Practical Property-Based Attestation on Mobile Devices

Kari Kostiainen, N. Asokan, and Jan-Erik Ekberg

Nokia Research Center
{kari.ti.kostiainen,n.asokan,jan-erik.ekberg}@nokia.com

Abstract. We address property-based attestation in the context of an in-vehicle communication system called Terminal Mode that allows mobile devices to "stream" services, such as navigation or music, to car head-units. In Terminal Mode, attestation of the mobile device is needed to enforce driver distraction regulations and traditional binary attestation is not applicable due to frequently needed measurement updates and limited connectivity of car head-units. We present a novel attestation scheme that bootstraps from existing application certification infrastructures available on mobile device platforms, and thus avoids the need to setup and maintain a new service that provides translation from software measurements to properties, and consequently makes realization of property-based attestation economically feasible.

1 Introduction

Embedded computers on modern car head-units are capable of running various applications such as navigation services or digital music libraries. By utilizing integrated displays, speakers and user input controls on cars usage of such applications is convenient even while driving. Like any other software system such applications benefit from frequent software updates (e.g., bug fixes and new features) and new content (e.g., updated map data or new music). On-demand software updates and content downloads to car head-units may be difficult to realize since cars typically have limited Internet connectivity. On the other hand, mobile devices, such as phones, are well-connected and frequent software and content updates are a matter of course. Additionally, a much wider range of applications is typically available for mobile devices than for car head-units. Thus, a natural approach is to let a mobile device provide, or "stream," application content to the car head-unit and to use the displays, speakers and user input mechanisms integrated to the cars for optimal user experience. Terminal Mode [6] is a recent industry standard that specifies such an in-vehicle communication system.

A system like Terminal Mode raises obvious driving safety concerns. Studies have shown that driver distraction due to usage of in-vehicle information systems is a major contributing factor to traffic accidents [5]. To limit driver distraction, regulatory authorities and automotive industry bodies have issued various regional recommendations and requirements for in-vehicle information

systems [33]. While there is no single, globally accepted set of requirements that would be directly applicable to a system like Terminal Mode, representatives of the automotive industry required that the Terminal Mode system should address the following two issues: First, the car-head unit should present to the user only information that it has received from a device that has been authorized by the user. This requirement can be easily fulfilled by relying on secure pairing and first-connect mechanisms available on the most widely used short-range connections, e.g., Bluetooth Secure Simple Pairing [29] or WiFi Protected Setup [2]. Second, the car head-unit needs reliable means to learn the type of content streamed to it in order to enforce any applicable driver distraction regulation. For example, the head-unit might disallow video content while the vehicle is in motion. Thus, the head-unit needs to verify, or attest, the trustworthiness of the software component on the mobile device that provides such content type information.

During the past decade hardware-based "trusted execution environments" (TrEEs) have started to become widely available in commodity mobile devices. Many current mobile phones support integrated security architectures like M-Shield [30] and TrustZone [4] that augment the mobile device central processing unit with secure storage and isolated execution capabilities. The Trusted Computing Group (TCG) has specified a security element called the Trusted Platform Module (TPM) [32] that provides pre-defined hardware-based security operations. For mobile devices the equivalent is called Mobile Trusted Module (MTM) [13]. Whereas the TPM is always a separate security processor, the MTM interface can be implemented on top of a general-purpose TrEE, such as M-Shield or TrustZone.

The TCG standards include a mechanism called "remote attestation" [32] that allows an *attesting* device to prove the current (software) configuration of the attesting device to a remote *verifier*. In a typical remote attestation process, a trusted software component on the attesting device measures other software components that are executed on the same device and accumulates these measurements into so called Platform Configuration Registers (PCRs) of the TrEE. A signature calculated with a TrEE-resident key over the PCRs proves to the verifier the software configuration the attesting device is running.

Such "binary attestation" has well-known drawbacks. Modern computing devices consist of large number of software components that must be updated frequently. To be able to verify the trustworthiness of the attesting device the verifier must know the exact measurement of each trustworthy software component. For many use cases this assumption does not hold true. For example in the Terminal Mode system the mobile device might be manufactured after the car head-unit has been shipped, and the applications on the mobile device may be updated frequently, but deploying frequent software measurement updates to car-head units is not always practical due to the limited connectivity.

Another well-known drawback of binary attestation is loss of privacy. For example in Terminal Mode, the car-head unit only wants to guarantee that the content it receives has been assigned correct type information. Attesting exact

software measurements of the mobile device leaks more information about the attesting mobile device than is necessary to ensure driving safety. (Since head-units typically have limited connectivity, such information leakage may not be considered harmful in this particular use case.)

To address these drawbacks of binary attestation, several researchers have proposed attestation of *high-level properties* instead of exact measurements [25], [23], [7], [19], [8], [17], [16]. In "property-based attestation" a trusted authority defines and maintains mappings from approved software component measurements to more high-level properties, e.g., in the form of signed property certificates. The attesting device, or another device acting as a trusted attestation proxy, can report these properties instead of exact measurements to the verifier based on the matching property certificates.

Despite many research papers, property-based attestation has not seen wide-scale deployment. We feel that the primary reason for this is that there has not been a concrete use case in which deployment of property-based attestation is both highly needed and practical to realize. All property-based attestation schemes are based on the assumption that there exists a trusted authority that defines and maintains mappings from acceptable software measurements to properties. Without a concrete use case it is unclear what the exact requirements for a property-based attestation scheme are, what exact properties should be attested, and most importantly, which entity would have the required business incentive to set up and maintain such a translation service, especially since due to the large number of software components and frequent software updates on current devices running such a translation service is a non-trivial task.

Another drawback of existing property-based attestation schemes is that attesting an unlimited number of properties with a limited number of registers requires that multiple properties are accumulated into a single register. For successful attestation verification all accumulated properties must be reported to the verifier which leaks more information about the attesting device than is necessary.

Contributions. The primary contributions of this paper are two-fold: First, we have identified a concrete use case, the Terminal Mode system, in which property-based attestation is needed to enforce driver distraction regulation. Binary attestation is not applicable to Terminal Mode because of frequent software updates on mobile devices and the difficulty to issue updated measurements to car head-units due to their limited connectivity. Second, we present a novel property-based attestation scheme that bootstraps from existing mobile application certification infrastructures, and thus eliminates the need for setting up and maintaining a *new* trusted authority that defines translation from exact software measurements to properties, and consequently makes deployment of property-based attestation feasible. We describe two variants of our attestation scheme: one for TrEEs that support secure execution of arbitrary code and a TCG-compliant variant for TrEEs that are limited to MTM functionality. Our scheme provides attestation of mobile applications and we show how application-specific content attestation can be built on top. We have implemented both of

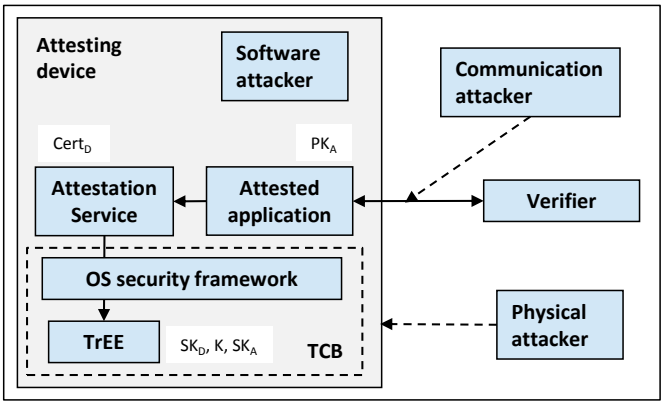

Fig. 1. Remote attestation scheme system model

these variants on mobile phones with M-Shield TrEE by taking advantage of the Symbian Signed application signing infrastructure and the platform security architecture available on Symbian OS devices. We also discuss how our attestation scheme can be implemented on other mobile platforms. The TCG-variant of our attestation scheme has been included as part of the recent Terminal Mode industry standard [6].

The secondary contribution of this paper is a simple register re-use technique that allows unlimited number of properties to be attested independently of each other with a single register, and thus preserves privacy of the attesting device, since only the property that the verifier is interested in must be reported to the verifier.

2 Assumptions and Objectives

Assumptions. The entities involved in the attestation process are shown in Figure 1. We assume that the attesting device has a hardware-based TrEE. The TrEE may either support secure execution of arbitrary manufacturer signed code or it may be limited to MTM functionality. The TrEE is equipped with a statistically unique asymmetric device key $\mathsf{SK_D}$. The public part of this key $\mathsf{PK_D}$ has been issued a certificate $\mathsf{Cert_D}$ by a trusted authority. The TrEE is also equipped with a symmetric key K that can be used for sealing (local authenticated encryption). For MTM TrEE, the asymmetric device key is called Attestation Identity Key (AIK) and the symmetric sealing key Storage Root Key (SRK).[1]

We assume that the attesting device has an operating system (OS) security framework with a permission-based security model in which access to system services can be limited either in terms of permission or application identifiers. The OS security framework enables the called system service to reliably determine the

[1] The TPM specifications define SRK as an asymmetric key. The MTM specifications allow implementation of SRK as a symmetric key, as well.

identity of the calling application. We assume that the application identities are assigned by a (central) trusted authority and verified with code signing during application installation. The OS security framework provides runtime isolation between applications and isolated storage for each application. We assume that the integrity of the OS security framework itself is protected with secure boot, i.e., the TrEE enforces that only manufacturer signed OS images are booted. For more information on security frameworks and secure boot on current mobile platforms see e.g. [18].

The attesting device has a trusted software component called Attestation Service. There may be any number of attested applications on the device. Each attested application has an application identity assigned by a central application signing infrastructure and its own application-specific key pair $\mathsf{SK_A}/\mathsf{PK_A}$. The private part of this key $\mathsf{SK_A}$ can be kept within the TrEE. We assume that the verifier device has a trust root for verifying the device certificate $\mathsf{Cert_D}$ of the attesting device.

Threat model. The attacker may have any of the following capabilities:

- "Software attacker" can install any applications on the attesting device and reset the device.
- "Physical attacker" can read and modify any data persistently stored on the file system of the attesting device when the attesting device is turned off.
- "Communication attacker" can control all communication between the attesting device and the remote verifier according to the typical Dolev-Yao model [9].

On the attesting device, the trusted computing base (TCB) consists of the TrEE and the OS security framework. The attacker cannot read or modify secrets stored or alter any computation that takes place within the TrEE of the attesting device. The attacker cannot compromise the OS security framework of the attesting device at runtime either.

Objectives. The attestation scheme should enable the verifier to verify that the application that it is communicating with on the attesting device complies with a chosen property. In the context of Terminal Mode, this is needed so that the verifier may trust content type information provided by the attested application. More precisely, the attestation scheme should fulfill the following objectives:

- *Attestation trustworthiness*: The verifier should only accept attestations of application properties reported by a trusted Attestation Service, i.e., the attacker should not be able to report false properties.
- *Attestation freshness*: The attacker must not be able to replay old attestations.
- *Attestation binding*: The attestation of an application should be securely bound to the subsequent communication between the attested application and the verifier, i.e., the attacker must not be able to masquerade as a previously attested application.
- *Attestation independence*: Attesting one application property should not disclose information about other properties of the same or another application on the attesting device.

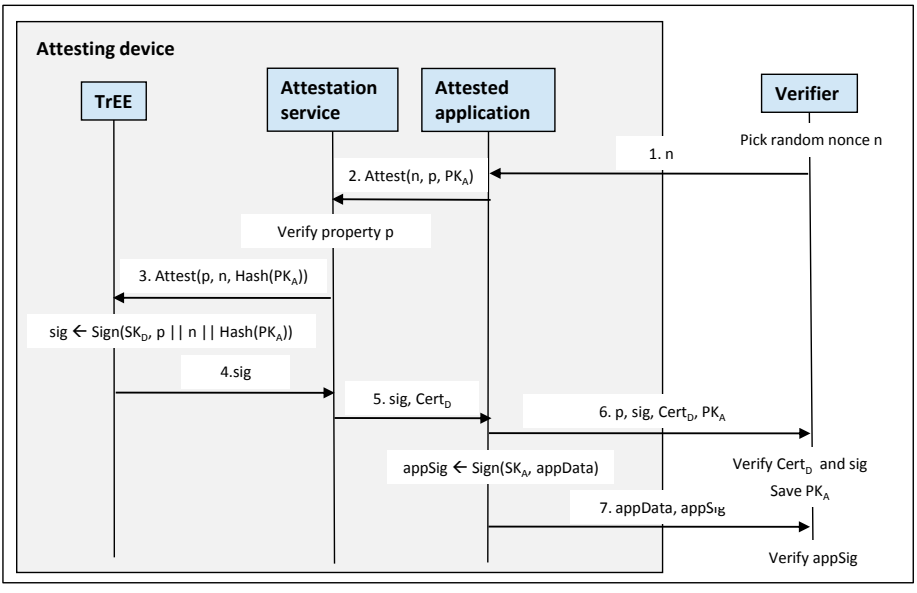

Fig. 2. Attestation protocol for TrEEs that support execution of arbitrary code

3 Attestation Scheme

Our attestation scheme bootstraps from existing application certification infrastructures. Instead of maintaining mappings from exact software measurements to properties, the trusted Attestation Service component on the attesting device maintains mappings from *application identifiers* issued by the application signing infrastructure to properties. The Attestation Service component determines the identity of the attested application using the underlying OS security framework. An application-specific key is appended to the reported property for attestation binding.

3.1 Attestation Protocol

Figure 2 shows the attestation protocol for TrEEs that support execution of arbitrary code.

1. The verifier picks a random nonce n, saves it for later verification, and sends the nonce to the attested application.
2. Each application may match multiple properties. The attested application picks the property p that should be attested and calls an attestation function $\mathsf{Attest()}$ from the Attestation Service with the nonce n, the property p and an application-specific public key $\mathsf{PK_A}$ as parameters.
3. The Attestation Service determines the identity of the calling attested application and checks if the requested property matches the application identity

in a translation table that contains list of properties for each application identity. If the property matches, the Attestation Service calls Attest() function from the TrEE with the property p, the nonce n and a hash of PK_A as parameters.
4. The TrEE signs a concatenation of the received parameters using the device private key SK_D. The resulting attestation signature sig is returned to the Attestation Service.
5. The Attestation Service returns the attestation signature sig and the device certificate $Cert_D$ to the attested application.
6. The attested application sends these parameters together with the property p and the application-specific public key PK_A to the verifier. The verifier checks that the device certificate $Cert_D$ validates with respect to a trust root available on the verifier. The verifier validates the attestation signature sig using the public key extracted from $Cert_D$ and verifies that the signature contains the expected nonce n, the received property p and a matching hash of PK_A. If this is true, the verifier accepts the attestation of property p and saves the application public key PK_A for later use.
7. The attested application may use the application-specific key to authenticate all subsequent communication with the verifier. One approach is to sign all sent application data appData with the private part of the application-specific key SK_A and append the resulting signature appSig to the message. Another alternative is to establish a client-authenticated TLS connection using the application-specific key.

3.2 Attestation Protocol for MTM TrEEs

Figure 3 shows the attestation protocol for MTM TrEEs. We assume that a fixed Platform Configuration Register (PCR) is reserved for application attestation on the TrEE of the attesting device. All attested properties are accumulated into that register, but instead of reporting all accumulated properties to the verifier, the PCR is extended with a random value for each attestation, and the random value is reposted together with the extended property to allow independent attestation of properties.

1. The verifier picks a random nonce n, saves it for later use, and sends the nonce n to the attested application.
2. The attested application picks the property p that should be attested and calls Attest() function from the Attestation Service with the nonce n, the property p and the application public key PK_A as the parameters. The Attestation Service checks the identity of the calling application and checks if the requested property matches the application identifier in its translation table.
3. The Attestation Service picks a random value r and extends the reserved PCR with r by calling TPM_Extend().
4. The TPM_Extend() command returns the current PCR value. The Attestation Service saves this value for later use as old.
5. The Attestation Service extends the same PCR by calling TPM_Extend() function. As a parameter, the Attestation Service uses a hash calculated

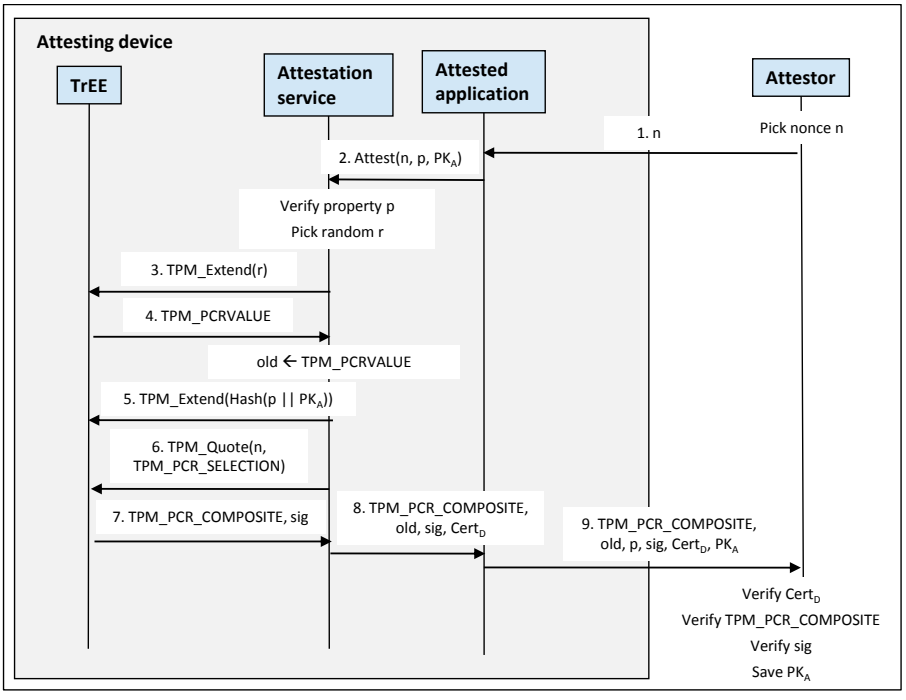

Fig. 3. Attestation protocol for MTM TrEEs

over the concatenation of property p and the public application key $\mathsf{PK_A}$. The return value is ignored.

6. The Attestation Service signs this extended PCR value by calling TPM_Quote with the nonce n as parameter. Additional TPM_PCR_SELECTION parameter is used to define that the signature should be calculated over the reserved PCR.
7. The TrEE constructs a TPM_PCR_COMPOSITE structure and signs it using the $\mathsf{SK_D}$ (AIK). The composite structure and the resulting signature sig are returned to the Attestation Service.
8. The Attestation Service returns these parameters together with the saved PCR value old and the device certificate $\mathsf{Cert_D}$ (AIK certificate) to the attested application.
9. The attested application sends these values, the property p, $\mathsf{PK_A}$ and $\mathsf{Cert_D}$ to the verifier. First, the verifier validates the device certificate $\mathsf{Cert_D}$. Then, the verifier checks that the received TPM_PCR_COMPOSITE structure matches a structure that would result when TPM_PCRVALUE old is extended with a hash calculated over p and $\mathsf{PK_A}$. After that, the verifier validates the attestation signature sig. If all these conditions hold true, the verifier accepts the attestation of property p and saves $\mathsf{PK_A}$ for authentication of subsequent communication with the attested application.

4 Implementation

We have implemented both variants of our attestation scheme for Nokia phones running Symbian OS with M-Shield TrEE in the context of the Terminal Mode system.

TrEE implementation. M-Shield is a security architecture for mobile devices in which the central processing unit of the device is augmented with secure storage and execution. In M-Shield, the pieces of code that are executed within the TrEE are called Protected Applications (PAs). We have implemented three PAs. Our first PA creates the needed device key, seals it for local storage and runs a device certification protocol according to the TCG Privacy CA principles [32] with a CA we have implemented. The second PA implements attestation signatures for our first protocol variant using the sealed device key.

The third PA implements a subset of MTM functionality needed for the MTM protocol variant. During the first invocation this PA creates an MTM state structure within the TrEE. The state structure is sealed for persistent storage on the OS side and a hash of the state is saved inside the TrEE in volatile secure memory. During subsequent invocations of the same PA, this hash is compared to a loaded state to prevent state modifications outside the TrEE. After each operation an updated state hash is stored inside the TrEE. The TPM_Extend() operation updates the state with the new PCR value. The TPM_Quote() operation unseals the device key and signs the chosen PCR selection with it using formats defined in the TPM and MTM specifications. We use the Symbian OS security framework to enforce that only trusted operating system components, i.e. components with manufacturer vendor identity, are allowed to use these PAs.

Symbian implementation. We have implemented the Attestation Service as a Symbian server component. The Attestation Service has a translation table, in the form of a text file, that contains list of properties for pre-defined Symbian application identifiers. This table is stored in so called "private directory" of the Attestation Service component—Symbian security framework provides an isolated storage mechanism called private directory for each application and system server. The Attestation Service determines the identity of the attested applications using the Symbian inter-process communication (IPC) framework that allows the called server process to determine the application identifier of the calling application. The Attestation Service component itself has manufacturer vendor identity that allows it to access our PAs.

Terminal Mode integration. A straight-forward design choice would have been to directly attest each application on the mobile device that "streams" data to the car head-unit. The drawback of such an approach is that changes would have been needed to many applications, since each application would have had to implement the attestation protocol. For Terminal Mode deployment we wanted to re-use existing applications as much as possible, and thus we chose a hierarchical attestation approach in which the mobile device attests the Terminal Mode system core components using our attestation scheme, and these core components in turn attest, or *label*, the content sent through them by different

applications. The content labeling must be integrity protected using keys bound during attestation.

The Terminal Mode system uses several widely used protocols for communication between mobile devices and car head-units. Universal Plug and Play (UPnP) [10] is used for service discovery and remote control of applications on the mobile device from the car head-unit. Virtual Network Computing (VNC) and Remote Framebuffer (RFB) protocol [24] are used for streaming application video and Real-time Transport Protocol (RTP) [27] is used for streaming application audio from the mobile device to the head-unit. In our implementation, the mobile device attests the in-device server processes that implement these protocols. Thus, the attested properties are names of Terminal Mode core components, such as TerminalMode:UPnP-Server and TerminalMode:VNC-Server. Each attested property is prefixed with a use case specific label to prevent collisions with attestations for use cases.

Each of these Terminal Mode core components maintains a list of known application identifiers and matching *application categories*, such as Navigation or Music. When an application sends data to the car head-unit, the mediating and already attested Terminal Mode core component labels the data accordingly. The identity of the active application is determined from IPC calls between applications and Terminal Mode core components.[2] In case of UPnP communication, the integrity of such content labeling is protected by signing the sent UPnP messages with the UPnP server specific key that was bound to the attestation of the UPnP server component. We use standard XML signatures. Transfer of frame buffers generates large amount of traffic and thus creating an asymmetric signature over each frame update causes an unacceptable performance overhead. Instead, the car head-unit picks a symmetric session key, encrypts that using the public key that was bound to the attestation of the VNC server, and sends the encrypted key to the VNC server which may then create symmetric signatures (HMAC-SHA-256) over frame buffer updates.

5 Analysis

Objective analysis. A noteworthy aspect of our attestation scheme is that the Attestation Service does not measure and report the entire OS, as is typically done in TCG-style attestation. Instead, we rely on availability of secure boot on the attesting device. The verification of the device certificate guarantees that the attested device originates from a trusted manufacturer and on such devices only trusted OS images are booted. Thus, it is sufficient to only measure and attest the application at hand.

Our first objective was attestation trustworthiness. The verifier should accept attestations of properties that have only been reported by a trusted Attestation

[2] Transfer of video content is an exception. The VNC server sends system frame buffer updates without communicating with the application directly. For content labeling, the the VNC server can determine the identity of currently active application with a Symbian OS system call that reports application currently holding the screen.

Service. In our scheme attestation signatures are made with a TrEE-based device key. The attacker does not have direct access to TrEE-resident secrets. We limit software-based access to these keys to software components that have manufacturer vendor identifier, and assume that other system components with the same identifier do not use the device key for signing arbitrary data that could be used as false attestations. The application identifiers used by the Attestation Service are provided by the OS security framework. For the integrity of this framework, we rely on secure boot.

The attestation trustworthiness also relies on the integrity of the Attestation Service itself, the integrity of the translation table maintained by it, and the integrity of the attested application code, i.e., the attacker must not be able to modify the attested application after successful attestation. The integrity of all of these is guaranteed by the OS level security framework against a software attacker. In our implementation platform, Symbian OS, the integrity of the translation table can be protected against offline attacks of a physical attacker by sealing it with a TrEE-resident key. The integrity of the Attestation Service itself can be enforced with TrEE-based secure boot.

Our second objective, attestation freshness, is easily fulfilled with the use of random nonce in both of the attestation protocols. To meet our third objective, attestation binding, we use a well-known technique to include a hash of application-specific public key to the measurement that is signed and reported (see e.g. [12,11] for more details on binding attestation to subsequent communication). The private part of the application-specific key must not leak to the attacker. Against software attackers it is sufficient to use the means provided by the OS security framework, i.e., in Symbian we can store the private key to the private directory of the application. To protect against physical attackers the key should be sealed in TrEE. If TrEE-based sealing is not available to applications, an alternative is to create an ephemeral application-specific key after each boot and discard it when the device is powered off.

In our implementation, we attest the Terminal Mode system core components that in turn attest or label application-specific data sent via them with matching application categories. Thus, the implementation must preserve integrity of the application category configuration files maintained by these Terminal Mode core components as well. Again, for software attackers we rely on application-specific storage provided by the OS security framework, for physical attackers TrEE-based sealing must be used.

Our last objective, attestation independence, is not an issue in our first protocol version. For the MTM protocol variant we apply simple PCR re-use technique. Instead of reporting all properties accumulated to the dedicated PCR, we extend the PCR with a random value and only report this random value and the property that has been extended to the PCR after that.

Deployment analysis. Traditional property-based attestation schemes are based on the assumption that there exists a trusted entity that maintains mappings from approved software measurements to properties. Maintaining such a translation service is a tedious task if the attested software components are

updated frequently and originate from various sources, as is the case with most application level software on mobile devices today. Our approach bootstraps from existing application certification infrastructures, and thus in our attestation scheme we only need to maintain a translation table from application identifiers to properties. When mobile applications are updated, the applications identifiers remain unchanged, and consequently no changes to the translation table are needed.

In the Terminal Mode system the translation table maintained by the Attestation Service component is constant, since the set of attested properties, the Terminal Mode core components, is fixed. The attested software components maintain their own translation tables that contain identifiers of known applications and matching application categories. These tables need to be updated whenever a new application is added to or removed from the list. Such updates are expected to be considerably less frequent compared to software updates to the applications themselves.

The Terminal Mode system is an industry standard that should be applicable to all major mobile device platforms. The MTM-based variant of our attestation protocol can be implemented in variety of mobile devices, including devices with M-Shield or TrustZone TrEE, MTM TrEE, or even purely software-based implementations relying on OS level security and device authentication are possible. Naturally, purely software based implementations are not resistant against physical attackers.

Our implementation is based on Symbian application signing infrastructure and OS security framework, but the same protocol can be implemented for other mobile platforms as well. For example the MSSF security framework [15] used in MeeGo devices provides similar IPC mechanism that allows the called process to determine the identity of the calling process (assigned by a trusted authority). Also Windows Phone OS and iOS platforms are based on centralized application signing infrastructures. Although the internals of these platforms are not public, most likely similar attestation could be implemented on those platforms as well. Android platform is not based on centralized application signing infrastructure, instead most applications are self-signed. On Android devices, the Attestation Service would need to maintain mappings from signing key and application identity pairs to properties. The Attestation Service itself could be implemented as part of the virtual machine that runs the Android applications (similar to the designs reported in in [14] and [22]).

6 Related Work

Binary attestation. Using a trusted software entity to measure other, untrusted software components is by no means a new invention. The concept of software measurement based secure boot was introduced in the early 90's [20] and concrete bootstrapping architectures were built later in the same decade (see e.g. [3]). The Trusted Computing Group (TCG), and its predecessor the Trusted Computing Platform Alliance (TCPA), standardized the notion of software measurement based authenticated boot and binary attestation [32]. While

the basic TCG-attestation is limited to measuring the booted operating system components, architectures exist for measuring loaded applications during system runtime as well [26], and binding application produced data to the attestation of applications [28]. The current PC processors allow security-critical parts of applications to be measured, executed in isolation, and attested to a remote verifier [21].

Property-based attestation. Attesting high-level properties of native software components based on TPM technology was first proposed in [25]. Based on this concept researchers have proposed various property-based attestation schemes for TPM-based platforms ranging from provably secure protocols [7] to schemes that do not require a trusted third party (the attesting device and the remote verifier agree on the attested properties directly) [8], utilize a trusted proxy [23], or bundle multiple attestation requests into a single signature operation for better scalability [31]. Implementation of an operating system boot loader that is enhanced with a translation table that provides mappings from measurements to properties is described in [19] and further developed in [17].

In the context of managed code running on virtual machines, property-based attestation, or "semantic attestation," was first proposed in [14]. In their approach a trusted virtual machine determines properties of applications based on byte code inspection and application runtime monitoring. Attestation based on static analysis and runtime monitoring has been proposed for native code as well [16]. Similar concepts are discussed also in [1].

The Mobile Trusted Module (MTM) specifications introduce a concept of Reference Integrity Metric (RIM) certificates [13]. A RIM certificate contains a software measurement reference value and a matching identifier, e.g. a property, that should be extended to an MTM register when such a software component is loaded. Thus, RIM certificates can be used to implement property-based attestation on mobile devices.

Attestation on mobile devices. Recently binary attestation has been implemented for Android devices [22]. In Android, applications run on top of Java-based Dalvik virtual machine (VM). Traditional TCG-style measurements are used to attest the trustworthiness of the VM which in turn measures either entire Android applications or individual Java classes that are loaded. The measurements are accumulated into a single PCR of a software-based minimal TPM implementation that is included to the kernel.

The primary difference between all of the above mentioned attestation approaches and our scheme is that we bootstrap from existing application certification infrastructures, and thus with our scheme it is sufficient to maintain a translation from application identities to properties. There are also noticeable difference in the amount of code that is attested. While in most the TPM-based approaches, e.g. [25,19,17], the entire OS is measured and attested, in our approach the integrity of the OS is guaranteed with secure boot, and thus it is sufficient to measure and attest only the application at hand. In that sense, our approach bears similarities to [14] in which a trusted virtual machine attest byte code applications.

7 Summary

In this paper we have presented a novel attestation scheme that bootstraps from existing application certification infrastructures, and thus makes deployment of property-based attestation practical to realize. This attestation mechanism has been included to the recent Terminal Mode industry standard and the deployment is currently on-going to mobile devices and car head-units from multiple manufacturers.

We acknowledge Jörg Brakensiek and Matthias Benesch for intoruducing us to the problem and for their feedback during the design.

References

1. Alam, M., et al.: Model-based behavioral attestation. In: Proc. 13th ACM Symposium on Access Control Models and Technologies (2008)
2. WiFi Alliance. WiFi protected setup specification v1.0 (2007)
3. Arbaugh, W., et al.: A secure and reliable bootstrap architecture. In: Proc. IEEE Symposium on Security and Privacy (1997)
4. ARM. Trustzone-enabled processor, `http://www.arm.com/products/processors/technologies/trustzone.php`
5. National Highway Safety Traffic Association. The impact of driver inattention on near-crash/crash risk: An analysis using the 100-car naturalistic driving study data (2006), `http://www.nhtsa.gov/DOT/NHTSA/NRD/Multimedia/PDFs/Crash%20Avoidance/2006/DriverInattention.pdf`
6. Brakensiek, J.: Terminal mode technical architecture (2010), `http://www.nokia.com/terminalmode`
7. Chen, L., et al.: A protocol for property-based attestation. In: Proc. First ACM Workshop on Scalable Trusted Computing (2006)
8. Chen, L., et al.: Property-based attestation without a trusted third party. In: Proc. 11th International Conference on Information Security (2008)
9. Dolev, D., Yao, A.: On the security of public key protocols. Technical report. Stanford University (1981)
10. UPnP Forum, `http://upnp.org/sdcps-and-certification/standards/`
11. Gasmi, Y., et al.: Beyond secure channels. In: Proc. 2nd ACM Workshop on Scalable Trusted (2007)
12. Goldman, K., et al.: Linking remote attestation to secure tunnel endpoints. In: Proc. 1st ACM Workshop on Scalable Trusted Computing (2006)
13. Trusted Computing Group. Mobile trusted module specification, version 1.0 (2008)
14. Haldar, V., et al.: Semantic remote attestation - virtual machine directed approach to trusted computing. In: Virtual Machine Research and Technology Symposium (2004)
15. Kasatkin, D.: Mobile simplified security framework. In: Proc. 12th Linux Symposium (2010)
16. Kil, C., et al.: Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In: Proc. International Conference on Dependable Systems and Networks (2009)
17. Korthaus, R., et al.: A practical property-based bootstrap architecture. In: Proc. 4th ACM Workshop on Scalable Trusted Computing (2009)

18. Kostiainen, K., et al.: Old, new, borrowed, blue: A perspective on the evolution of platform security architectures. In: Proc. 1st ACM Conference on Data and Application Security and Privacy (2011)
19. Kühn, U., et al.: Realizing property-based attestation and sealing with commonly available hard- and software. In: Proc. 2nd ACM Workshop on Scalable Trusted Computing (2007)
20. Lampson, B., et al.: Authentication in distributed systems: theory and practice. In: Proc. 13th ACM Symposium on Operating Systems Principles (1991)
21. McCune, J., et al.: Minimal TCB Code Execution (Extended Abstract). In: Proc. IEEE Symposium on Security and Privacy (2007)
22. Nauman, M., et al.: Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. In: Proc. International Conference on Trust and Trustworthy Computing (2010)
23. Poritz, J., et al.: Property attestation scalable and privacy-friendly security assessment of peer computers. Technical Report RZ3548, IBM Research (2004)
24. Richardson, T.: The rfb protocol (2010), `http://www.realvnc.com/docs/rfbproto.pdf`
25. Sadeghi, A.-R., Stüble, C.: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: Proc. Workshop on New Security Paradigms (2004)
26. Sailer, R., et al.: Design and implementation of a tcg-based integrity measurement architecture. In: Proc. 13th USENIX Security Symposium (2004)
27. Schulzrinne, H., et al.: RTP: A transport protocol for real-time applications (2003)
28. Shi, E., et al.: Bind: A fine-grained attestation service for secure distributed systems. In: Proc. IEEE Symposium on Security and Privacy (2005)
29. Bluetooth SIG. Bluetooth specification version 2.1 + edr (2007)
30. Srage, J., Azema, J.: M-Shield mobile security technology (2005), TI White paper, `http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf`
31. Stumpf, F., et al.: Improving the scalability of platform attestation. In: Proc. 3rd ACM Workshop on Scalable Trusted Computing (2008)
32. Trusted Platform Module (TPM) Specifications, `https://www.trustedcomputinggroup.org/specs/TPM/`
33. International Telecommunications Union. Decreasing driver distraction, itu-t technology watch report (August 2010), `http://www.itu.int/dms_pub/itu-t/oth/23/01/T230100000F0001PDFE.pdf`