many emerging mobile platforms, third-party application development is based on web programming techniques. How can web applications access TEE services? Current mobile devices are equipped with several security critical interfaces, including SIM cards, NFC payment and user interfaces. Can future trusted applications control and use such interfaces? If trusted applications are distributed from traditional REE application stores, what is the security model by which these trusted applications are authorized for the TEE within the destination device, and how is associated confidential data distributed to the device? Provisioning of confidential data implies encryption with a device-specific key which transforms a traditional application publishing and downloading process into an interactive provisioning protocol. How are secrets collected and amassed in a trusted application migrated to a new device when the user updates his device? How are backups, factory reset and device service issues handled? Can centralized trusted authorities, such as Trusted Service Managers (TSMs) [6] assist the user in such credential life-cycle management operations? If the device has many parallel personalities, like in *bring your own device* (BYOD) concepts, what is the authorization model to trusted applications in the respective usage modes?

As with any new problem domain, only when the first significant stakes are set, we see the full range of issues lying ahead. TEEs in mobile devices is now at this critical juncture. While some form of developer access to TEEs can be expected in the short term, a number of open issues need to be addressed before developer use of TEEs becomes widespread. When that happens, the ingenuity and creativity of developers will lead to novel ways in which they will make use of TEEs to improve their applications. In just the same way, we can expect attackers to find creative ways of exploiting TEE functionality to strengthen their attacks. Efforts in this direction [15] are already under way!

References

[1] 3GPP. *3GPP TS 42.009 Security Aspects*. 3GPP, March 2001. `http://www.3gpp.org/ftp/Specs/html-info/42009.htm`.

[2] Apple Inc. iOS security, October 2012. `http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf`.

[3] ARM. Building a Secure System using TrustZone©Technology. ARM Security Technologyg, April 2009. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html`.

[4] L. Chen, J. Franklin, and A. Regenscheid. Guidelines on Hardware-Rooted Security in Mobile Devices, Draft. Technical Report NIST SP 800-164, National Institute of Standards Technology, October 2012. `http://csrc.nist.gov/publications/drafts/800-164/sp800_164_draft.pdf`.

[5] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. *ACM SIGPLAN Notices*, 43(3):2–13, 2008.

[6] C. Cox. Trusted service manager: The key to accelerating mobile commerce, 2009. First Data white paper. `http://www.firstdata.com/downloads/thought-leadership/fd_mobiletsm_whitepaper.pdf`.

[7] J.-E. Ekberg and S. Tamrakar. Mass transit ticketing with NFC mobile phones. In *Proceedings of International Conference on Trusted Systems (TRUST)*, pages 48–65. Springer, 2012.

[8] N. Elenkov. Jelly bean hardware-backed credential storage, 2012. `http://nelenkov.blogspot.ch/2012/07/jelly-bean-hardware-backed-credential.html`.

[9] Global Platform. A New Model: The Consumer-Centric Model and How It Applies to the Mobile Ecosystem, March 2012. `http://www.globalplatform.org/documents/Consumer_Centric_Model_White_PaperMar2012.pdf`.

[10] K. Kostiainen, N. Asokan, and J.-E. Ekberg. Practical property-based attestation on mobile devices. In *Proceedings of Trust and Trustworthy Computing (TRUST)*, pages 78–92. Springer, June 2011.

[11] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 104–115. ACM, 2009.

[12] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, new, borrowed, blue – a perspective on the evolution of mobile platform security architectures. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 13–24. ACM, 2011.

[13] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for tcb minimization. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 315–328. ACM, 2008.

[14] Oracle. JSR 177: Security and trust services API for J2ME, 2007. Available at: `http://jcp.org/en/jsr/detail?id=177`.

[15] T. Roth. Next generation mobile rootkits. Presentation at BlackHat Europe 2013 `http://leveldown.de/bh_eu_2013.pdf`.

[16] RSA Laboratories. PKCS # 11 v2.20: Cryptographic Token Interface Standard, 2004. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf`.

[17] J. Srage and J. Azema. M-Shield mobile security technology, 2005. TI White paper. `http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf`.

[18] Trusted Computing Group. Mobile trusted module 2.0 use cases, March 2011. `https://www.trustedcomputinggroup.org/developers/mobile`.

[19] Trusted Computing Group. TPM 2.0 library specification, parts 1-4, Level 00, Rev. 00.96, March 2013. `http://www.trustedcomputinggroup.org/resources/tpm_library_specification`.