An Alternate Secure Element Access Control for NFC Enabled Android Smartphones

Waqar Anwar, Dale Lindskog, Pavol Zavarsky, Ron Ruhl
Concordia University College of Alberta

Abstract

For mobile payments using Near Field Communication (NFC), a Secure Element (SE) is the preferred place to securely store cardholder data. This paper summarizes shortcomings in Global Platform's (GP) SE access control specifications and weaknesses in its implementation by the Android Operating System (OS). Moreover, a coherent model for an alternate and secure SE access control is proposed using SE and Mobile Trusted Module (MTM) specifications. This new model is secured by design and can be implemented using existing specifications, technologies and hardware.

1. Introduction

Android is a smartphone OS, developed by the Google Inc. from Linux source and highly customized for the resource constrained mobile phone environment. Google Inc. relies heavily on Android for its NFC mobile payment solution, Google Wallet, which makes use of an SE embedded in the NFC chip to store card holder data. The SE is a highly secure and tamper resistant execution environment. The SE execution environment is based upon Java and is capable of securely executing Java Virtual Machines (VM) [1]. Each VM can store and execute a single physical smart card. The NFC chip is connected to the SE and can emulate a smart card over its inductive wireless link. A NFC reader device can read this information and process the card information, similar to when a card is presented. NFC operates within a very short distance, usually about one centimeter. Moreover, NFC radio in an Android mobile phone is only enabled when its screen is activated [16]. These design considerations prevent remote leakage of card holder information over the NFC wireless interface. There is, however, a relay attack that may result in leakage over the NFC internal interface [7]. The SE is also connected internally to the Android OS. The SE needs to communicate with Android applications (apps), such as Google Wallet, for user interaction, card selection and updating. In response to the relay attack described in [7], Google Inc. disabled its payment SE applications to process payments to and from the interface connected to the Android OS. Instead, Google SE applications now communicate only payment related information to and from the NFC radio interface. This workaround is possible because of SE applications' ability to detect the communication interface.

However, this countermeasure limits the system's design and operation, and Google SE applications can no longer be used for online browser based payments because the Internet browser runs on top of the Android OS, which used an internal Android to NFC chip interface to communicate with the SE [20]. Any Android app can access the SE provided it is whitelisted. This whitelist is stored inside the Android system partition, and only the Android OS or its modules can access or modify it.

GP SE specifications are based upon smartcard specifications and its secure operation is guaranteed by strict implementation of these specifications for its architecture, functions and access control. It is very important that SE access is granted only to an app able to authenticate itself to the SE. A GP compliant SE will transition to a non-reversible TERMINATE state if an app fails to authenticate itself within ten successive attempts. This fail-safe characteristic of the SE can be manipulated by a rogue app, in order to launch a denial of service (DoS) attack. Since an embedded SE is embedded in the NFC chip on the mobile phone mainboard, the SE is rendered useless after transitioning to the TERMINATE state, and so a user will be unable to make mobile payments unless they procure a new mobile phone, which is obviously costly and time consuming. As NFC mobile payments become more prevalent, this type of DoS attack may occur on a wider scale, and is an unacceptable risk to the economic system of a large population or country.

To address this problem, GP specifies three different kinds of SE access control [1]. Each control makes use of a whitelist stored inside the SE. The SE provides secure storage, and so the whitelist cannot be tampered with or changed by the Android OS. The Android OS can only fetch and read from this whitelist stored inside the SE, which can only be modified by one having access to the security domain of that SE, i.e., the SE owner or its delegate.

There is an important assumption underlying this approach to SE access control. Since the mobile OS is ultimately responsible for fetching this whitelist from the SE and implementing access control (using the so-called Access Control Enforcer system module), both Android's implementation, as well as GP's specification relies on the OS for SE access enforcement, and thus both ultimately trust the OS. This is not a realistic assumption, especially in the case of Android, since Android has been shown to be rooted by malicious code.

On a rooted Android device, the underlying kernel, including any access enforcement module can be manipulated, and thus so can the whitelist for SE access control. The SE will be unaware of any such tampering.

In addition to relay and DoS attacks, other types of privacy threats, such as leakage of NFC broadcasts, pose significant security risks to the adaptation of NFC mobile payment on a wider scale. For these reasons, we suggest that it is time to rethink current SE access control implementations and specifications, and address this problem in such a way that there is a significant level of assurance with NFC mobile payment technology.

Mobile trusted computing provides this much needed level of assurance. The Trusted Computing Group (TCG) has published Mobile Trusted Module (MTM) specifications, version 1 [8]. Use cases for version 2 are published [8] and MTM specifications version 2 may be published in the near future. TCG does not specify how a MTM is to be implemented, but it can be implemented either in software or in specialized hardware.

Android devices are based upon the ARM architecture, which is capable of a fairly seamless adaptation of trusted computing, since it has, since 2003, provided the needed execution environment [11]. This execution environment is called Trusted Execution Environment (TrEE) and is separated from the normal, so-called Rich Execution Environment (REE). MTM has been demonstrated to be implementable in TrEE.

Trusted computing based on MTM [11] can ensure that an Android OS is tamper resistant, but its use in a mobile environment poses some serious limitations. These limitations can render a mobile device unusable, sacrificing availability to integrity. This paper proposes a solution for SE access control that not only ensures integrity but also ensures confidentiality and availability for NFC mobile payment on Android devices.

The proposed solution is designed specifically for Android, but we believe the design is an improvement over current GP SE access control specifications. In the following sections, we first describe Android's various implementations and GP's specifications and their weaknesses, and then explore various hardware and software primitives. Subsequently, we describe our alternate model and the changes to those specifications that are needed in order to implement our proposed solution.

2. Existing approaches to SE Access Control

The Android development framework specifies that each application or executable module must be accompanied by a certificate, and the hash of that certificate must be signed by its developer [17]. Each module runs within a separate Java Dalvik VM. The architecture limits execution to that VM and enforces app isolation. However, modules signed by the same developer can share data and communicate with each other. As certificate hashes are self-signed, Public Key Infrastructure (PKI) is not used to verify the authenticity of its source, but this method ensures that further updates to that specific module come from the original source.

Since Android version 2.3.4, Google Inc. includes an NFC Application Program Interface (API) and, initially, access to NFC functions was limited to system modules only. Since Android version 4.0.4, Google Inc. has implemented a more flexible method, using a whitelist written to an XML file (NFCEE_ACCESS.XML), which stores the self-signed hashes of those third party Android apps permitted to access the SE [2]. Google Inc. retains the exclusive right to grant SE access to a third party Android app: the whitelist is stored inside Android's system partition, and only those system modules signed by Google Inc. can access and modify that whitelist. This whitelist can be updated, over-the-air, using system software provisioning. This solution is scalable, and third party application developers receive access to the SE

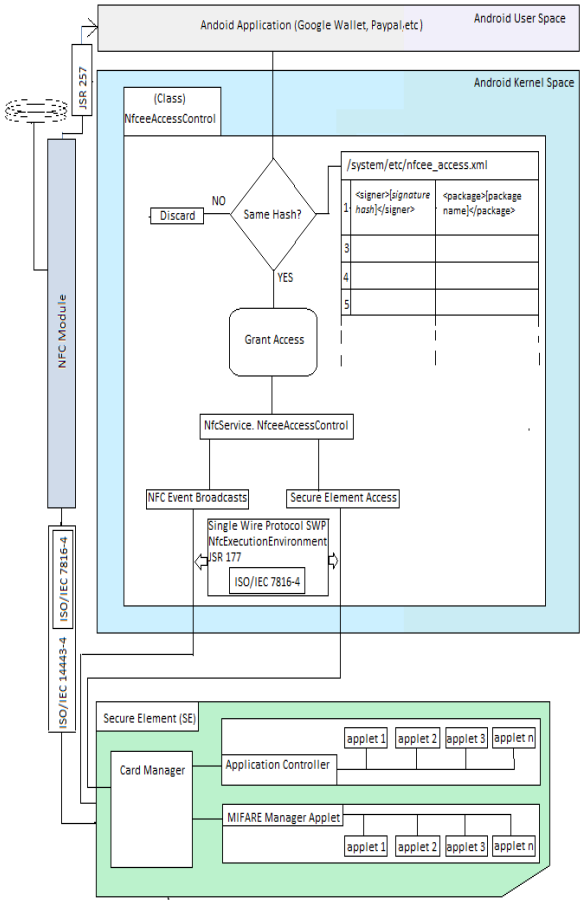

Figure 1: Android SE Access Control

when their application's certificate hash is added to the whitelist. Figure 1 depicts this method of access control. These two methods of access control so far described are secure only in so far as the Android OS is secure [2].

GP specifies SE architecture and functions; it also specifies SE access control. In GP's architecture, the operating system fetches the access rule stored inside the SE. Figure 2 shows one of the rule fetching scenarios by a mobile operating system using the SE. Access Control Enforcer fetches the rules from Access Rule Application Master ARA-M, verifies device application's certificate and grant or deny access to SE using transport layer. GE further mandates the use of a whitelist file that is stored in a PKCS#15 based file system inside the SE, and is fetched by the access control logic of a mobile operating system [1]. These rules can be cached by the OS. These specifications ensure that a mobile OS has only a read-only copy of the set of rules, and that it is not allowed to update or modify this access control data: the data is stored inside the SE and hence is modifiable only by a trusted party with access to the SE issuer's or SE application's security domains. GP specifies the use of SE as a secure storage and all the access control functionality is the responsibility of the mobile device OS. This access control methodology is as secure as the device OS is.

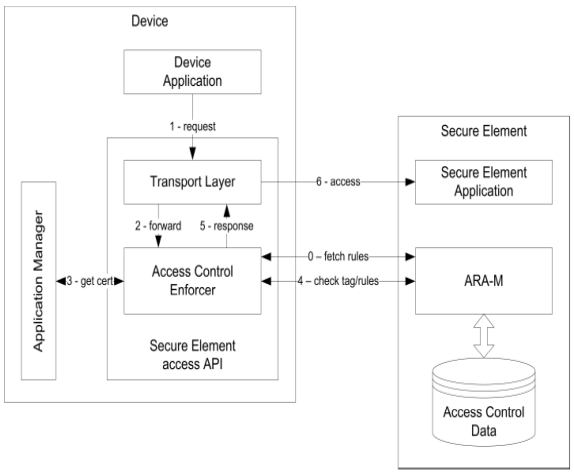

Figure 2: GP SE Access Control Specification [1]

All of the methods of SE access control reviewed so far have weaknesses. The Android operating system is an open platform which allows for dynamically installing, upgrading and removing of Android applications. A mobile operating system is neither a trusted nor a fully controlled environment. Tampering with a whitelist is relatively easy after an Android phone is rooted. Moreover, Android uses a static permission based model [4] to grant access to its various hardware resources, such as its camera, SMS functionality, MIC, Internet access, etc. Once a rogue application gets access to the SE and these other hardware resources, it can listen to NFC event broadcasts, connect to the Internet, send SMS messages for any event, or join a botnet for harvesting a user's private information.

Android application developers are encouraged to have their certificate self-signed [5]. This practice helps grow the android market, as developers need not have their application certificates signed by a globally trusted authority. This implies that, if a rogue application successfully exploits an android phone, there is no way to revoke its certificate and stop it from being installed on other android phones. Though Google Inc. or a Mobile Network Operator (MNO) could remotely terminate a specific instance of an application, this will not block its spread, and it is not clear how efficient this remote termination process is.

Once an application has added its certificate signature to the XML file, it can communicate with the SE and listen to NFC broadcasts. Although this application cannot communicate with the Card Manager unless it has the required keys, it can still perform a DoS attack by repeatedly attempting to authenticate to the Card Manager [2]; after the 10th unsuccessful attempt, a Global Platform compliant card goes into an irreversible TERMINATED state. As the Card Manager is the interface between SE applications and the outside environment, transition to this TERMINATED state has the result that the SE cannot communicate with the outside environment and is essentially bricked. Thus the mobile phone cannot perform mobile payments and the user must change the mobile phone. [2]. A similar type of DoS attack can also be performed against individual SE applications. The Card Manager will change the state of an SE application to TERMINATED if an Android application from outside the SE fails to authenticate to that particular SE application. This second type of DoS attack will not brick the whole SE, but it will terminate that particular SE application [2]. This means that the terminated SE application must be installed again in order to be used.

Another concern with this access model is remote execution of SE commands, where one may use the SE from another phone for mobile payment over the Internet. Relay attacks have been described in [7], and [6] demonstrates how NFC Event broadcasts can be sniffed by any third party application. A rogue Android app can use these broadcasts as triggers, since these broadcasts provide indications about the background processing of the NFC hardware. It also poses privacy concerns, since harvesting this information on a large scale can reveal the buying habits, type of card used, and location (using phone GPS data) at which a particular card is used. Such information may then be transferred and sold for marketing purposes without the user's knowledge.

We believe that the forgoing considerations demonstrate that SE access control implementations and specifications have failed to properly secure SE access. In subsequent sections of this paper we propose that SE access control rely on trusted computing principles and

specifications, and propose moving the enforcement of SE access control from the OS to the SE itself.

3. Redesigning SE Access Control

It is clear by now that a smartphone operating system such as Android cannot be trusted to enforce access control to its SE. Use of keyed authentication, such as a Personal Identification Number (PIN), is not robust in a smartphone as it can be sniffed by any rogue application installed on that smartphone, which can then use that PIN to automatically authenticate itself to the SE and perform any operation a valid user can perform. Similarly, caching or storing the authentication PIN hash inside the application would make it vulnerable to all kinds of attacks designed to recover and reuse that hash. Restricting an application's communication with the SE to payment specific commands is also not ideal, since this limits the true potential of the SE for mobile payment. Our proposal is based upon trusted computing. Only the trusted modules are granted access to SE. Careful reliance on trusted computing principles should ensure the following:

- The Android OS and its applications should be trustable, not only at start-up, but at run time also. If an Android OS is compromised, then the underlying file system (and thus, e.g., the whitelist described in the previous section) is also compromised. Securing the Android OS and its applications automatically prevents known NFC specific relay, replay and DoS attacks.
- An Android application's access to the SE should be enforced by the SE, rather than by the OS. This ensures independent decision making and reduces the chances of relay attacks by a remote application acting as a local android application. Only authorized applications should communicate with the SE. These objectives are met only if access control is implemented independent of the mobile operating system. On mobile devices, there is need for an underlying, independent, tamper resistant and security enhanced module. One such module is the SE itself. Other candidates include MTM and TrEE. Let us now turn to these components, to see whether and how their functionality might be used or modified to achieve our objective.

A. MTM

To achieve our objective of mobile trusted computing, we propose the use of Mobile Trusted Modules (MTM) as specified by the Trusted Computing Group (TCG), and its derivations as described in [8]. MTM is a Trusted Platform Module (TPM) implementation for mobile devices that discards many traditional functions of a standard TPM and introduces new functions tailored to the mobile environment. There are two types of MTM: Mobile Remote-owner Trusted Module (MRTM) and Mobile Local-owner Trusted Module (MLTM). MRTM mandates the use of additional security functions necessary to communicate with the remote owner of the module. MRTM uses a subset of the TPM v1.2 specification, making use of Root-of-Trust-for-Storage (RTS) and Root-of-Trust-for-Reporting (RTR). [8]

Figure 3 shows the building blocks of a typical MRTM. It requires at least two additional components, Root-of-Trust-for-Verification (RTV) and Root-of-Trust-for Measurement and Measurement and Verification Agent (MVA). We will refer to these two as 'RTV+RTM'. RTV+RTM must be executed, in order, before anything else can execute in the mobile environment. RTV+RTM verifies and registers its own hash inside the MRTM, using a function named MTM_VerifyRIMCertAndExtend, as shown in Figure 3. RTV+RTM then measures and loads MVA, and registers its measurement inside the MRTM using the same function. MRTM then verifies the measurements presented by RTV+RTM, both of itself and of the MVA module, and aborts the boot sequence unless these measurements match the already provisioned measurements stored inside MRTM.

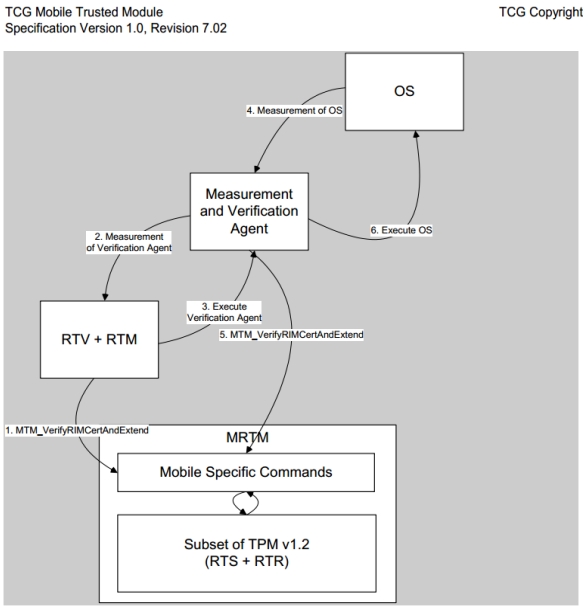

Figure 3: TCG Mobile Trusted Module [8]

Mobile trusted computing uses two forms of boot sequence: a secure boot sequence and an authenticated boot sequence:

During a secure boot sequence each module measures the next module before executing it [10]. This measurement involves computing the hash of the next module image. The measuring module then forwards the Reference Identity Metric (RIM) of the measured module

to the MRTM. RIM is specified by TCG, and is essentially a certificate containing the signed hash of a module. The MRTM is used for this certificate verification. If the MRTM trusts the presented certificate RIM, the MVA verifies the measurement it made. If the MRTM does not trust the presented RIM, the boot process is aborted. Measurements are registered in the Platform Configuration Registers (PCR) by the measuring module at each step.

There are two options for validating an Android application's RIM certificate. One option is for its RIM certificate to be signed by a party whose public key is signed by the Root Verification Authority Identifier (RAVI) key. RAVI is the top level key generated inside a TPM for RIM verification and certification. A second option is for an Android application's RIM to be certified by the TPM using RAVI. For Android using NFC, the first option is more practical, as a Trusted Security Manager (TSM) can become a RIM certification authority by having its public key signed, at provisioning time, by the RAVI of its TPM.

Unfortunately, a secure boot sequence is inconsistent with the traditional practice of self-signing Android applications. For the authenticated boot sequence, measurements are made and the next module is executed [10]. There is however no verification process, but rather, measurements are registered in the PCR at each step. Because most modules in Android use self-signed certificates, an authenticated boot sequence is the preferred choice for our design. Moreover, measurements registered in the PCR can be used to attest to the system state, for authentication and access control purposes. The MLTM provides these necessary functions. MLTM uses a shared secret to authenticate a local user, and the SE can act as a local user in this scenario.

As no other executable is loaded before RTV+RTM and MVA, a low level Application Programing Interface (API) can be used to communicate with the MLTM. A high level API similar to a system service can be used once the OS is loaded [10].

B. ARM TrustZone

The majority of mobile devices are based upon the ARM architecture. ARM provides a Trust Zone Security Extension [12] that enables a single Central Processing Unit (CPU) to execute code in parallel without affecting or mixing with each other. The ARM Architecture defines two parallel environments or 'zones', a 'secure zone' and a 'normal zone'. This secure zoning technology is called TrustZone. TrustZone is briefly described in [11].

The TrEE environment has been further standardized in GP's Trusted Execution Environment (TEE) [1] as shown in Figure 4. TEE is independent of REE and TEE can have more control over REE by having full access to its shared memory space. GP specifies TEE as an intermediary environment between a normal or Rich Execution Environment (REE) [1] and the SE. TEE can be implemented using System-on-a-Chip (SoC) technology, such as TrustZone. However, TEE is not as physically secure and tamper resistant as the SE. The fact that TrEE is independent of the mobile OS is particularly useful for our purpose, as MTM or mobile TPM can be implemented in software in TrEE, and TrEE can host code for the TPM's low level API to communicate with the SE. This communication is independent of the mobile OS and hence is highly resistant to any type of outside attack.

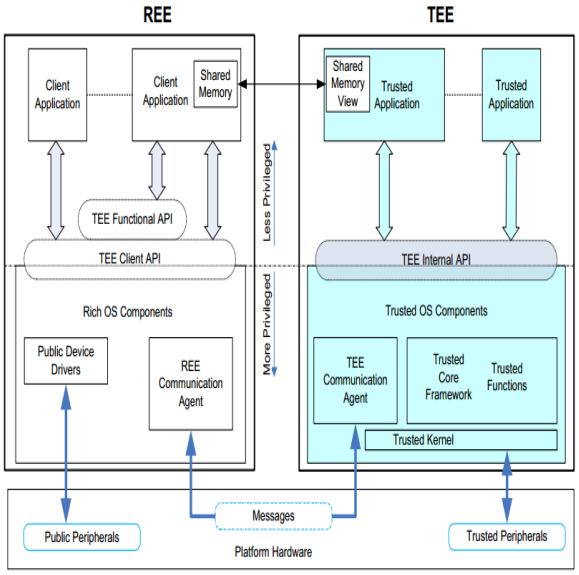

Figure 4: GP TEE Architecture [1]

In TrustZone TrEE, a secure zone module can access the resources in REE, but a REE module cannot access TrEE resources [11]. If an SE access control module is implemented in TrEE, this module can proxy APDU communication by reading and writing to REE memory, but a REE module such as an Android payment application cannot directly communicate with this proxy module or the SE.

Another consideration is the use of system peripherals. In ARM, the 'Corelink' system bus interconnects the Central Processing Unit (CPU) and memory using the Advanced eXtensible Interface (AXI). The rest of the system peripherals interconnect using the Advanced Peripheral Bus (APB). AXI is capable of distinguishing TrEE and REE transactions, but APB does not. In this case, the AXI-to-APB bridge is responsible for managing security relevant states. AXI-to-APB logic selects the desired peripheral based on the incoming AXI transaction. The bridge is responsible for rejecting REE transactions to the peripherals designated to be used by TrEE [11]. The AXI-to-APB Bridge is programmable, and can dynamically switch the security state of a given

peripheral. This property is especially useful for access control. A number of studies have either described theoretically, or report having actually implemented software based on TPM in TrEE. See [11], [12], [13] and [15]. Practical integration of the MTM with the Android platform is described in [8].

C. Secure Element

The secure element is a highly trusted and tamper resistant execution environment. It is based on smart card technologies. Java and MultOS are two popular operating systems for this execution environment. GP has published detailed specifications for smart cards and SE implementation [1]. The SE execution environment consists of various types of Virtual Machines (VM). These VMs are essentially executable modules containing applications and data. Each SE app's VM is associated with but firewalled to an executable VM called Security Domain (SD). This association and isolation is guaranteed by the card execution environment. The SD is responsible for securely storing keys and for cryptographic operations. Upon initializing a SE, the first VM installed is the Card Manager, and the first security domain created is the Issuer security domain. Also, the Global Platform execution environment (OPEN) is created. OPEN is responsible for the secure architecture of SE, and implements application isolation and API functions between SE applications and the Card Manager. The Card Manager sends and receives APDUs to SE applications using the OPEN API, and vice versa. The Card Manager has global access to the SE and all other SE applications and security domains. The Card Manager acts as an interface between installed SE applications and the outside world. It also acts as a proxy for other SE applications and forwards all APDUs to the relevant SE application, unless the APDUs are directed to itself. The Card Manager also provides card holder verification services, essentially a PIN verification service.

Card Manager is the card's representative. Figure 5 shows several of its states. Our proposed design ensures that an irreversible TERMINATED state does not occur as a result of using the card in a smartphone environment. Instead, we propose a reversible Card_Locked state for the card and other smartphone primitives.

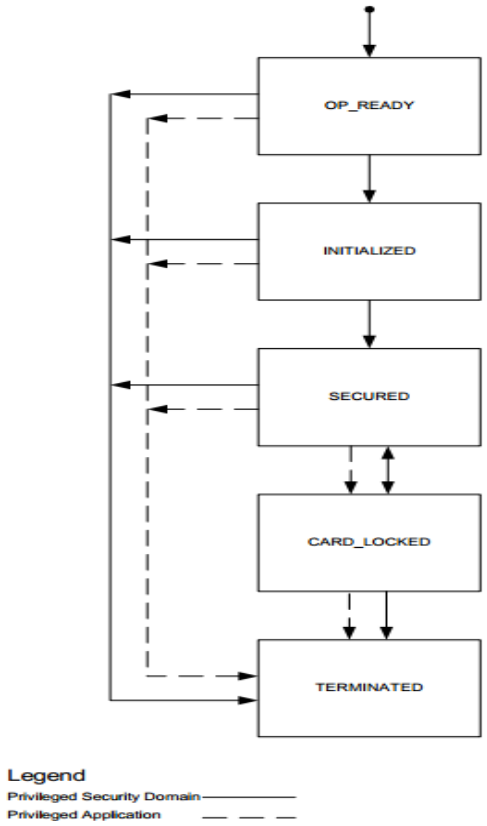

Figure 5: Smart Card Life Cycle State Transition [1]

4. Proposed approach to SE Access Control

The proposed SE access control is designed to achieve the two objectives as outlined in section III: a) provide a level of assurance of integrity of Android and its system module and b) control access to SE independent of the mobile OS and where that access is based upon the level of integrity of the OS. We believe this can only be achieved by establishing a chain of trust starting from the most secure parts of this system and working towards less secure parts of the system. This design is more secure than the existing solutions for the following reasons:

Whitelists are not stored in the system partition but instead are stored in the most secured area: the SE. SE is tamper resistant and is secure by design. In this new design SE not only serves as secure storage for the whitelist but is also capable of making decisions. It decides and then signals MTM to enforce access. The whitelist never leaves SE secure storage and there is no chance that it can be tampered with. This approach also ensures that only the most secure part of this system decides whether to grant access to the secure areas. In existing designs, less secure parts of the system, such as

the mobile OS decide to grant access to more secure areas of the system, and are flawed by this simple reason.

Secondly, SE is the most secure part of this system and therefore the chain of trust should start from it. In our design SE is trusted by the owner of the trusted domain of SE. SE then trusts MTM and MTM eventually trust the mobile OS. If this chain of trust cannot be established then access to SE is denied by default. This design ensures that only the valid and genuine MTM is allowed to communicate with SE. A trusted MTM then ensures that only a valid and genuine OS is allowed to communicate with SE. Presently this design is only limited to establishing a boot-time chain of trust. Future research in this area can be focused on establishing a run time chain of trust and allowing access based upon that.

Our proposed SE access control solution for Android based NFC smartphones makes use of all the above and other available primitives with little or no deviation from standards. SE, TrEE and REE all are available in modern NFC enabled Android smart phones. We select the SE as the most highly trusted module, to which access must be protected.

An MTM implementation is described in [18], and it is this implementation that we select for our proposed model. It makes use of virtualization under TrEE to further secure the MTM, its keys and registers against rogue commands and other types of attack. Figure 6 shows all design components and how they interact.

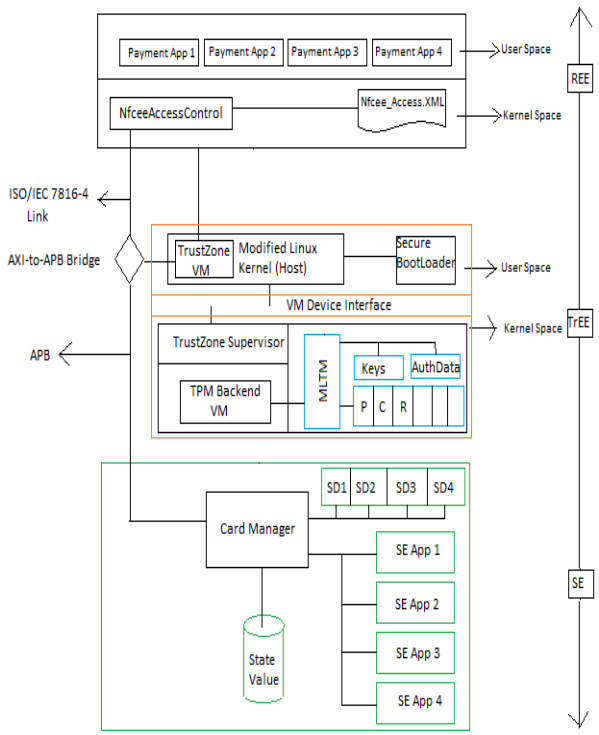

Figure 6: Proposed SE Access Control Model

The proposed design has three main components: SE, MTM and a mobile OS such as Android. SE is the most secure part of the design. The owner of the SE trusted domain ensures its integrity. MTM is a less secure part and the mobile OS is an insecure part. MTM ensures its own integrity and is designed in such a way that it won't work if integrity of its components is compromised. Our design operates at boot time to ensure SE boots first and then MTM is booted and then, finally, the mobile OS is booted. This sequence allows us to ensure that the first booted component is capable of measuring the integrity of the next component to be booted, and so on down the chain of trust.

The proposed model operates in following order, starting from system boot:

1. At system startup, the secure boot process in TrEE ensures that the TrustZone VM is loaded securely using a secure boot-loader.
2. The TrustZone VM then measures the Android OS boot loader and records its value in the PCR, using the TPM backend VM.
3. The Android boot loader then measures the Android kernel image and loads it. It also communicates this measurement to the MTM using the low level TPM API calls.
4. The Android kernel measures other system modules, reports to the MTM using standard the TPM API, and loads those modules.
5. The Android system module then measures any Android application image and reports that to MTM before loading it into memory.
6. Up until the time that the system modules are measured and loaded, the TrustZone VM makes sure that the NFC peripheral is locked. It can achieve this by programming the AIX-to-APB Bridge. This locking of the NFC peripheral prohibits any communication to the SE while the system is being measured and those measurements reported to the SE.
7. Once the system modules are measured and the results stored in the PCR, the MTM initiates APDU communication with the Card Manager using the low level API implemented in the TrustZone VM.
8. The MLTM receives a nonce from the Card Manager, encrypted with a shared secret. The MTM decrypts the nonce using the same shared secret.
9. Using the nonce in a hashing function on the PCR values, the MLTM computes the master hash.

10. The MTM then encrypts the master hash with the shared secret, and reports the master hash to the SE, again using the low level API implemented in the TrustZone VM. It resets the PCR after reporting, in order to protect against a replay of those measurement values.
11. Upon receiving the master hash from the MTM, the SE Card Manager decrypts it using the shared secret and compares the result to the value in its state value database. The Card Manager communicates its (positive or negative) evaluation to the TrustZone VM.
12. If the evaluation is positive, the TrustZone VM unlocks the NFC peripheral and the Card Manager engages in no further action. If the evaluation is negative, the TrustZone VM will not unlock the NFC peripheral, and the Card Manager changes its state from SECURED to CARD_LOCKED.
13. The Card Manager acts as an interface between the outside world and SE applications. In the LOCKED state, the Card Manager will not send or receive APDUs to or from the outside world, when those APDUs are destined to SE apps, essentially blocking communication between the outside world and any SE application.
14. The Card Manager will also go into the LOCKED state if any APDU, other than that intended for evaluation, is received after the boot process. This safeguard prevents bypassing of evaluation in any case.
15. If evaluation is negative, the TrustZone VM will not unlock the NFC peripheral. The SE Card Manager remains LOCKED until the system is rebooted. The Card Manager will change its state from CARD_LOCKED to SECURED upon reboot.

The proposed model deviates slightly from the TCG MTM, GP card and SE access control specifications; it deviates in following ways:

1. The Card Manager must store state values for evaluation. The Card Manager can easily do that by storing the values in its application data space.
2. The evaluation routines have to be implemented in the Card Manager. These routines closely match or exceed the current functionality of the GP Execution Environment (OPEN) as described in GP's Card Specifications v. 2.2.0-15 [1]. OPEN is, basically, the card management arm of Card Manager.
3. Nonce generation and communication with MTM functions must be incorporated into the Card Manager.
4. Fall back mechanisms and exception handling must be incorporated into the Card Manager.
5. A low level API designed to communicate with the SE must be implemented in the TrustZone VM. This is an addition to the TrEE based MTM design mentioned in [18].
6. Conversion of APDU and API calls inside TrEE can be achieved by the methods described by [19].

In addition to the aforementioned, a provision method must be implemented, in order to update the Card Manager's state values and the secret shared between the MTM and Card Manager. This provisioning process includes the following:

1. MTM software must be loaded securely into TrEE at device manufacturing or initialization time.
2. The MTM must generate its own key pair using its cryptographic functions. The public portion of this key and the public portion of the Card Manager key must be signed by the same or its delegated authority.
3. The MTM and Card Manager must agree on a shared secret using asymmetric cryptography.
4. This shared secret must encrypt all subsequent communication between the MTM and Card Manager.
5. State values must be populated, e.g. by the Issuer of the SE, using Over-the-Air (OTA) updates and GP's Secure Channel.
6. RID certificates must be appropriately coupled with executable modules inside the TrEE, in order for a secure boot to occur inside the TrEE.

Once this proposed access control model is implemented, the current SE access control implementation (as shown in figure 1) is sufficient to control access for third party Android apps. As the mobile operating system will get access only if its integrity is verified using MTM and SE, extended trust can be established between the SE and mobile operating system. Since extended trust is established, the operating system can safely make use of whitelists to protect against any rogue Android application.

This model is an alternative to the current SE access control models and we have argued that this model is a better defense against rogue access. However, this model does not completely eliminate the risk of denial of mobile payment services. DoS attacks against mobile services and devices is still possible. If an attacker

successfully corrupts a mobile OS system partition or system module, the mobile device is rendered unusable and a denial of mobile payment service does happen for that particular user of the mobile device. This model does protect a mobile device from becoming completely unusable, as a corrupted OS can be reinstalled easily and SE is reusable after that. On the other hand a 'bricked' mobile device's SE is totally unrecoverable.

5. Conclusion

The current design and implementation of SE access control is flawed, and is vulnerable to different kinds of attacks that threaten to compromise the whole system. We propose a theoretically sound and portable trusted computing model and design. However, the concepts and logical model presented in this paper must be implemented and tested for full assurance in real world applications. It must also be noted that our design ensures the integrity of a device OS to enforce SE related security only at boot time, but not at run time.

6. References

[1] GlobalPlatform, *http://www.globalplatform.org*, (Access Date: 15/9/ 2012).

[2] N Elenkov, "Exploring Google Wallet using the secure element interfaces," *http://nelenkov.blogspot.ca/2012/08/exploring-google-wallet-using-secure.html*, (Access Date: 30/8/ 2012).

[3] XDA Developers Forum, "[02/02/12] Google Wallet v1.1-R48V4 - ICS v4.0.3+," *http://forum.xda-developers.com/showthread.php?t=1311072*, (Access Date: 22/10/ 2012).

[4] R. Johnson, Z. Wang, C. Gagnon, A. Stavrou, "Analysis of Android applications' permissions," Software Security and Reliability Companion (SERE-C), 2012 IEEE sixth international conference on, vol., no., pp.45-46, 20-22, June 2012, *http://doi: 10.1109/SERE-C.2012.44*.

[5] A. Gargenta, "Deep dive into Android security", Presented at the Android Developer Confernce, San Franscisco CA, USA. 2012, *http://marakana.com/static/tutorials/AnDevCon2-DeepDiveIntoAndroidSecurity.pdf*, (Access Date: 10/11/ 2012).

[6] N. Elenkov, "Accessing the embedded secure element in Android 4.x," Aug 22, 2012, *http://nelenkov.blogspot.ca/2012/08/accessing-embedded-secure-element-in.html*, (Access Date: 10/11/ 2012).

[7] L. Francis, G. Hancke, K. Mayes and K. Markantonakis, "Practical NFC peer-to-peer relay attack using mobile phones," In 6th international conference on Radio Frequency Identification: Security and Privacy Issues (RFIDSec'10), Siddika Berna Ors Yalcin (Ed.). Springer-Verlag, Berlin, Heidelberg, 35-49, 2010.

[8] Trusted Computing Group, *http://www.trustedcomputinggroup.org/*, (Access Date: 08/11/ 2012).

[9] H. Uppal, "Enabling trusted distributed control with remote attestation," August, 2012, *http://people.cs.umass.edu/~hardeep/Thesis.pdf*, (Access Date: 13/11/ 2012).

[10] K. Dietrich and J. Winter. 2008. "Secure Boot Revisited," In Proceedings of the 2008 The 9th International Conference for Young Computer Scientists (ICYCS '08). IEEE Computer Society, Washington, DC, USA, 2360-2365, DOI=10.1109/ICYCS.2008.535, *http://dx.doi.org/10.1109/ICYCS.2008.53*.

[11] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome and J. M. McCune, 2012. "Trustworthy execution on mobile devices: what security properties can my mobile platform give me?" In Proceedings of the 5th international conference on Trust and Trustworthy Computing (TRUST'12), S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer and M. Reiter, (Eds.). Springer-Verlag, Berlin, Heidelberg, 159-178. DOI=10.1007/978-3-642-30921-2_10, *http://dx.doi.org/10.1007/978-3-642-30921-2_10*.

[12] J. Ekberg and S. Bugiel, 2009, "Trust in a small package: minimized MRTM software implementation for mobile secure environments," In Proceedings of the 2009 ACM workshop on Scalable Trusted Computing (STC '09). ACM, New York, NY, USA, 9-18, DOI=10.1145/1655108.1655111, *http://doi.acm.org/10.1145/1655108.1655111*.

[13] J. Grossschadl, T. Vejda, D. Page "Reassessing the TCG specifications for trusted computing in mobile and embedded systems," Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, vol., no., pp.84-90, 9-9 June 2008, doi: 10.1109/HST.2008.4559060.

[14] M. Landsmann, "Evaluating an MTM based security concept for Linux-kernel grounded mobile systems," Bachelor Thesis, Dept. of Comp. Science, Hamburg Univ. of Applied Sci., Hamburg, 2011, *http://opus.haw-hamburg.de/volltexte/2012/1447/*.

[15] M. Lemay, C.A. Gunter, "Cumulative Attestation Kernels for Embedded Systems," Smart Grid, IEEE Transactions on , vol.3, no.2, pp.744-760, June 2012, doi: 10.1109/TSG.2011.2174811.

[16] E. Haselsteiner and K. Breitfuß, "Security in Near Field Communication (NFC), strengths and weaknesses" Philips Semiconductors, June 2010.

[17] Google Inc. Android SDK, 2013, *http://developer.android.com/sdk/index.html*, (Access Date: 19/01/ 2013).

[18] P. England and T. Tariq, "Towards a programmable TPM," In proceedings of the 2nd international conference on Trusted Computing (Trust '09), L. Chen, M. J. Chris and A. Martin, (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-13, 2009, DOI=10.1007/978-3-642-00587-9_1, *http://dx.doi.org/10.1007/978-3-642-00587-9_1*.

[19] K. Dietrich and J. Winter, "Implementation aspects of mobile and embedded Trusted Computing," In proceedings of the 2nd international conference on Trusted Computing (Trust '09), L. Chen, M. J. Chris and A. Martin (Eds.). Springer-Verlag, Berlin, Heidelberg, 29-44, 2009, DOI=10.1007/978-3-642-00587-9_3, *http://dx.doi.org/10.1007/978-3-642-00587-9_3*.

[20] G. Alpár, L. Batina and R. Verdult, "Using NFC phones for proving credentials," In proceedings of the 16th international GI/ITG conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB'12/DFT'12), Jens B. Schmitt (Ed.). Springer-Verlag, Berlin, Heidelberg, 317-330, 2012, DOI=10.1007/978-3-642-28540-0_26, http://dx.doi.org/10.1007/978-3-642-28540-0_26.