# Trusted Mobile Platform

## Software Architecture Description

10/27/2004

Trusted Mobile Platform
NTT DoCoMo, IBM, Intel Corporation

File Name: TMP_SWAD_rev1_00_20040405.doc

# Change History (Informative)

| Type of Change | Date | Section | Description |
|---|---|---|---|
| Rev 1.00 | 06/22/04 | | Revision 1.0 for release |

## Trusted Mobile Platform

## Software Architecture Description

Rev. 1.00

June 23, 2004

## Copy Right Notice



## Status

This is a stable revision of the Trusted Mobile Platform Software Architecture Description that was agreed upon by Trusted Mobile Platform Promoters.

*Contents*

# 1. INTRODUCTION

Trusted Mobile Platform (TMP) is comprehensive end-to-end security architecture for mobile wireless platforms. It consists of the hardware and software architectures, as well as the protocol specifications. This document describes the software components that take advantage of the hardware components (such as the Trusted Platform Module and hardware domain separation) and that enhance the security of the platform. Some of these software components also bridge the security features with the protocol defined in the protocol specifications.

The Trusted Mobile Device (TMD) software architecture is designed so that these components are operating system independent. It also accommodates different security levels. Each component has different requirements that are applicable to the three different security levels defined for a TMD. For example, biometric user authentication is optional for a Security Class 1 TMD.

## 1.1. Document Structure

This document is structured as follows:

- Section 2 lists related documents
- Section 3 provides an overview of the software architecture
- Section 4 (Integrity Measurement), Section 5 (Domain Separation), and Section 6 (Access Control) describe the basic security features of the operating system
- Section 7 describes encrypted file systems and persistent memory protection mechanisms
- Section 8 (Cryptographic API) and Section 9 (TSS) describe the library software that provides cryptographic capabilities to application programs
- Section 10 (User Authentication) and Section 11 (Trusted User Interface) show guidelines to design secure user interface.
- Section 12 describes TMD device management
- Section 13 describes Trust Level Guidelines, including relationship to software certification

- Appendix A provides sample implementation options
- Appendix B provides a list of used acronyms and their definitions

# 2. Related Documents

[1] FIPS PUB 140-2 *Security Requirements for Cryptographic Modules*

[2] PKCS *Cryptographic Token Interface Standard*

[3] RFC 2119

[4] *Trusted Computing Group (TCPA) Design Philosophies and Concepts* Version 1.0

[5] *Trusted Computing Group (TCG) Main Specification* Version 1.1b, http://www.trustedcomputinggroup.org/, February 2002 (also known as *Trusted Computing Platform Alliance (TCPA) Main Specification* Version 1.1b)

[6] *TCPA Software Stack (TSS) Specification* Version 1.0

*[7] TMP Security Requirements*

*[8] TMP Hardware Architecture Description*

*[9] TMP Protocol Specification Document*

**[10]** *Provisioning Bootstrap 1.1*", **Open Mobile Alliance**

[11] *OMA Provisioning Architecture Overview Specification*", **Open Mobile Alliance**

[12] *SyncML Device Management Protocol*, version 1.1", SyncML

[13] *SyncML Device Management Trees and Descriptions*, version 1.1", SyncML

[14] *SyncML Device Management Standardized Objects*, version 1.1", SyncML

[15] *SyncML Device Management Security,* version 1.1", SyncML

[16] Trusted Computing Group - Main Specification - Version 1.1b 22 "*Mandatory / Optional Functions*", Pages 150-176, February 2002

*[17]* Schneier, Bruce; Applied Cryptography, Second Edition, John Wiley & Sons, 1996. *"Section 22.1 Diffie-Hellman*

[18] ANSI X9.63

[19] Schneier, Bruce; Applied Cryptography, Second Edition, John Wiley & Sons, 1996. *"Section 20.1 Digital Signature Algorithms"*

[20] ANSI X9.62

[21] Common Criteria Part: Security functional requirements, Aug 1999, Ver. 2.1

[22] BioAPI Specification Version 1.1, March 16, 2001, the BioAPI Consortium

# 3. Architecture Overview

A Trusted Mobile Device (TMD) allows many different applications, either Java applications or native applications, to run. Some applications are relatively more trusted (e.g., bank applications) and others are less trusted (e.g., animated graphics in a browser). These applications use the software services provided by the TMD system software. At a minimum, the core services, such as security services (cryptographic services, authentication services, etc.) and resource management (memory, process, etc.) must be assumed to be trusted. There is also a trusted computing base (TCB) that contains a minimal set of trusted components including the trusted boot code (Core Root of Trust for Measurement). Malicious applications may try to attack the TCB by modifying some part of it. Thus, the TCB needs to be protected from tampering. In Security Class 2 devices, the TCB's integrity is measured by the TPM. In Security Class 3 devices, the TCB is also protected by hardware storage mechanisms after trusted boot. The TCB must also able to detect tampering through integrity measurements.

Different applications running on the same TMD may belong to different security domains. A malicious application program may try to attack other applications. The TMD's domain separation provides protection against this type of attack. At the same time, the separation model needs to allow flexible data exchange if the application's access control policy allows data exchange between applications. Secure data exchange is achieved by an Inter Process Communication (IPC) mechanism. Access to a specific domain is permitted, based on the TMD's access control model, according to the access control policies of the target domain.

In addition, the TMD software should provide a set of security services, including cryptographic services, encrypted storage services, device management services, and user authentication services. These are all integral parts of the TMD software architecture. It should be noted that in an ideal case, all these components should have ideally equal strength. Attackers always try to break the weakest part of the system. Balancing the trust level of each component is a cost-effective way to design a secure TMP device.

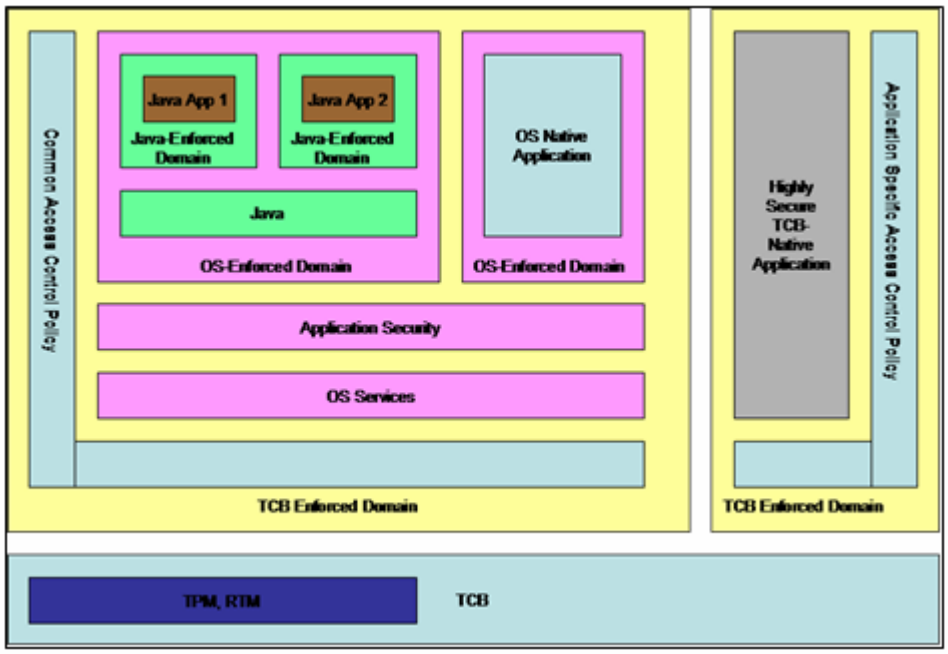


*Figure 3-1 Software Reference Architecture with TCB*

Figure 3-1 shows the reference software architecture of a TMD. The operating system relies on the device's trust in the hardware TPM, the RTM, the Trusted Computing Base (TCB), and protection mechanisms in the CPU. The operating system kernel and a minimal set of related security services, together with the hardware, constitute the TCB. Applications run within domains that are separated from other domains. The TCB-enforced domain mode protects highly secured applications. These applications have their own access control models and policies to protect their resources. OS-native applications run within an OS-enforced domain (e.g., a process space) and are subject to the access control mechanism provided by the OS. If the applications are written in Java, Java's domain separation mechanism is employed.

## 3.1. Trusted Computing Base (TCB)

The Trusted Computing Base (TCB), in the TMD software architecture, is a security boundary that provides a level of trust in terms of the software integrity and the domain separation. The TCB can be implemented in several different ways. Class 1 devices can rely on a closed system assumption (no user-modifiable system software) for integrity and Java sandbox model for domain separation without having any special

hardware. For Security Class 2 devices, a hardened operating system with a trusted bootstrap using a TPM would constitute a TCB. In Security Class 3 devices with a hardware-based domain protection mechanism, the TCB is a trusted microkernel that is small enough to be verified. It is clear that the level of trust that may be placed in the TCB will vary according to the class of the device.

## 3.2. Integrity Measurement

In the TMD software architecture, several components need to reside in the TCB (or at least these components need the use of the security services provided by the TCB). Therefore it is very important that the TCB's integrity is assured to be an appropriate level for the trust level of the device. The use of integrity measurement in a trusted bootstrap sequence is the [5] defined process of taking hash values of the software configuration. These measurements can also be securely reported to remote agents. Integrity measurements are dependent on two hardware components, the TPM (Trusted Platform Module) and the CRTM (Core Root of Trust Measurement).

Mobile devices usually have a long rebooting cycle. Some cellular phones are rarely turned off during their entire lifecycle. The longer the device is used after the bootstrap, the greater the chance of an attack on the TCB. It is desirable that the TCB integrity be measured periodically, even after bootstrap, so the TMD software architecture also defines optional runtime integrity checking.

A TMD must be able to share information regarding the current configuration of the platform with remote parties that desire to participate in some type of transaction. The TMD must be able to attest to its configuration in a trusted way that ensures that the attestation exchanged is the one collected by the trusted hardware and software.

## 3.3. Domain Separation

In TMD's, the TCB should protect an application from other potentially malicious applications. This is done by the TMD's domain separation mechanisms. Specifically, domain separation prohibits illegal access to memory (address space) used by other applications.

Domain separation can be achieved by several different mechanisms. When the applications are written in Java, Java language-based separation, and Java-based access control mechanisms, such as JAAS (Java Authentication and Authorization Services) and OSGi, can provide application protection. For applications executing the native machine language, the TCB should provide the necessary protection. Usually, an operating system has the ability to separate one process space from another using the hardware memory management unit. For the higher degree of protection in Class 3 devices, the TCB uses additional hardware support of the processor. These mechanisms provide different levels of security and should be used accordingly.

## 3.4. Access Control Model

For highly secure applications, access control is based on the domain separation provided by the TCB. Sometimes applications need to share information. For example, an address book application may want to allow read access to its database from other applications. This kind of security policy needs to be enforced by some access control mechanism. As is the case in other components, the Trusted Mobile Device access control model allows various different mechanisms to be employed. In Security Class 3 devices, a highly secure application has its own access control mechanisms and policies to protect it from attacks coming from outside of the domain. This is the most flexible yet most secure model. However, it requires each application to implement its own access control mechanism. OS-native and Java applications can rely on the access control models provided either by the OS or Java.

## 3.5. Other Security Services

Besides protecting the platform and the applications running on it, a Trusted Mobile Device also provides several security-related services. These services include cryptographic services, encrypted storage services, device management services, and user authentication services.

# 4. Integrity measurement

A trusted platform includes software and hardware components that implement functions critical to the trustworthiness and security of the platform. A trusted platform must ensure that these critical components are not modified or replaced. The trusted platform validates the integrity of these components by calculating a cryptographic hash that is used to provide a measurement of the component. The Trusted mobile device can also authenticate the source of a component by checking a digital signature applied to the component. The Trusted mobile device must perform these checks for all of the software and hardware components that comprise the TCB. It may also check the integrity and authenticity of additional hardware and software components that are not part of the TCB.

Security Class 1 TMDs do not have requirements to perform trusted boot and to measure the platform integrity. Security Class 2 and Class 3 TMDs must satisfy the following set of requirements for integrity measurements (Refer to the *Trusted Mobile Platform Hardware Architecture Description*, Section 4.4 for a definition of the three security levels):

- Perform an integrity check of the code that comprise the TCB after power-up
- Perform an integrity check of security and trust relevant hardware
- Measure and log sequences of software events
- Perform integrity measurements of applications and other software components that are not part of the TCB
- Provide the results of integrity measurements to applications
- Gather integrity measurements using trusted hardware and software
- Securely store integrity measurements so that they cannot be modified by un-trusted applications
- Initiate trusted boot by CRTM residing in the Root of Trust hardware
- Employ integrity measurements techniques defined by the [5]

## 4.1. Trusted Boot

Trusted boot is based on the transitive trust model defined in the [5]. Transitive trust

allows the trust boundary to be systematically extended from a small, formally trusted root to additional hardware and software components that are not initially part of the platform root of trust. The transitive trust model starts with a small trusted element, in this case the CRTM residing in the ROT hardware. The CRTM must be rigorously tested and its manufacturing processes tightly controlled to guarantee that the code has been implemented correctly and is correctly loaded onto the TMD platform. Using the hardware ROT to store the CRTM guarantees that the code cannot be modified and that this code is the first code to execute on power up. The CRTM, taking advantage of the platform's TPM, measures, validates and records the integrity measurements of other software components on the platform. Components with recorded measurements that match stored measurements provided by a Validation Entity (VE) are considered trusted and can in turn be used to measure additional hardware and software components. Figure 4-1 shows a block diagram representation of the transitive trust process.

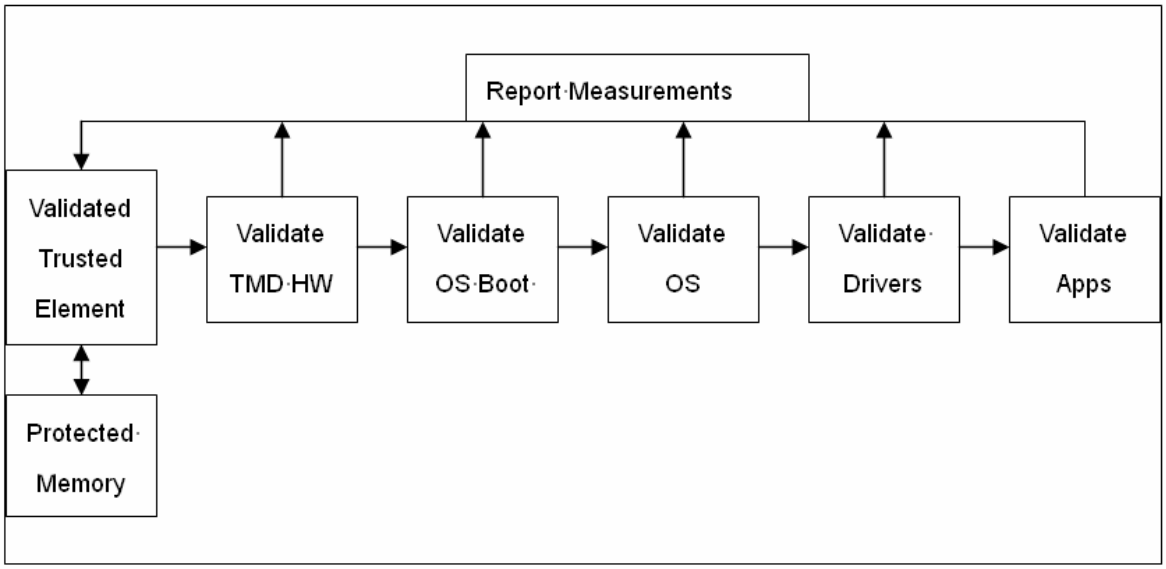


*Figure 4-1 Transitive Trust Process*

There are three components required to perform platform integrity metrics. These components are:

1. The measurement process used to collect the metric
2. The predicted values of measured component
3. The actual values of measured component

The process to collect the platform integrity metrics and to compare the measured values to predicted values is defined by the CRTM. The CRTM is stored in a ROM or a flash memory with lockable boot area. The CRTM uses the TPM on the platform to generate and store the measurement values. The measurement specifics, which includes the definition of what get measured and the order of measurement, is platform dependent and is defined by the platform builder. In a simple example, Figure 4-2 shows two approaches that can be used to measure an OS and three applications. In the first instance, one measurement is made on the combined OS and three applications. In the second instance, the OS and the three applications are measured independently. Both measurement processes are valid and either may be used. To collect these measurements, the CRTM must interface to the TPM that allows the CRTM to transfer the data being measured to the TPM and collect the measurement when the process is completed.

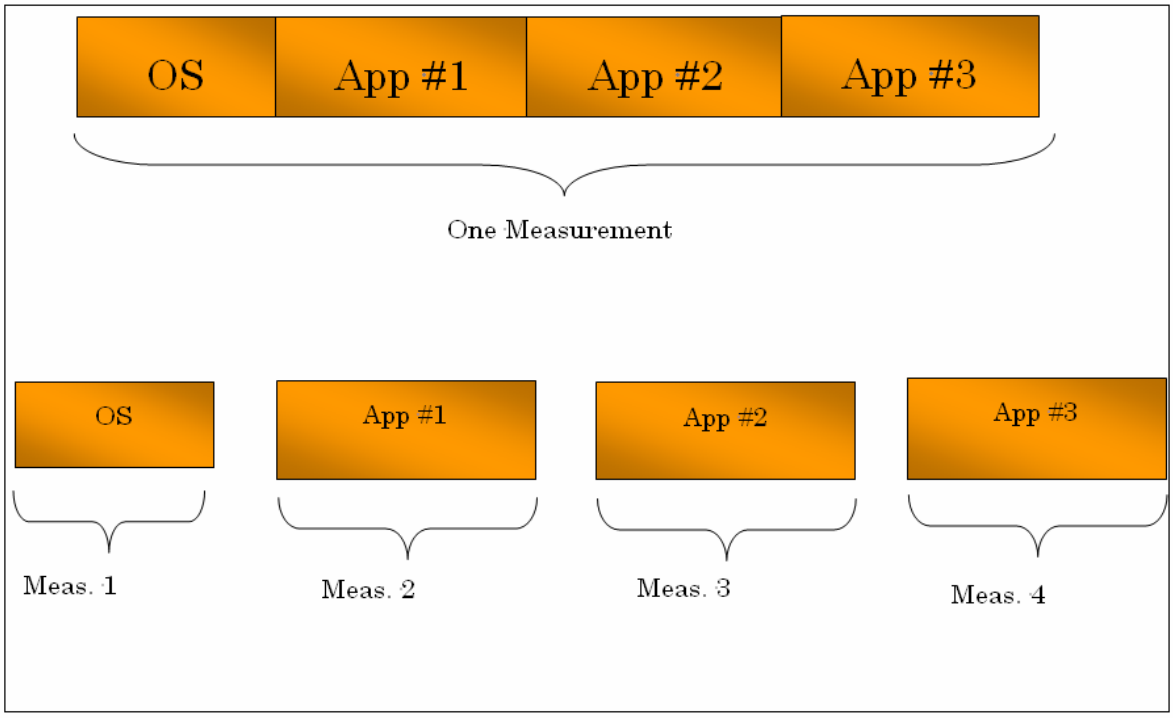


*Figure 4-2 Methods of Integrity Measurements*

## 4.2. Measurement Values

The actual value of a measured platform component is a cryptographic hash based on

the SHA-1 algorithm of the object being measured. Two trusted elements, the CRTM and the TPM, are used to ensure that the value is measured accurately and that the measured value cannot be modified. The measured value is then compared to a predicted ("known good") value. If the measured value and predicted value match, the measured component is validated as not being modified. If the measured and predicted valued do not match, then the component has been modified and cannot be trusted. The CRTM responds by notifying the user of this condition.

## 4.3. Predicted Measurement Values

Since the trustworthiness of the TMD depends upon the comparison between measured and predicted values, it is essential that the predicted valued be entered and stored on the platform in a highly reliable way. To ensure the correctness of the validation data on the platform, that data can be stored on the platform in a credential or may simply be digitally signed by the VE. There are also hardware options that could be used to reliably store predicted measurement values such as One Time Programmable (OTP) bits in flash memory. In addition to being stored reliably, the predicted values must also be reliably inserted onto the platform. Potential VEs include equipment manufacturers, OSVs, ISVs, and carriers. VEs generate the validation data (predicted measurements) using any method that allows them to generate the SHA-1 hash value for their component(s). The predicted measurement values are loaded to the trusted platform as part of the manufacturing and provisioning process. The approved values for components on the platform are not secret, but they must be protected against being modified. Protecting the list of approved values by a checksum stored in the TPM does this.

### 4.3.1. Authenticated and Secure Boot

The [5]defines two types of trusted boot: authenticated boot and secure boot. As used in this document, "trusted boot" refers to both types of trusted boot, and both types of trusted boot are allowed, based on the policy of the operator. In authenticated boot, the CRTM starts by obtaining the measured integrity metrics. The measured integrity metrics are compared to the predicted values inserted by the VEs. At that point, the platform attempts to boot normally, even if the measured and predicted measurement values differ. Secure boot also starts by collecting the measured

integrity metrics and comparing them to the predicted values. If the measured value matches the predicted value, the SW is trusted and the boot continues normally. If the measured and predicted values differ, a policy-defined action is taken to limit the services provided by the platform. In essence, since the platform cannot be trusted in this case, then any trust related services should be disabled.

## 4.4. Run time integrity measurements

The [5] defined a method to record system metrics after the OS is loaded and the system is running. The measurement of the system is performed using the TPM with the result being stored in a Platform Configuration Register (PCR) internal to the TPM (Refer to Section 6 of *the TMP Hardware Architecture Description*). In this scenario, a PCR in a TPM does not store a measurement based on one event (like boot) but stores an integrity measurement in a PCR related to a sequence of events that take place on the TMD. The process tracks both the value of the components being measured and the order that they are executed and measured.

Internal to the TPM, the PCRs are used to store platform state. At power up, the PCRs are set to zero. Run time integrity metrics are recorded by calling a TPM_Extend operation. In an extend operation, the measured 160 bit integrity metric for an event is concatenated with the current 160 bit value in a PCR to form a 320 bit value. This concatenated value is then hashed using SHA-1 and the resulting 160 bit result from the SHA-1 hash is loaded back into the PCR. This resulting value is dependent upon the component currently being measured as well as all of the components that were previously measured by prior extend operations relative to that PCR.

Unlike personal computers, the usage model for TMDs is that they are personal communications devices that remain powered on for very long periods of time and are only infrequently powered down. The long time periods between power-down operations, which reset the PCRs, makes it very difficult to gather and log meaningful run time integrity measurements. Over the long period of time between power downs, the TMD will perform a complex and arbitrary sequence of trusted and un-trusted operations. There are two approaches that can be used to perform real time

measurements on the platform. The first approach is to use a run time check based on integrity metrics stored on the platform. The run time integrity measurements can be executed at the specific point in time that the TMD needs to attest to its trustworthiness. The run time integrity measurement would measure only those software components needed to perform the specific trusted operation that is being requested and would enable only the software components needed for that function. This action could be preceded by a reset to the platform that would terminate all operations and force the initiation of trusted boot using the CRTM. Subsequent to boot, the requested process can be measured and its integrity metric compared to that provided by the VE. If the measured value matches the value stored on the platform, the application is launched.

The second approach for run time integrity measurements allows for the VE to supply the integrity measurement to the platform. The protocol for this operation is described in sections 5.4 and 5.5 of the Trusted Mobile Platform Protocol Specification Document.

# 5. Domain Separation

Trusted platform architecture depends on domain separation (sometimes referred to as "domain isolation".) Applications and Services execute in isolation from each other without observation and interference from other applications, services, and I/O devices. Together with Access Control (see next section), domain separation protects code from intentional or unintentional modification, and data from unauthorized access.

Domain separation implies a programming model in which a domain has complete control over what parts of its runtime memory is shared with other domains. Typically, the only form of sharing is that required to implement a trusted Inter-Process Communication (IPC) mechanism. The programming models of most COTS operating systems do not meet this requirement, and therefore could not be classified as Security Class 3.

The programming model also implies that an application or service executing inside a domain has complete control over what code gets executed inside that domain. This is typically achieved by executing the code from a single signed manifest associated with the domain. Current COTS operating systems do not meet this requirement; another reason why a COTS OS would not be classified as Security Class 3.

Different systems architectures provide domain separation with different degrees of protection. A Security Class 1 device provides less rigorously protected domains only, which we will call "Managed Domains" in the following. A Class 2 device may rely on the process memory space isolation provided by a COTS operating system. We will call these domains "OS-enforced Domain". A Class 3 device requires the strong domain separation provided by a formally-evaluatable TCB and programming models that control the sharing of memory. We will call these domains "TCB-enforced Domain".

## 5.1. TCB-enforced, OS-enforced and Managed Domains

There are three forms of domain separation that may co-exist in a system; TCB-enforced, OS-enforced and Managed Domains. The first two are process spaces in which native machine language application code executes, while Managed Domains are

execution stacks for Java applications. TCB-enforced domains and OS-enforced domains are implemented by the native operating system. Managed Domains are implemented by the Java Virtual Machine, or by another interpreter-based runtime environment. An instance of the Java Virtual Machine would execute inside a separate TCB-enforced or OS-enforced Domain, and may support multiple Managed Domains. There may be multiple instances of the Java Virtual Machine (in multiple TCB- or OS-enforced Domains), each supporting multiple Managed Domains. The concept of different Security Classes correlates with the mechanism through which the Domains are provided.

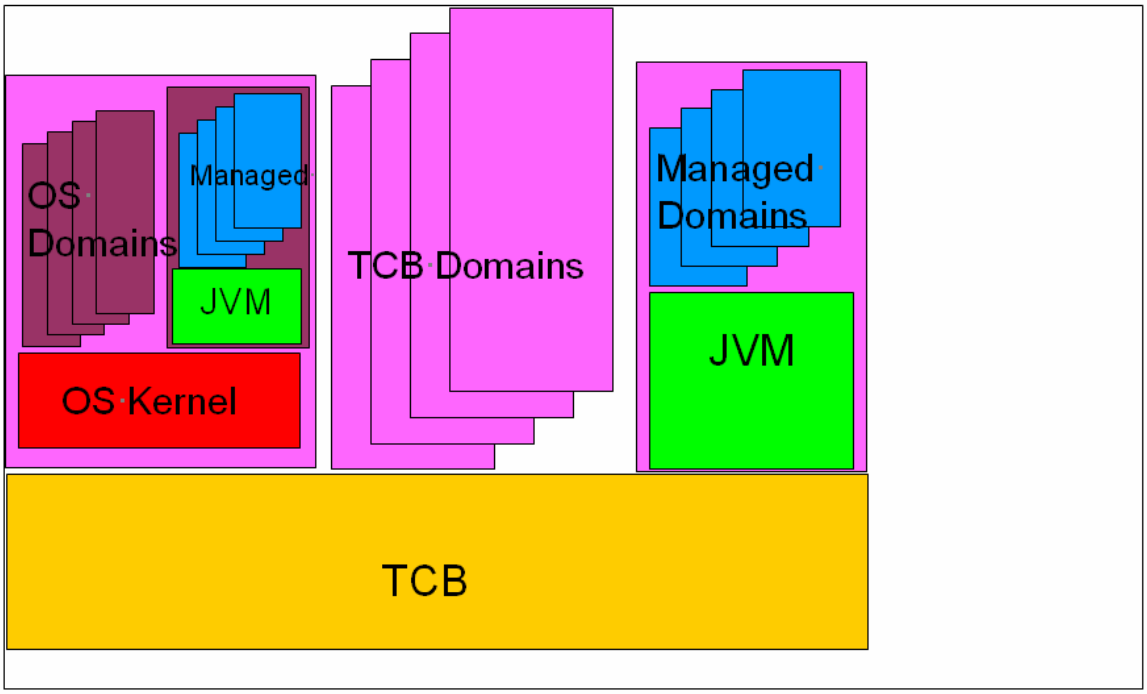


*Figure 5-1 Domain Separation Model*

## 5.2. Domain Separation Mechanisms and Security Classes

### 5.2.1. Security Class 1

Security Class 1 devices offer Managed Domains only. Such systems offer a limited protection to the services executing in the domains. Therefore it is advisable to use them preferably with tested and certified services. Although not as protected as the

Class 2 and Class 3 devices, services executing in these devices are better protected than in a system using a plain operating system without any domain concept.

### 5.2.2. Security Class 2

Some versions of existing operating systems have significant security enhancements. These enhancements take two different forms. One is to have extensive review (and sometimes formal certification) for less vulnerable implementations. For example, some commercial operating systems have a CAPP EAL-4 Common Criteria certification. Other enhancements come from adding tighter and finer-grained access controls mechanisms. SELinux and Trusted BSD are examples of such operating systems. These types of operating system can be used to provide Security Class 2 domain separation for a Trusted Mobile Device provided that is the appropriate forms of Access Control are supported, and the appropriate programming models are supported.. Note that the Security Class 2 Trusted Mobile Device must have a trusted boot capability, and it must be possible to measure its integrity periodically using the TPM. Any tampering with the operating system kernel would be detectable, and a reliant party may choose not to trust the system if tampering has been detected.

### 5.2.3. Security Class 3

For Domain Separation to achieve the strongest level of security, the kernel must be structured and implemented as a Trusted Computing Base (TCB). That is, it must be small enough and stable enough to be formally evaluated. (Whether it is actually evaluated is a business decision beyond the scope of this document). Such an operating system kernel is often referred to as a trusted microkernel on top of which operating system services are implemented as user processes inside Domains. This style of operating system is sometimes referred to as a Client/Server style operating system. The trusted microkernel provides an underlying Inter-Process Communication mechanism that allows domains to communicate. If such an Operating System were to be constructed from an existing COTS OS, the hardware may require memory management extensions.

Existing COTS operating systems typically do not meet this requirement, but there are some COTS OS that are microkernel-based (e.g., QNX), and may be suitable for adaptation to Security Class 3.

## 5.3. Managed Domains

A virtual machine exercises control over the bytecodes it is executing. The Java Virtual Machine and the Java byte-code verifier together try to ensure that no two Java applications interfere with each other; effectively providing Java Application Domain separation.

The strength of the separation depends on the trustworthiness of the Java runtime environment. Typically, the Java runtime environment is significantly more complex than a trusted microkernel, and therefore cannot provide the same level of assurance. If it is necessary to implement a highly secure application in Java, it would be possible to run such an application on its own instance of the Java runtime in its own TCB- or OS-enforced Domain.

## 5.4. Summary of Domain Separation and Security Classes

| Security Level | Security Class 1 | Security Class 2 | Security Class 3 |
|---|---|---|---|
| Basic structure | Standard OS with interpreter runtime environment (for example Java and OSGi) | Hardened COTS OS (with mandatory access control) | Microkernel (TCB) |
| Type of Domains | Managed Domains only | OS-enforced Domains, optionally managed domains | TCB-enforced Domains, optionally OS-enforced and Managed Domains |

# 6. Access Control Model

A critical facet of a Trusted System is Access Control; a Trusted System is intended to guarantee that only authorized access to data and resources is permitted.

The Trusted Mobile Device supports two access control models. The first one, called the Autonomous Model, relies on each application that manages resource to enforce its own access control policies. Each such application running in a domain is completely autonomous as if it were running on a separate machine in a network. The mechanism of its access control enforcement, the semantics of its access control policy language, semantics of its data objects, its internal architecture, and its communication protocols are unconstrained by this architecture; they are its own business. In this sense, the architecture is completely open to allow applications to innovate. This model is described in Section 6.1.

The second model (System-Mediated Model) relies on the underlying system software, be it the operating system or middleware (such as Java), to enforce access control policies. There has been several access control mechanisms proposed so far. Some are integrated into the operating system (e.g., mandatory access control systems such as SELinux) and some are features of middleware (e.g., Java's JAAS and OSGi). This model is described in Section 6.2.

Note that these two models are not mutually exclusive. As discussed in Section 5, a domain may have sub-domains and the outer domain and the inner domains may have different access control models. For example, in the Autonomous model, the TCB provides a strong domain separation but no or little access control support, thus the access control should instead be provided by the protected network application. On the other hand, one TCB-enforced domain can host another layer of execution environment for multiple sub-applications (e.g., multi-user operating system or Java), which in turn provides the System Mediated Model.

## 6.1. Autonomous Access Control Model

The model described in this section can be viewed as a "Set of sand-boxes with

privileges". Each Protected Network Application (Data Service Provider) is autonomous and is given a sand-box (a domain) in which to execute. The semantics of its objects, its internal architecture, and its wireless communication protocols are unconstrained by this architecture; they are its own business. In this sense, the architecture is completely open to allow Data Service Providers to innovate. Unlike the original concept of Java sand-boxes, these sand-boxes are given rights to use other services, by contract (or by default). The Data Service in each sand-box implements and enforces its access control policy as it sees fit within the confines of the domain[1].

The model could be seen as supporting an arbitrary number of special-purpose closed systems (the data services) in an open partitioned general-purpose system. The difference is that the special-purpose systems are provided with a set of platform services they can use to optimize development.

It is also possible that a suite of data service applications may be installed with a set of middleware services developed by the service provider. These middleware services may be represented as callable libraries in the data service domains or as protected domains in their own right (augmenting the services of the platform).

### 6.1.1. System Model

As described in chapter 5, the basic system model consists of domains in which each domain receives a degree of protection depending on the degree of security provided by the system. Underlying the domains is a privileged layer (called the TCB in security level 3) that creates and binds domains, and provides a mechanism to allow them to communicate. Services and Applications are provisioned into domains, and execute within their protection.

Given this basic system structure, there are many ways in which services and applications can be decomposed into domains, and these choices represent different approaches that could be taken by the system developer.

[1] In the literature this model is called Domain and Type Enforcement (DTE)

#### 6.1.1.1. Terminology

The Autonomous Access Control Model uses two terms frequently: Service and Application.
A Service is anything that provides an API library and can be “invoked” locally by an application.
A service may manage hardware resources or manage user-related objects.
An Application is anything that invokes a service.
An application may also have a UI.
An Application may also be a service. Typically, a Data Service Domain is an application that manages valuable objects for the benefit of the user. In some cases, a data service provider may also be a callable service with an API library for other client applications to use.

#### 6.1.1.2. First Principles

Applications and Services that enforce access control over resources or objects that they manage must be provisioned into separate domains. Only code which the service or application trusts will be allowed to execute in the domain alongside the service or application. Normally, this package of code would be delivered as a signed manifest to guarantee its integrity; no other code would be allowed to execute in the domain.

Domain isolation is required to protect access control policy enforcement. Since domain isolation forces service execution threads into separate processes (inside their domains), services must be invoked using an inter-process communication (IPC) facility provided by the underlying privilege layer. An IPC is usually an asynchronous queuing mechanism between processes. Much work has been done over the last two decades on IPC, but system developers will determine the actual choice of IPC mechanism based on available hardware and operating system facilities.

Usually, an RPC or Queuing runtime is layered on the IPC primitive and made available as a library for API development. This represents the “stack” required to invoke the service.

This style of operating system is commonly called a client/server model, since the IPC mechanism acts as a network between domains. All the same architectural principles

that apply to a secure network client/server model also apply to a client/server operating system with the privileged layer IPC acting as the network:

- There will likely be a directory mechanism to find services in domains
- Connections may need to be authorized
- Normally in Client/Server, a Server executes on behalf of the client's logged-in user (that is, the thread of execution on the server has a security context that identifies the user). However, to allow for mandatory access control, some services may permit execution on behalf of the client application (that is, the security context in the thread of execution on the server identifies the calling application).
- A service that manages resources (e.g., the file service, the TPM service) supports multiple client applications on the platform (possibly concurrently). The service is responsible for making sure that information (state) that it holds on behalf of one client application does not leak into the state of another application. In this sense, a server is stateful, and must protect the state of one application client from another.
- A server is responsible for enforcing its own access control policy over the resources/objects that it manages (e.g., as in a distributed file system).
- A server provides a client API library for applications to invoke it. This API library is bound into the client application manifest.
- The API library (in conjunction with the "stack") generates the protocol messages that flow on the IPC channel

#### 6.1.1.3. Service & Application Categories

Services and Applications fall into four categories:

1. Native Services: These services are divided into two sub-categories:
   a. Services that require authorization to use
   b. Services that do not require authorization to use

These services are either platform services that are provisioned on all platforms, or middleware services installed by a service provider as part of its data services implementation.

2. Native Network Data Applications (to which the User can subscribe): [These are the so-called Data Services]. These Data Services may also provide services for use by other applications, and a client API library to invoke them. Generally, use of this type of service by other applications would require authorization.
3. Native Local Applications (PIM, Games, etc) in two sub-categories:
   a. Those that use services requiring authorization

   b. Those that don't
4. Managed runtime applications in two categories:
   a. Those that use natives services requiring authorization
   b. Those that don't

### 6.1.2. Authorization

An application that calls a service requiring authorization, must acquire authorization credentials. These credentials can only be acquired from the owner of the service or a trusted third-party. The term owner, here, refers to an agent of the business that provides the service.

An authorization credential contains a statement of the rights granted in some policy language. The choice of policy language is at the discretion of the grantor (the service). In its simplest form, it is a right to bind to the service and call it (enjoying the full semantics of the interface), but it may contain finer-grained rights that limit the full semantics of the Service interface. A grantor or trusted third-party would sign the credential. Detailed definition of authorization credentials is at the discretion of the grantor.

The application developer, once all the authorization credentials have been acquired, would bind the credentials into the application manifest before provisioning. Acquiring authorization rights is part of a contractual process.

### 6.1.3. Access Control Model

The Access Control Model depends on Domain Isolation. Domain Isolation gives a service tamper-resistant execution so that it can enforce its access control policies. Every Native Service that requires authorization has two responsibilities:

- To honor the contractual rights of access to its objects; this entails the service being its own access control enforcement point.
- To protect the semantics of its objects; the semantics of an object may include discretionary access control over the object. In which case, the service implements and manages that discretionary policy and chooses the mechanism. An example would be the semantics of a file as provided by the File Service. Files would have ACLs (Access Control Lists).

An authorized IPC connection must be established before an application can invoke a protected service. This connection can either be established when the application domain is instantiated (early binding), or it can be established on the first use of the service (late binding). The connection is established by presenting the appropriate authorization credential to the domain binding mechanism in the privileged kernel. The API library finds the target service using the domain directory, and then requests the domain binding mechanism to establish a connection to the target service, passing in the necessary authorization credential. The domain binding mechanism mediates with the target service to request permission to bind. If the target service grants permission (based on the credential), the domain binding mechanism will bind the application and service domains. That is, a communication channel (IPC) will be created to allow the two domains to communicate. If permission is denied, no communication will be possible (by any means), and the requesting application will simply be told that a binding request has failed due to lack of authorization. The communication channel enables the application domain to exercise the interface of the service domain and thereby to use some subset of the semantics of the service domain's objects or resources. The service domain may tailor the channel, based on the fine-grained permissions, in order to optimize the invocation of the interface.

The binding and authorization mechanism between domains is entirely analogous to the establishment of an authorized network connection between client and server; the IPC mechanism is essentially a network that links the applications and services on the platform. Just as network services are discovered through some network directory mechanism, services on the platform are discovered using a local domain directory. Just as, in the network, the client RPC or Object runtime performs discovery and binding. On the platform, the runtime library performs the same discovery and binding services on top of the IPC mechanism. Just as in network security, some kind of token (e.g., a Kerberos ticket or a Digital Certificate) is required to authorize the connection, so also on the platform the authorization credentials bound into the client are used to authorize each connection that the client makes to a service. And finally, just as in network communication, the details of the binding mechanisms are hidden under the client API. The details of binding to a service are hidden under the client API on the platform as well.

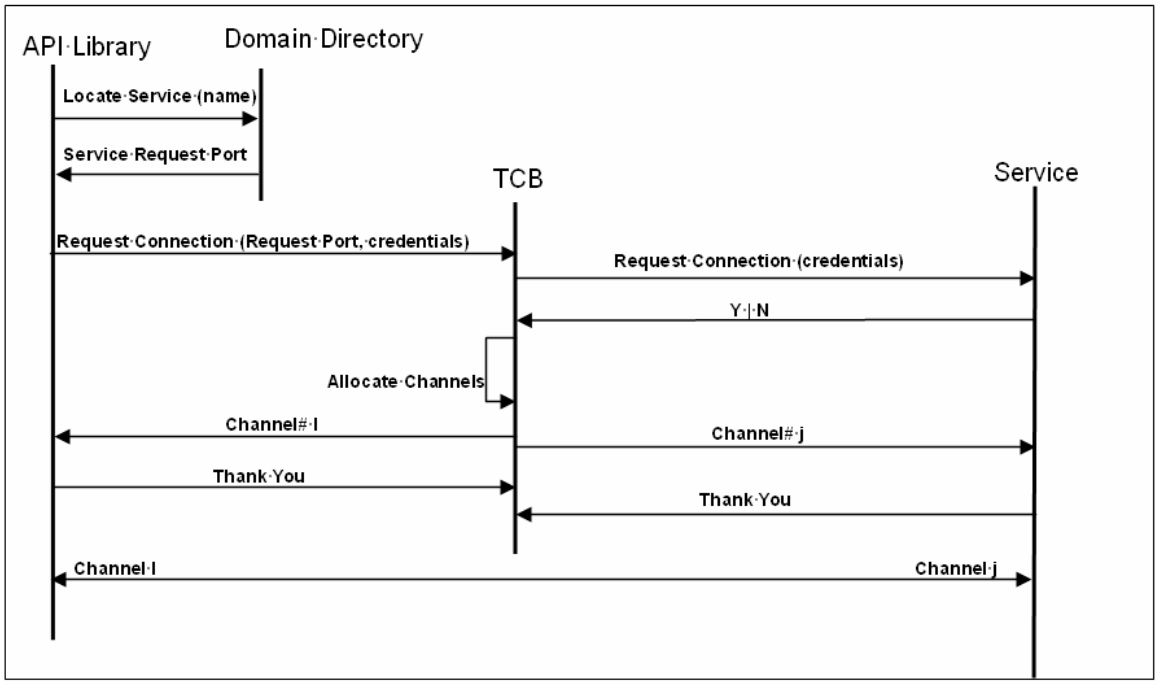


*Figure 6-1 Binding Flow*

### 6.1.4. Administrative Model

Administration starts with the manufacture of the device. At manufacture time, the device would be given a unique ID. This ID could be an Endorsement Key within the TPM (generated by the TPM manufacturer, as required in [5]), a public key (generated either by the manufacturer or the Network Operator, or both), or could be a generated serial number. In any case, the ID needs to be certified, either by creating an administrative data base entry for the device or certifying the public key [the hash of the public key could be the serial number]. The device has the ability to display its own serial number, so no external label is necessary. The ID could be used as the authority to authorize the provisioning of the device.

When the subscriber takes possession of the device, it must, if it is a phone product, be given a phone number, and that phone number must be associated with the serial number. This could be done by certification or by binding it into the administrative data base entry.

Each service that is provisioned (including platform services) needs keys to identify

them. These keys are generated by the provisioning service on the device. Platform services could use symmetric keys; third-party services probably need public keys. These keys are used to authenticate the service. For third-party services, the key would be certified by the provisioning service.

The provisioning service would create an entry in the local domain data base whenever a service (including platform service) is provisioned.

No on-device access control administration is necessary. Provisioning provides all the administration necessary. A Service Provider receives its authorization credentials, and a manifest generation tool embeds them in the manifest. When the service has been qualified to the satisfaction of the Network Operator, the Network Operator can certify its availability for subscription. When a consumer decides to pay for a subscription, the service can be provisioned onto the subscriber's device. Any updates to the service would require re-provisioning. Provisioning could be a push model or a pull model. The source could be either the Network Operator or the Service Provider (or even a third-party). But, the provisioning mechanism always guarantees that the security policy is properly administered.

## 6.2. System-Mediated Access Control Model

The Autonomous model is a simple yet flexible model. However, the application needs to implement its own access control model. In addition, the access control policy in the Autonomous model is completely controlled by the application because the application implements its own access control mechanism. This model is appropriate as long as the application owner can be fully trusted..

The System-Mediated Model provides a common access control model to multiple applications in a domain. The applications can share the same enforcement mechanism, the same syntax and semantics of the access control language. The applications do not need to implement their own access control mechanisms. In addition, access control enforced by the system (not by the resource owner) can protect sensitive resources even if the applications have security breaches.

### 6.2.1. Common Access Control Policy Language

At this time, a common access control policy language has not been determined in this specification. However, we envision that XACML (eXtensible Access Control Markup Language) is one of the candidates.

Section 8 (Access Control Model) in the Trusted Mobile Platform Protocol Specification introduces XACML (eXtensible Access Control Markup Language). On the other hand, each system software or middleware may have a different form for expressing access control policy. For example, in JAAS, the Java VM has a policy configuration file named java.policy that contains access control rules. Operating systems have file access control properties (e.g., access control bits in Unix file systems) or more sophisticated access control policy configuration (e.g., policy files in SELinux).

## 6.3. Summary of Access Control and Security Levels

| Security Level | Security Class 1 | Security Class 2 | Security Class 3 |
|---|---|---|---|
| Autonomous Model | optional | optional | optional |
| System-Mediated Model | optional | optional | optional |
| System-wide policy | optional | optional | optional |

In a Class 2 and Class 3 devices, at least one access control model should be supported.

# 7. Secure storage / Secure file system

The Trusted Mobile Device must provide “secure” storage facilities. The operative keyword here is “secure”, meaning that any confidential data for the specific entities in the system are never exposed to other entities (e.g. any other unrelated processes running on the same platform, or persons or servers connected to the platform through network traffic) and that the integrity of the data should be preserved.

There are several levels of secure storage in the Trusted Mobile Device. A TPM equipped TMD can provide a small amount of secure storage that is tamper resistant at the hardware level. Since TPM storage is implemented as hardware with some tamper resistance, it provides possibly the highest level of protection from both physical and logical tampering. However, the size of discrete TPM storage is very restricted mainly due to the hardware resources it can afford and it is not practical to store all sensitive information for every running application within the TPM. Adding this memory would greatly increase the cost of a Trusted Mobile Device. Therefore, the Trusted Mobile Device should provide alternative methods of secure storage. Security Class 2 platforms will not have the full set of hardware and software features to allow for secure storage without encryption. Even in Security Class 3 devices, encryption provides protection from physical attacks not protected by domain separation alone. Based upon the notion of protected storage in connection with [5], the TMD can effectively expand the amount of TPM storage to virtually as large as the amount of logical memory in the platform. This is attained by taking advantage of the Storage Root Key, a [5] concept explained briefly in the following section.

Another possible candidate for secure storage mechanism in system memory is to use domain separation, which is incorporated into the Trusted Mobile Platform with Security Class 3 devices (*see* Table 7-1 *Secure Storage/Secure File System with each Security Level*). Execution environments of running processes are rigorously isolated with domain separation mechanisms provided by software or hardware, allowing the possibility to attain secure storage facilities with domain separation quite easily.

Another consideration is necessary when protecting persistent data objects such as files. Ordinary Operating Systems (OSes) provide different levels of access control mechanisms for files based on the notion of user or group identities in the system. With

most OS's having a special privileged user (or super user, e.g. root in UNIX-like systems or Administrator in Windows), it is quite easy for a malicious user to access any file in the system, if he or she also manages to break-in and get special privilege. A solution that can be used to secure such non-volatile data is an encrypted file system, which can be seen in several PC or workstation OS's, commonly known as the Encrypted File System (EFS) in Windows2000/XP or Cryptographic File System (CFS) for several UNIX-like systems.

### 7.1.1. Summary of Secure Storage/Secure File System and Security Level

| Security Level | Security Class 1 | Security Class 2 | Security Class 3 |
|---|---|---|---|
| TPM Usage for Secure Storage | None or Just use TPM's internal storage | TPM's protected storage facilities | Same as left |
| Domain Separation for Secure Storage | None | Yes /w Hardened OS & Cryptographic Memory System | Yes /w Secure Kernel |
| Encrypted File System | None | Yes | Yes |

*Table 7-1 Secure Storage/Secure File System with each Security Level*

## 7.2. Protected Storage

As described in the previous section, [5] is designed to have unlimited volume of protected storage. This is attained under the notion of a key hierarchy, where only keys are stored in the physically tamper resistant TPM hardware. The keys, except for the root, are used to protect arbitrary data, which are all protected through encryption by using one of several other keys. The root key in the hierarchy is called the Storage Root Key (SRK) and it can be used to encrypt the keys within the second layer, and then these second layer keys are used to encrypt the third layers, and so forth. Since the SRK never leaves the TPM, it is sufficiently protected by the tamper resistance of the TPM alone (not necessary to use the protection provided by encryption). Other encryption-protected data or keys can be retrieved through the

TPM and stored within the normal storage area such as platform's ordinary memory structure or hard disk. Before using a datum or a key, one must submit it to the TPM and let the TPM do the required decryption.

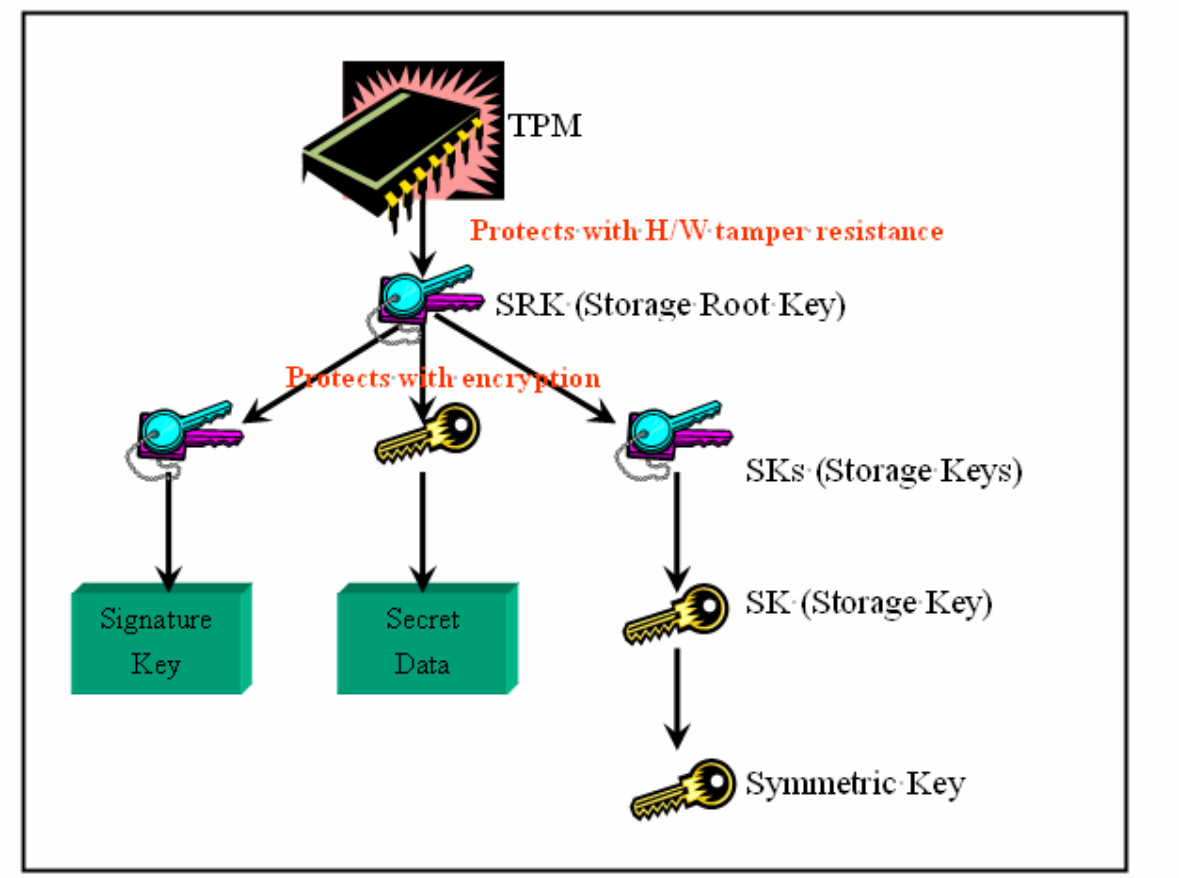


*Figure 7-1TPM Key Hierarchy*

As seen in *Figure 7-1* (*TPM Key Hierarchy*), storage keys can be either asymmetric key pairs or symmetric keys, and data at the leaf level of the tree are the actual data to be protected, such as signature keys, symmetric keys, or just arbitrary confidential data. There are only two types of keys. First is non-migratable keys, which are generated inside of the TPM and never leave it in plaintext form. The second is migratable keys, which can be either created outside the TPM or leave the TPM without any form of encryption. A migratable key is totally controlled by its creator's TPM and can never be utilized by inappropriate entities.

Two types of key operations can be used to conceal data: the seal operation is used to encrypt data so that encrypted data can only be decrypted under specified conditions,

while the other operation, bind, and simply encrypts the data.

Conformant Trusted Mobile Devices at Security Class 2 or Class 3 levels should implement the mandatory functions listed as [5] Mandatory/Optional Functions..

## 7.3. Protecting memory by encryption

Current COTS based multi-task operating systems have several drawbacks in protecting data stored in memory. They lack the ability to efficiently separate the memory space of one process from others. Although most operating systems incorporate virtual addressing mechanisms that realize the isolation of processes' memory spaces, there are still several sources for a process to interfere with other processes' memory data. For example, swapping or paging mechanism, which are used when physical memory is exhausted during the execution, makes the contents of one process' memory accessible to any other processes by putting the data onto the external storages. There also is a mechanism by which one process is able to examine and modify other processes' memory data on the fly, whose main purpose is to provide the debugger programs (or alike) the ability to manage the execution of other processes being debugged.

As for the Trusted Mobile Platform, Security Class 3 devices incorporate some kind of domain separation mechanism, which eliminates such threats. However, Security Class 1 and Class 2 devices do not have such mechanism and it is recommended that at least Class 2 devices incorporate some mechanisms for protecting the data in memory by encryption.

### 7.3.1. Encrypting swap devices

As for the swapping or paging operation, it is recommended that the operating system kernel encrypts the content being swapped onto the external storage. This swapped out data should be decrypted when they are swapped in to the memory.

### 7.3.2. Encrypting memory directly

Another possible solution to mitigate the threats being interfered by other processes is

to encrypt data directly while stored in memory. Of course it is rather difficult to encrypt data in memory because of both the encryption/decryption key management and the efficiency of the encryption/decryption operation.

However, there is one possible candidate for such protection scheme [1], which realizes the memory protection by encryption with reasonable overheads. It is recommended that Security Class 2 Trusted Mobile Devices incorporate such kind of countermeasure for memory sniffing. It may be useful even for Class 3 devices in order for the processes belonging to the same security domain to be able to protect themselves from each other.

## 7.4. Key and certificate management

Keys and certificates must be managed in such a way that no key is revealed to unauthorized entities.

### 7.4.1. Lifecycle Management

Keys should be managed appropriately throughout its lifecycle, that is, from generation to termination. For example, a key should be securely generated and the generated key should be protected from unauthorized accesses. In conformance with the Trusted Mobile Platform policies, lifecycle management of keys and/or certificates should follow the following precautions:

#### 7.4.1.1. Key Generation

Keys must be generated using a cryptographically strong pseudo/true random number generator (PRNG) as a source.

#### 7.4.1.2. Key Protection

Generated keys must be protected from unauthorized accesses, (e.g., adequate levels of access control must be provided, both in memory and in persistent storage). This can be done using TPM's tamper resistance, using encryption, and/or by using encrypted file system.

#### 7.4.1.3. Key Import/Export

Keys created outside a TMD must be handled with special care when imported onto the platform in order not to expose any fragment of the key to unauthorized entities during the import operation (e.g., migration mechanism defined by [5]).

#### 7.4.1.4. Key Certification

When an asymmetric key pair needs to be certified by a digital certificate, it is required to submit some kind of Proof-of-Possession (POP) evidence for the corresponding private key. The implementation should take care not to submit any data that can be used to deduce any part of the encrypted key.

#### 7.4.1.5. Key Backup

Key backup is required for several reasons. For instance, in the case of a key used to protect important persistent data objects such as confidential files, it is a catastrophic if the key is lost by accident. Not only the key itself, but also the important data protected by the key is essentially lost. If the key in the example has been backed up, it is quite simple to recover the lost key and the associated data encrypted with it from the backup data. This is particularly important in the case of businesses, where an employee may become unavailable for various reasons. There must be a key backup mechanism to ensure that no data can be lost as the result of the loss of individual keys. As was shown earlier, the TMD has notions of migratable and non-migratable keys. For non-migratable keys, the TPM itself protects the keys and key backup by the system software is not required. However, system software must back up migratable keys. The backup operation must guarantee that the unprotected keys are not revealed. Such backed-up migratable keys must be protected by encryption from unauthorized accesses.

# 7.5. Encrypted File System

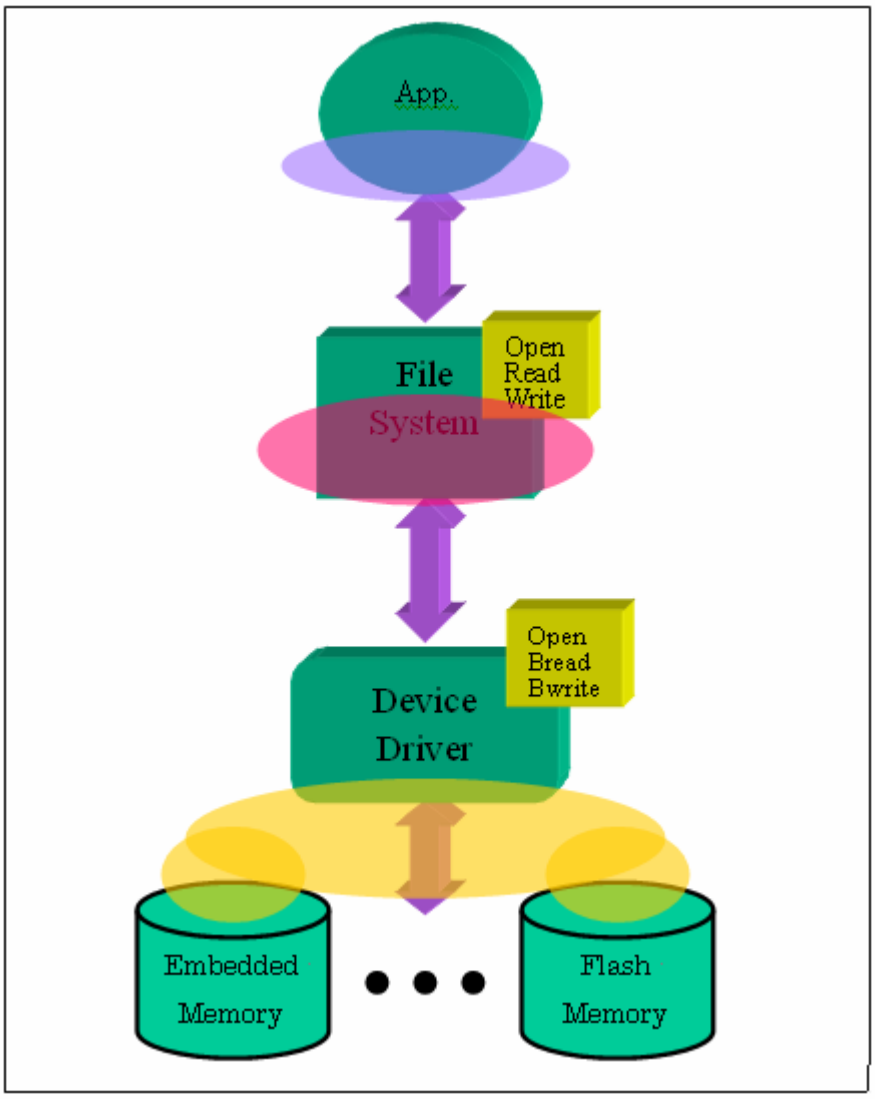


*Figure 7-2 Possible Realization Points of Secure Storage*

## 7.5.1. Implementation Options

There are several strategies for protecting data stored in files through cryptography:

- Application Level Protection
- Filesystem Level Protection
- Device Driver or Device Level Protection

This specification does not mandate which strategy or strategies a conformant implementation must deploy. However, it shall mandate the conformant

implementation to deploy file encryption capabilities, obeying and following characteristics in Security Class 2 and Class 3 devices:

- File encryption must be conducted without any vulnerability windows where plaintext data can be accessed by inappropriate entities.
- At the worst, an encryption key must be provided as a pass-phrase and must not be saved as persistent data in plaintext form. It is recommended that the implementation include a mechanism mandating the selection of quality pass-phrases.
- It is recommended for Security Class 3 level implementations with biometrics devices incorporated that the implementation use biometric authentication as a file encryption key activation mechanism.

It is recommended that the conformant implementation having the capability to obfuscate the names of files so that inappropriate entities cannot deduce any information from names of files.

## 7.5.2. An Example in Encrypted File System Type Implementation

When file encryption mechanisms described above are incorporated in an encrypted file system implementation, the implementation should provide following functionalities for application programs.

### 7.5.2.1. Mount operation

The mount operation provides a means for making a portion of disk drives visible to application software. Since an encrypted file system can transparently encrypt and decrypt files in the system using symmetric cipher, the mount operation requires the key used for encryption and decryption. Although encrypted file systems for ordinary PCs or workstations tend to use user-supplied passwords or pass-phrases to yield the symmetric key, it might not be desirable for the mobile devices since the input interface is not suitable for keying sufficiently complicated pass-phrases. Consequently, conformant implementations should use a symmetric key for encryption and decryption. This key is protected by symmetric or asymmetric key encryption key and the key to activate the session key is supplied in the mount operation. The key encryption key is best stored in the secure storage, so it should be part of the TPM key hierarchy.

Conformant implementations should use 128bit AES keys for the symmetric algorithm and 1024bit RSA keys for asymmetric algorithm.

#### 7.5.2.2. Unmount Operation

The unmount operation removes the access to a portion of the disk drives through a file interface provided by OS's. This operation is almost same as that of normal file system, except that all buffers associated with files belonging to the encrypted file system must be erased before the unmount operation has completed.

#### 7.5.2.3. Read and Write Operation

Read and write operations are almost same as that of normal file system except for the transparent encryption/decryption operations. The file system must decrypt the contents of files on read, and encrypt the data on write with the symmetric key supplied in the mount operation.

#### 7.5.2.4. File/Directory Creation

When a file or a directory is created in the encrypted file system, it is preferable to make the name of the file/directory unreadable while the disk drives holding the file/directory while being unmounted. This can be done, for example, by encrypting the file/directory name with the symmetric key used for contents encryption/decryption.

Reference:

[1] "Cryptographic Memory System", to be appeared in Usenix security 2004.

# 8. Cryptographic API

## 8.1. Overview

In general, a Cryptographic API (CAPI) is an application programming interface for programmers who use crypto related functionalities such as encryption, decryption, one-way hash calculation, digital signature creation and so forth, in their programs. Just like other APIs, the CAPI should supply those functions through well-defined and documented interfaces.

The method of integrating cryptographic functionality into the targeted implementations requires that the developer tightly couples it to the associated cryptographic module. There are several different CAPIs are available in today's security aware market. This section does not go into detail and nor recommend any particular CAPIs. The implementer should consider a certain set of criteria when choosing the CAPI integration. An example of such criteria is the one proposed by the U.S. National Security Agency in evaluating the existing CAPIs **[NSA].** The criteria used by the evaluation are as follows:

1. Algorithm Independence
2. Application Independence
3. Crypto-module Independence
4. Degree of Cryptographic Awareness
5. Modular Design and Auxiliary Services
6. Legacy Support
7. Safe Programming
8. Security Perimeter.

The remaining part of this section describes higher levels of requirements that should be satisfied by CAPIs for the Trusted Mobile Platform implementation. It is the implementer's responsibility how to break the requirements down to actual implementation level.

Just as cryptography plays an important role in modern security technologies, so does

in the Trusted Mobile Platform technology. A Trusted Mobile Platform heavily relies on the fact that cryptographic functionalities can be properly supplied in timely fashion in order for it to perform its required functionalities. For example, a Trusted Mobile Platform with mobile class types 2 and 3; calculates the one-way hash of every executive module such as boot loader, OS, installed peripheral devices, etc., and compares it with the TPM-stored value known to be legitimate, before the module is allowed to be executed. At the same time, information about the executive module is used to update a PCR value by one-way hashing the current PCR value concatenated with the information about the executive in order to reflect the status where the platform is in. If such one-way hash calculation is not done properly, the result can be catastrophic. Same can be said for other cryptographic operations. These cryptographic functionalities are, in some sense, the fundamental for the Trusted Mobile Platform.

Developers must undertake extreme contemplation when implementing a CAPI based solution. The slightest conceivable errors or security holes may possibly lead to the malfunctioning or the exploitation of the trusted platform. Mainly, when interfacing with the TPM, make sure the implementation method is solid and secure. Most of the vulnerabilities exist at the source code level through exploitable buffer methods and or DMA calls.

Consideration points that the implementer should keep in mind for CAPI implementation include, but are not limited to:

- Never breaching any confidential data such as decryption keys to improper entities
- Wiping out memory region before returning it; IF the library uses dynamic memory allocation mechanisms and IF it also uses such dynamically allocated region to hold confidential data.
- Effectively mitigating side channel type attacks such as timing attack, power analysis attack etc.

When implementing a CAPI implementation the implementer should keep in mind the quality of implementation. It is quite (or maybe rather) difficult for most programmers to implement an efficient application without any security holes. As mentioned before, the majority of security holes reside at the source code level. Even the slightest errors

when implementing certain APIs may lead to vulnerabilities.

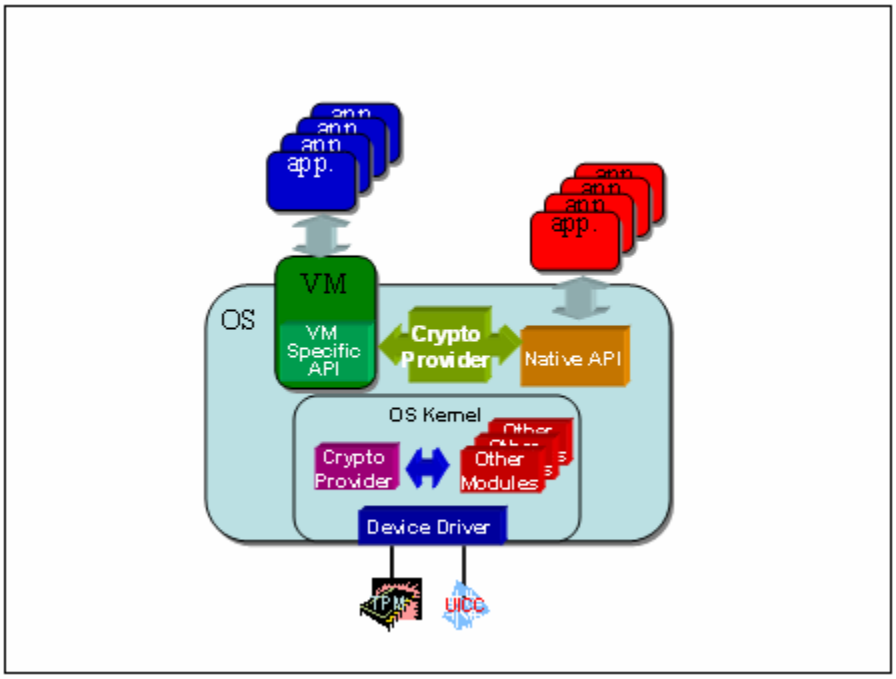


*Figure 8-1 Layers and Providers for Crypto-related functionalities*

There are several levels of cryptographic functionalities the Trusted Mobile Platform should provide. First, application program can conduct cryptographic operations through the standard library interfaces provided by the OS. Second, certain application environments (e.g., VMs), such as Java, provide programming interfaces for cryptographic operation. And third, there should be a kernel level crypto provider that can provide crypto-related functionalities to other kernel modules on request.

There are also several possibilities with regards to who supplies such functionalities:

- Mobile platform's processor
- TPM's cryptographic functionality
- Smart card modules. (UICC)

Of course, there are merits and shortcomings for each of them, so it is important to select appropriate computational resources suitable for the situations.

### 8.1.1. Summary of Cryptographic API and Security Level

| Security Level | Security Class 1 | Security Class 2 | Security Class 3 |
|---|---|---|---|
| TPM Usage for Crypto | None | Possibly used for offloading main processor | Possibly used for accelerating Crypto-related operations |
| Decryption keys in memory | No protection | Keys should be protected through the use of cryptography | Keys should be protected through the use of domain separation and cryptography |
| User authentication | PIN | Pass-phrase<br>PIN protected digital signature | PIN protected biometrics |

*Table 8-1 Cryptographic API with each Security Level*

## 8.2. Required and Recommended Functionalities

This section describes the necessary functionalities for Trusted Mobile Devices in terms of types of functions and security levels.

### 8.2.1. Symmetric Cipher

#### 8.2.1.1. Algorithms and Modes of operation

Conformant Cryptographic APIs must implement symmetric encryption/decryption functionalities including following algorithms and modes of operation.

1. AES with 128 bit key
   A) ECB mode (optional)
   B) CBC mode (required)
   C) CFB mode (optional)
   D) OFB mode (optional)
2. 2-key Triple DES
   A) ECB mode (optional)

B) CBC mode (recommended)

This specification does not inhibit support for cryptographic algorithms other than those listed above. This specification does not mandate any type of stream cipher as an implementation option. However, it is strongly recommended that conformant Cryptographic APIs should contemplate criteria for their choice about algorithms. Possible criteria are listed below.

**Algorithms:** Chosen algorithms should have been exposed to public peer review. An implementation may use algorithms without such review, (e.g. private algorithms that are not published) if it clearly states this in its documentation.

**Key's entropy:** It is desirable to use algorithms that can support key lengths of at least 128 bits. If a Cryptographic API provides weaker keys, it should be noted clearly in its documentation

**Stream ciphers:** If a Cryptographic API introduces stream ciphers, it should make certain that weaknesses inherent to stream cipher are effectively mitigated. Possible weaknesses include:

- Using the same key more than once.
- The possibility of modifying plain text in a controlled manner by exclusive or-ing of artificial bit strings to the cipher text.

#### 8.2.1.2. Key Generation

For key generation in symmetric ciphers, it is required that the implementation uses a cryptographically strong pseudo/true random number generation algorithm.

### 8.2.2. Asymmetric Cipher

#### 8.2.2.1. Algorithms and constructions

Conformant Cryptographic APIs implement asymmetric encryption/decryption functions including the following algorithms and constructions.

1. RSA encryption and signature

A) RSA-OAEP for encryption (a la PKCS#1 v2.1) (recommended) [PKCS]
B) RSA-PSS for signature (a la PKCS#1 v2.1) (recommended) [PKCS]
C) PKCS#1 v1.5 (required) [PKCS]

– Mainly used for backward compatibility with legacy implementations. Necessary for mitigating Bleichenbacher attack by using all the predetermined byte values for sanity check

2. DH key agreement (recommended) [DH]
3. ECDH key agreement (optional) [ECDH]
4. DSA signature (recommended) [DSA]
5. ECDSA signature (optional) [ECDSA]

Conformant Cryptographic APIs may implement asymmetric algorithms other than the ones listed above with discretion. The documentation should clearly state the fact and describes the rationale for the selection.

Note: Support for PKCS#1 v1.5 is required because it is mandatory for TPM [TCGMain]. If the TPM supports RSA-PSS and/or RSA-OAEP, the platform must support RSA-PSS and/or RSA-OAEP. It is recommended to support RSA-PSS and RSA-OAEP because they are considered to be more secure than PKCS#1 v1.5.

#### 8.2.2.2. Key length

Conformant Cryptographic APIs must support the following key lengths for each algorithm as defined below.

- Algorithms based on the hardness of factoring must support 1024 and 2048 bit keys.
- For algorithms based on the hardness of discrete logarithms over finite multiplicative group, (e.g. DH and DSA) key lengths of 1024 bit must be supported. Moreover, in the case of DH it is required that key length 2048 bit is supported.
- For the algorithms based on the hardness of discrete logarithm over group of elliptic curves over prime fields, (i.e. ECDH and ECDSA) 160 bit key lengths must be supported

These key lengths are selected so that they are not likely to be broken by cryptanalysis for at least for several years. However, there always potential threats that may result from technological breakthroughs, such as invention of practical quantum computers.

When the occurrence of such an event becomes likely with high probability, these criteria should be reconsidered. Also, these criteria should be reconsidered periodically as state of the art cryptanalysis technology improves.

Restrictions or limitations on importing, exporting and using cryptography remain an issue in some parts of the world. Depending on the locale and region, it may prove to be a difficult to build, sell and or use the TMD. Please reference your region's cryptographic export regulations/law (for example http://www.bxa.doc.gov/ for US export regulations/law). Under the assumption that the provider for Cryptographic API wishes to provide equipments conformant to such cryptographic restrictions and or regulations; it is acceptable to use a weaker form of cryptographic functions. The implementer will need to keep in mind that this will make the platform much weaker cryptographically. However the fact should be stated clearly in its documentation for such cases.

#### 8.2.2.3. Key generation

For key generation in asymmetric ciphers, it is required that the implementation use cryptographically strong pseudo random number generation algorithm. Moreover, it is recommended that RSA key pair generation meet the following criteria:

- Prime numbers P and Q have same bit length. (e.g., 512 bit in case of 1024 bit-length key.)
- The values of P and Q are not close to each other.
- Both P and Q are such that either P-1 or Q-1 does not have too many factors.

#### 8.2.2.4. Optimizations

Conformant Cryptographic APIs should incorporate common optimization techniques for multi precision integer arithmetic operations such as:

- Power optimization for most algorithms
- Optimization based on CRT, especially for RSA private key operation

Optimization is necessary because a TMD tends to use less powerful processors relative to ordinary PCs to conserve power. Moreover, it is strongly recommended to implement countermeasures that mitigate side channel attacks such as a 'timing attack', wherever possible. One possible example of such countermeasures is blinding

for powering operation. Since blinding makes overall performance worse in some degree, but effectively protects them from 'timing attack'. It is worthwhile to consider incorporating such mechanisms into one's Cryptographic API implementation.

### 8.2.3. One-way Hash Function and MAC

Conformant Cryptographic APIs must implement the SHA-1 hash algorithm and HMAC algorithm using SHA-1 and at least a 128 bit long shared secret. Cryptographic APIs should implement MD5 hash algorithm and HMAC algorithm with MD5 for backward compatibility to legacy implementations. However some conformant cryptographic APIs may implement hash algorithms and MAC algorithms other than those listed above, though documentation should clearly states that such algorithms may lack sufficient peer review and might be subject to attack.

### 8.2.4. Pseudo Random Number Generator (PRNG)

Conformant Cryptographic APIs must provide a cryptographically strong pseudo random bit stream generation facility. It is recommended that the PRNG uses truly random data, e.g. thermal noise, as the source for the pseudo random number generation function. In a Security Class 2 or Class 3 Trusted Mobile Device, the TPM's random number generation function may be used as the random number source.

### 8.2.5. Higher Level APIs

In addition to the basic APIs listed in the preceding sections, it is desirable for users if the API provides not only basic functionalities, but also APIs for higher level functions. It is possible to build complex operations based on primitive functions, but this approach is cumbersome for application programmers and might be error-prone. It is recommended that Conformant Cryptographic APIs provide those higher level APIs for application programmers. Examples of such higher level APIs are:

1. APIs for handling SSL/TLS functionalities
   - Opening SSL-protected session
     - Algorithms negotiation
     - Sending, receiving, and verifying digital certificates
     - Exchanging shared secrets
   - Raising a normal session to a SSL-protected one by means such as sending STARTTLS command in ESMTP or using upgrade mechanism

in HTTP/1.1

2. APIs for Managing digital certificates
   - User enrolment to CA through certain protocols
     - Certificate Signing Request (PKCS#10)
     - Certificate Management Protocol (CMP)
     - Simple Certificate Enrolment Protocol (SCEP)
   - Retrieving own/other's certificates
   - Renewing certificates
   - Revoking certificates
3. APIs for secure messaging
4. APIs for IPSec
5. APIs for Web Service Security

## 8.3. Language Bindings

Conformant CAPIs must provide its associated functionalities listed above by utilizing software development languages like C and or Java. It is also recommended such implementations should also provide C++ API. Conformant CAPIs may provide other languages bindings at the implementers' own discretion.

## 8.4. Hardware Support

In the case of Security Class 2 or Class 3 implementations, it is expected that TPM incorporated into the platform provide several cryptographic capabilities with hardware level. UICC is another candidate for providing cryptographic functionalities to the platform. In such cases, it is desirable for a CAPI to provide a means to take advantage of such hardware-based capabilities for cryptographic operations. However, the actual realization mechanism is left to the implementers. Possible implementation methods are:

- To provide a discrete function name for each hardware-driven cryptographic functions.
- To provide an abstract layer, e.g. Microsoft's Cryptographic Service Provider (CSP), this can be dynamically selected during execution.
- To provide functionalities through device driver abstraction, which are adopted by OpenBSD

However, there should be enough contemplation before trying to use TPM or UICC as a kind of off loader for main processor or accelerator for cryptographic operations. The discrete TPM tends to be connected to memory or other resources in the platform with narrower bus than that of the main processor and UICC may either be the same or in a worse situation. Moreover, pure performance of the processor in TPM or UICC itself tends to be far less powerful than that of the main processor. However, hardware support can include an embedded TPM module. The embedded TPM will not have the bus interface problems noted for the discrete TPM, and will likely have dedicated hardware that will allow it to execute algorithms comparable to the CPU. In addition, the overall security of the platform can be enhanced if the security processing is conducted inside a secure boundary where keys are not exposed. So it is quite likely that the overall performance of the platform would be better when all the crypto-related operations are done only with the main processor on the platform.

References:

[NSA] "Cryptographic API Recommendation", National Security Agency (NSA Cross Organization CAPI Team) - June 12, 1995

# 9. TPM Support Software (TSS)

## 9.1. Overview

TSS is a software stack that allows applications to communicate with the Trusted Platform Module (TPM) in the TMD. It provides an API that exposes the functionality provided by the TPM, namely Authentication, Authorization, Protected Storage and Attestation. It also provides an interface for utilizing the TPM as a hardware-based cryptographic service provider. This allows applications to use the TPM to generate keys, encrypt/decrypt, sign etc. The TSS provides synchronized access for applications consuming the services of the TPM. Additionally, the TSS manages TPM resources. This section is based on the TSS specification developed by the [5] working group.

## 9.2. TSS Requirements

- Should provide applications a common standard interface independent of the TPM's hardware implementation.
- Should provide an interface for applications to use the TPM for cryptographic operations, sealed storage and attestation. In addition, the TSS should provide support for TPM ownership/administrative functions.
- Should not be possible to invoke TPM trust services by bypassing the TSS.
- Should manage the TPM hardware resources in a manner transparent to the applications.
- Should provide logging/auditing services for TPM events.
- In addition to the TPM-based cryptographic support, the TSS should also provide means for integrating industry standard cryptographic service providers.

## 9.3. The TPM

The Trusted Platform Module (TPM) is a secure hardware subsystem as defined by the Trusted Computing Group (TCG), formerly Trusted Computing Platform Alliance (TCPA). The TPM is permanently attached to the platform and serves as a

hardware-based root of trust. The TPM h/w provides cryptographic functionality like – RNG, hashing, HMAC, Asymmetric Key Generation, Asymmetric Encryption/Decryption etc. It comprises of 160-bit Platform Configuration Registers (PCRs) to hold the results of a SHA-1 operation. The TPM has at least 8 such registers and are used to record the results of software integrity measurement during trusted boot.

### 9.3.1. Authorization Protocols

The authorization protocols are used to authorize the use of TPM or authenticate the entity using the TPM. They are based on the rolling nonce model. The two main protocols are Object Independent Authorization Protocol (OIAP) and Object Specific Authorization Protocol (OSAP). OIAP allows authorization of any TPM object after acceptable proof of authorization for each TPM object is presented. OSAP allows an authorization session to be established only with a specific TPM object. Multiple commands can be sent to the TPM as long as they involve a specific TPM object and the proof of authorization has to be presented only once.

### 9.3.2. Protected Storage

A TPM has limited amount of non-volatile storage used for storing TPM unique keys, a storage root key (SRK), ownership related data, flags for configuring the TPM etc. The TPM uses the storage root key to encrypt the rest of the keys and data which can then be stored in bulk storage. In addition, the TPM provides the capability to seal keys or data to the software integrity measurement value stored in the PCRs. Sealed data can be unsealed only if the platform is in the specified state. For example, a secret can be sealed to a specific version of an operating system. That secret is released or unsealed only when the platform has booted the specified operating system and it has not been tampered.

### 9.3.3. Attestation

Attestation is a process by which a platform reports its software environment to a challenger seeking to evaluate its trustworthiness. When a platform receives such a request, the TPM signs the integrity metrics recorded in the PCR during trusted boot and reports it back to the challenger. The TPM uses an Attestation Identity Key (AIK) to sign the hash values stored in the PCRs. The Certificate Authority (CA) issues the

AIK. Upon receiving the attestation record from the platform, the challenger verifies the signature, thus authenticating the platform. It then verifies the reported integrity data and makes a trust decision. Thus, attestation allows an entity to determine if the software environment in a platform has been compromised before engaging in any further transactions.

## 9.4. TSS Stack

As shown in Figure 9-1 TSS Stack, the TSS stack is comprised of TPM device drivers, TSS core services (TCS) and the TSS service provider (TSP). The TPM device driver is a hardware specific driver typically provided by the TPM vendor. The device driver library interface provides a common standard TPM interface for all the applications. TSS Core Services (TCS) provides the infrastructure required to manage keys and credentials, serialization of commands to the TPM, and context, event & audit management. Applications communicate to the TCS using the TSS Service Provider (TSP) layer. The TSP is the topmost component in the stack and provides TPM services for applications.

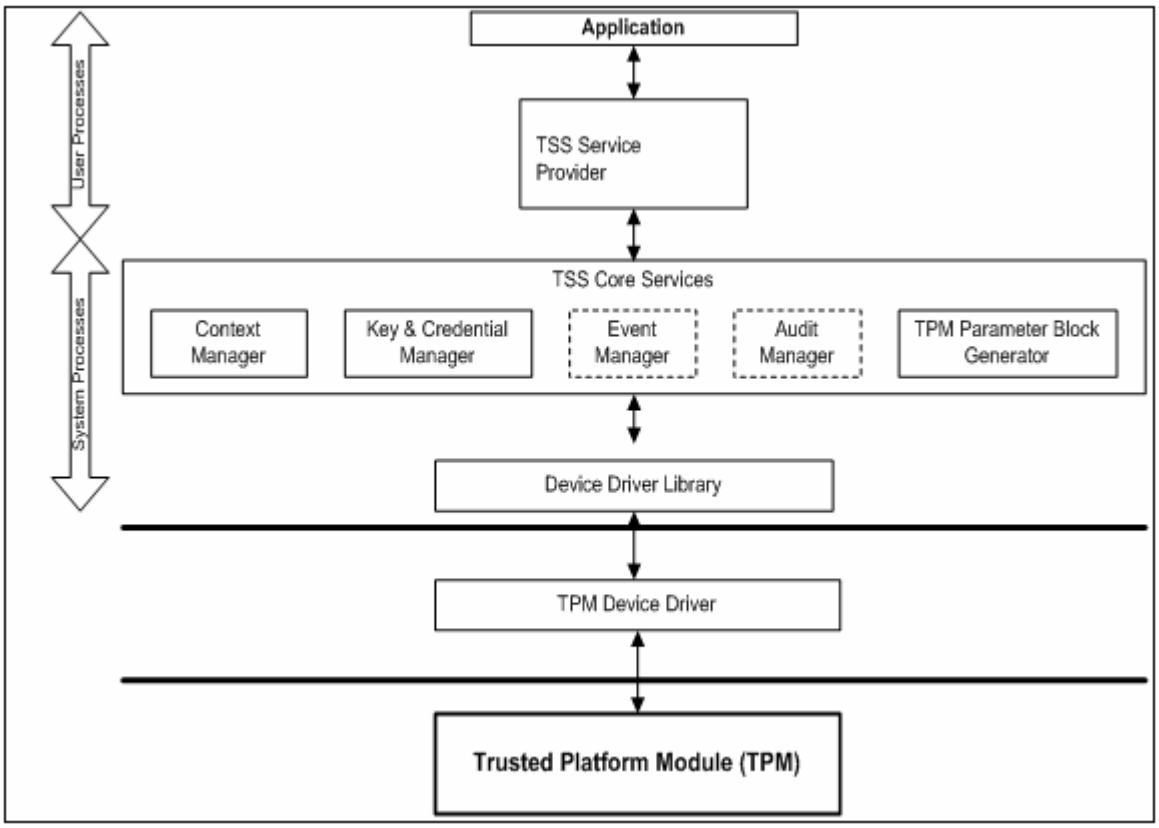


*Figure 9-1 TSS Stack*

### 9.4.1. TSP

The TSS Service Provider (TSP) module provides an API to applications using the TPM. The TSP hides the management of TSS related authorization sessions from the calling application by initiating required authorization sessions and handling all internal data for the session. The availability of existing general purpose Cryptographic Infrastructure (e.g. CDSA, MS-CAPI, and PKCS #11) can be integrated into the TSP, hence providing applications with services, which might not have otherwise been available from the TSS. The Cryptographic Service Provider (CSP) is integrated within the TSP, therefore applications can choose to use either the cryptographic service provided by the TPM or use the service available from the integrated Cryptographic Service Provider.

### 9.4.2. TCS Components

The TCS provides a common set of services required for managing the TPM in a TMD.

There is typically one TCS per platform. It communicates with the TPM via the device driver library.

#### 9.4.2.1. Key and Credential Manager

This TCS module manages the user and platform keys and credentials. The key manager facilitates the definition and management of a flexible persistent key hierarchy, which allows different key hierarchy structures (e.g. deep key hierarchy, shallow key hierarchy, etc.) to be managed. The credential manager facilitates access to the different platform credentials stored in the system.

#### 9.4.2.2. TCS Key Manager

The TCS Key manager provides support for defining a key hierarchy rooted in the storage root key (SRK) in the TPM. All keys are registered in persistent database and assigned a UUID. The keys are referenced and retrieved using the UUID.

#### 9.4.2.3. TCS Key Cache

The TCS key cache provides a mechanism for managing the limited physical storage in the TPM. The key cache provides virtual storage for keys so that applications don't have to be concerned about storage capacity of the TPM. The key cache is responsible for loading and unloading the appropriate keys in the TPM, thus ensuring that correct key is in the TPM when the application is ready to use it.

#### 9.4.2.4. TCS Credential Manager

The TCS Credential Manager manages the different platform credentials associated with the TMD. TCS might mandate the use of access control for applications to retrieve these credentials, hence alleviating privacy concerns.

#### 9.4.2.5. TPM Parameter Block Generator

Since the TPM is a serial device, there has to be functionality in the TSS to multiplex simultaneous requests for resources from different services/applications to the TPM. The TPM Parameter Block Generator is responsible for this functionality, and does so by serializing, synchronizing, and processing TPM commands. As part of its

serialization, it will build byte streams for input to the TPM, and process byte stream output from the TPM.

#### 9.4.2.6. Context Manager

The Context Manager in the TCS is responsible for assigning dynamic handles that allow for efficient usage of both the application and the TCS's resources. Hence, each resource that a calling application can work with will be assigned to a certain context represented by a handle within the Context Manager. Each handle will provide context for a set of interrelated TPM operations. Different threads within the application may share the same context or may acquire separate contexts per application.

The Context Manager is also useful for allocating memory in the TCS, which will be provided to the calling application (either external or internal to the TCS) on demand and will be assigned a certain context within the Context Manager.

#### 9.4.2.7. Event Manager

This component of the TCS manages PCR events associated with respective PCRs. The hash value recorded in a PCR is not by itself very meaningful. The TCS event manager maintains an event log that describes the event that was measured, the hash value and the order in which the events occurred. The event manager allows access to a challenger so that the cumulative hash value reported by the TPM via attestation can be interpreted.

#### 9.4.2.8. Audit Manager

The TPM generates an audit event in response to the TPM executing a function that has the audit flag set to TRUE for that function. The TPM also maintains the details for the last audit event in a log, and keeps track of all events that occur after TPM initialization. There are two portions of the log, an internal value kept by the TPM, and the external log of values that reflect the internal state. The TCS Audit Manager will be responsible for supporting this functionality of the TPM. Also, the Audit manager will support TPM mechanisms for re-synchronizing the internal and external logs.

# 10. User Authentication

User authentication is a process to establish the validity of a user and if they are who they claim to be. Typically the method is via some challenge-response request that the user must satisfy: if the user is the individual they claim to be, the user must then enter their password. Not all authentication methods are of this type, there are also hardware based authentication implementations (such as the use of pin-code and biometric devices), with suitable modules, which may be substituted seamlessly for more standard approaches for user authentication on the Trusted Mobile Device.

Trust in a mobile device is established at several layers. In order to establish trust between the mobile device, network operator and other trusted third parties, proper user authentication and identification mechanism is necessary. Efficient use of the security policy is necessary in identifying a trusted source. Implementation of user authentication on the Trusted Mobile Device allows proper authentication between the user and the service. The service operator also has the obligation to identify that it is a trusted source. The process of establishing trust between the TMD and the service provider is defined in *Section 8.3.4 – Mutual Authentication* of [9].

This section will discuss how a user can be authenticated on a Trusted Mobile Device. The following subsections will discuss several hardware and software based authentications methods.

| | |
|---|---|
| *"What you know"* | Password and pass-phrases |
| *"What you have"* | Tokens: physical keys and USIM |
| *"What you are"* | Static biometrics: fingerprint, iris, face, etc |
| *"What you do"* | Dynamic biometrics: voice, signature, etc. |

*Table 10-1 The four principal forms of Authentication are:*

These four principle methods of user authentication and authorization are the cornerstones of any well-implemented security policy. Since simple passwords are rarely sufficiently strong, developing and designing a comprehensive strategy that includes strong user authentication methods is imperative.

.

## 10.1. Identifying the User

Attestation plays a key role in user authentication on the TMD. The process involves an Attestation Identity Key (AIK), which proves the identity of the platform and the user while protecting the user privacy. Only the TMD owner can create an AIK by generating a RSA key pair and requesting an identity certificate from a trusted third party called the Privacy CA. The Privacy CA verifies the request, generates an identity certificate and returns it after it encrypt the message by the public key of the Endorsement Key of the requesting platform, so the only the genuine TMD can decrypts the response. A user needs to present the authorization secret of the AIK (which is presumably passed from the TMD owner to a user, if they are different individuals) to use the AIK to conduct attestation. The TMD must provide a mechanism to authenticate the user against TPM before executing operations that use AIKs. The communication protocols for requesting AIKs and exchanging attestation are defined in Section 5 of [9].

## 10.2. Identifying the Owner

The TMD owner needs to declare the ownership of the platform by the TPM_TakeOwnership operation before using features of the TPM. The operation creates the Storage Root Key (SRK), and returns the owner authorization secret as well as the SRK authorization secret. The TMD owner must be properly authenticated before owner specific TPM operations are conducted. The TMD owner or user must be authenticated against the SRK authorization secret before accessing the protected storage.

## 10.3. Authentication module API

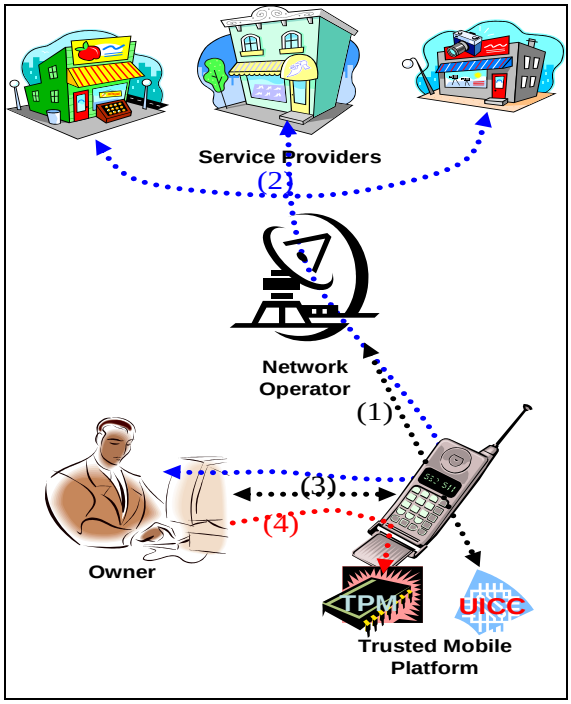


*Figure 10-1 Possible authenticating parties*

Authentication of the Trusted Mobile Device involves several entities as shown in 10-1 (Possible Authenticating Parties). Possible authenticating pairs are as follows:

- Between the network operator and subscriber
- Between the service provider and consumer
- Between the user and the equipment
- Between the owner and TPM

As shown in Figure 10-1 (Possible Authenticating Parties), one authentication method between a network operator and a subscriber is done via the USIM, an application stored within the UICC, and the network operator while establishing network connection. This mechanism is defined by network architectures such as 3G and is independent of the current Trusted Mobile Platform specification. Therefore this document leaves this first method uncommented. For more information regarding case two, please reference the Protocol Specification Document [9].

## 10.4. Authentication between owner and TPM

Conformant Cryptographic APIs must implement authentication functionalities between the owner and the TPM if the platform supports TPM. The TPM supports two challenge-and-response-based authentication methods called OIAP (Object Independent Authorization Protocol) and OSAP (Object Specific Authorization Protocol) [TCG]. Both of these protocols must be supported.

## 10.5. PIN-Code

PIN code entry is probably the easiest way to authorize the user to their mobile device. PIN codes rely on user memorization of at least a four to seven digit PIN code. The PIN code entry can be registered at the time of mobile device provisioning to the user, and it should be changed every 4-6 months. There is a security risk associated with PINs since the PIN code can be cracked through brute force attacks. It is recommended that if the manufacturer uses PIN code for user authentication, then it should be used in combination with another stronger form of user authentication, such as biometrics or a pass phase.

In the process of registering and/or changing the PIN code, the PIN will be registered with the TPM and/or the device's secure operating system. The AIK will not be effected in anyway since it is only used to reference the mobile unit's locality identification. The network operator and the TPM will properly register the new PIN code and/or future changes to a pin code. It is recommended that a secure utility application be provided for client management to allow changes to the user PIN. Changes to the PIN must be updated with the network operator.

## 10.6. Biometrics

The ever-growing use in using biometric based technology as a security method has increased in recent years. As mentioned and described in the Trusted Mobile Platform Hardware Architecture Document (HWAD), several industries have recently stepped up efforts in increasing the security of password based identity and migrating to biometric solutions. Methods of biometric use-cases have introduced physical access

control, spanning to small mobile devices. A well designed biometric implementation is fundamental in maintaining a trustworthy computing base. Biometric systems must ensure the following basic attributes to maintain a trustworthy session:

- Biometric information must come from a "live" person at the time of user authentication and verification.
- Biometric information matches the master biometric data on file.

The integrator must understand the user point of view and always establish trust, that he or she is using an "authentic" and trusted biometric system. The integrator must also assess that in many real-life applications, a user may want to retain privacy when accessing a service with biometric authentication.

Based aforementioned integration issues above, this section will also discuss possible threats, requirements and provide a detail solution for both biometric system integrity and biometric data protection. This solution makes use of an extension of [5] technology; and ensures that an unauthorized entity is not able to access sensitive information during biometric authentication.

### 10.6.1. Existing Biometric Authentication Technology

Today the existing biometric technologies have two procedures that are widely used in identification verification by biometric systems:

- **Enrolment**. A method that the biometric unit captures biometric code (BC) (an individual's biometric information) as a sort of registration template.
- **Matching**. Biometric Data (BD) (recently captured biometric information) is compared to the BC, to decide whether or not it matches.

Biometric implementation techniques vary depending on the vendor. The BC and BD will vary from fingerprint, and hand geometry to voice, retina, face and behavioural characteristics according to the biometric techniques used by the vendor. [DaFrMa98], [IEEE00], [JaRoPr98], [Rat99] and [Way98].

Since this section does not focus on any particular biometric technique, it only will

discuss the protection of biometric information and check integrity of the whole system, based on a typical user authentication model. As shown in Figure 10-2 (Biometric enrollment and authentication process) shows an example of letting a valid user access a computing platform, by involving three entities: Smart Card (SC) / SIM – holding the user BC, a Biometric Reader (BR) collecting the user's BD and the accessed computing platform running the matching process. This model is different from the existing biometrics with smart card technology, such as [BoRe95] and [Sei86], because it combines user authentication with integrity checking of the platform and Biometric Reader.

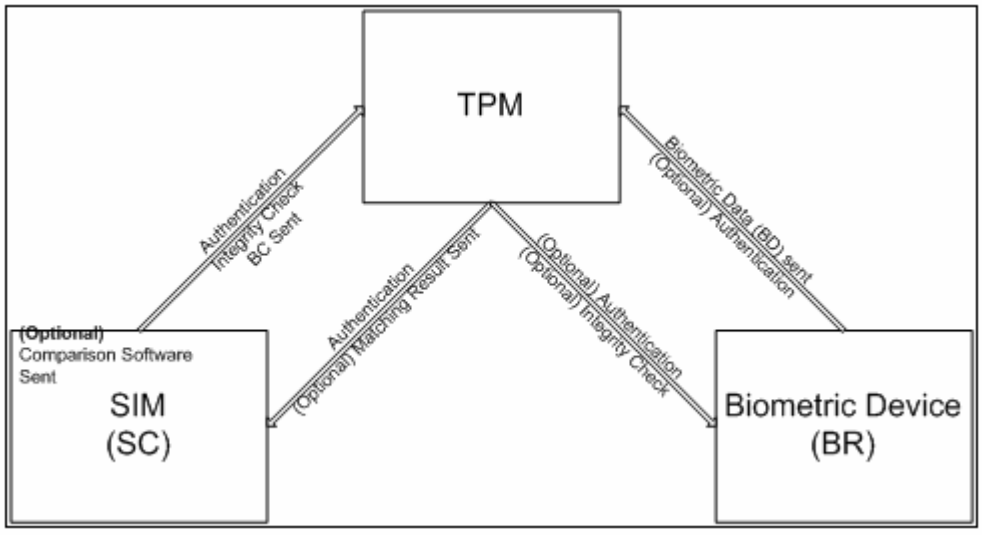


*Figure 10-2 Biometric enrollment and authentication process*

## 10.6.2. Possible Threats

When implementing a [5] biometric compliant trusted mobile device, the implementer must keep in mind the possible threats regarding biometric data protection and systems integrity. These threats have been addressed in [ChPeVa00]:

1. *Interception of communications between the SC and the Trusted Platform.* If the BC is sent in clear text or protected weakly, an eavesdropper could obtain the BC by listening in on communications between the SC and platform. This is a particular

problem if the SC communicates with the platform over public networks, because the platform is not located locally to the user.

2. *Interception of communication between the Trusted Platform and the BR.* If the BD is sent in clear text or protected weakly with mere scrambling on the line, an eavesdropper could obtain the BD by monitoring communications between the BR and platform. Again this is a particular problem if public networks are involved in the communications.
3. *Malicious BRs.* A malicious BR is able to record the BD of a user, and a malicious BR is able to send a fake BD of a user. Malicious code alters the state of the biometric reader.
4. *Malicious platform.* A malicious platform can obtain both the BC and BD of a user and of course a malicious platform can give a fake result of the user authentication.

### 10.6.3. Trusted Biometric System Solution Example

The proposed solution in this subsection is an example of establishing a trusted relationship among a SIM (SC), a Trusted Biometric Reader (TBR) and the trusted mobile platform. This trusted relationship among the user, BR and the trusted mobile platform demonstrates means of storing a version of the biometric comparison software. The solution allows two options in implementation. The first option allows the storing of biometric comparison software within the TPM. The second option allows a version of the biometric comparison software signed with a trusted third party, with the private keys stored within the platform, and the third party's public key certificate is verifiable to the TPM. Alternatively, the implementer can add extra feature sets of the existing standard TPM by adding an authentication function, (e.g., the ability to verify a digital signature based on its public-key certificate and a biometric matching function) with the option to extend some trusted software that can interoperate with the standard TPM, with such trusted software. Since the nature of the TPM executes upon boot time: the biometric comparison software is checked for integrity with the reference to the signed version and the third-party public key certificate ; and if the integrity check fails, the biometric comparison software would be prevented from loading. If the integrity check fails, it may be arranged that the complete platform integrity check fails.

As shown in Figure 10-2, the transaction between TPM, SC and BR; the BC is stored in

the SIM (SC) and is transferred into the TPM during the authentication procedure. Or another alternate method is to have the BC centralized and stored (for example in the TPM) in order process administration quickly. In the later case, the SC would still be used for integrity checking purpose. Although in certain hostile environments a BR "maybe", potentially untrustworthy and so there is an option for the TPM to require authentication with the BR and additional integrity check at the hardware level. The SC should be able to check the integrity of both the platform and also the BR (via the TPM). The TPM would then be able to perform integrity tests onto the platform, and the stored value of the BR integrity would be apart of the authentication protocol. Upon signing on using the SC, the mutual authentication between the TPM and the SC would check the integrity of the TPM, before proceeding further. This method can be combined with an additional integrity check by the TPM onto the BR. The BC would then be transferred from the SC to the TPM, which is protected via the Trusted I/O or through the use of encryption. Optionally, comparison software is also sent from the SC to the TPM. The TPM authenticates the BR and or authenticates to the TPM and the certificates are exchanged. This method should be left optional for the implementer because the BR may only have integrity check-related information or a serial number and may not have the cryptographic functionality needed for authentication purposes. The SC and BR have no direct communication link in general, therefore it does not authenticate directly. After the user is satisfied with the TMD's trustworthy state, the user would then apply user's finger onto the fingerprint sensor (or equivalent in other biometric methods). The BD is then sent from the BR to the TPM. The method of communications is protected via the Trusted I/O or through the use of encryption. The TPM then makes a comparison between the BC and BD to see whether if the BD matches with the data that is registered with the TPM. The TPM can then report the findings to the user directly via the display, or to the SC, signed using the TPM's private signing key. Reporting these findings to the SC could prove to be useful if it is desired that the SC should release certain secret information only when it has been determined that the user is the valid owner of the SC.

### 10.6.4. Trusted Biometric System Summary

Based on the solution given to the above subsection, biometric authentication can be carried out, as follows:

- Mutual Authentication between the trusted mobile platform and the SIM (SC) and optionally with the BR. The SC would then verify the integrity of the given trusted

mobile device and the biometric reader (BR). The integrity checking process allows and ensures that the TPM has a good record of the trusted mobile device and the biometric reader state (e.g., composition and current environment).

- The SIM (SC) allows the user to see that the TPM authentication and the platform and BR integrity checking has been successful (based on the methods discussed in [BaCHChPePr002]).
- The TPM authenticates the SIM (SC) identity and obtains the biometric code (BC) from the SIM (SC).
- The biometric reader (BR) takes the biometric device (BD) and negotiates with the TPM in a secure manner using a symmetric key, to validate its authenticity.
- The TPM compares the biometric device (BD) and the biometric code (BC) by using a biometric algorithm. Under the assumption that the information does match, according to the previously defined user threshold – the TPM will allow the user to log into the trusted mobile device (TMD) and access the appropriate services. (This user defined threshold is dependent upon the type of biometrics used in authentications and its associated applications involved in the process.)

### 10.6.5. References

[BaChChPePr002] B. Balacheff, D. Chan, L. Chen, S. Pearson, and G. Proudler. Securing smartcard intelligent adjuncts using trusted computing platform technology. In *the Proceedings of IEIF Fourth Smart Card Research and Advanced Application Conference (CARDIS 2000)*, pp 177-195, Bristol, UK, 20-22 September 2000.

[BoRe95] E. Bovelandar and R.L. van Renesse. Smart cards and biometrics: an overview. In *the Proceedings of the 12th World Conference on Computer Security, Audit and Control*, 1995.

[ChPeVa00] L. Chen, S. Pearson, and A. Vamvakas. On Enhancing Biometric Authentication with Data Protection. In *the Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies*, pp 249-252, IEEE,

2000.

[DaFrMa98] G. I. Davida, Y. Frankel and B. J. Matt.
On Enabling Secure Applications Through Off-line Biometric Identification. In *the Proceedings of 1998 IEEE Symposium on Security and Privacy*, pp 148-157, 1998.

[IEEE00] Special Issue on Biometrics, *Computer*, Vol 33, No 2, IEEE, February 2000.

[JaRoPr98] A.K. Jain, A. Ross and S. Prabhakar.
Biometrics-based web access. MSU Technical Report TR98-33, 1998.

[Rat99] N.K. Ratha et al..
A biometrics-based secure authentication system. In *the Proceedings of IEEE Workshop on Automatic Identification Advanced Technologies*, pp 70-73, 1999.

[Sei86] S. Seidman.
Biometrics and smart cards combine to offer high security. *Journal of Nuclear Materials Management*, vol 15, pp 143-145, INMM Annual Meeting, 1986.

[Way98] J.L. Wayman.
A generalized biometric identification system model. In *the Proceedings of the 31st Asilomar Conference on Signals, Systems and Computers*, pp 291-295, 1998.

## 10.7. User Authentication Consideration Points

The use of a fingerprint authentication and a pin code is an excellent authentication model for identifying both the mobile device and user. The TPM would manage the finger print's mathematical code, and bind/associate it with the AIK. This information would be one of two keys needed to access the TMD. The other recommended method along with the use of biometrics is pin code and/or a SIM/USIM. All of the users authenticating information would be encrypted and managed by the TPM within the secure domain. It is recommended that an alternative form of authentication be

provided at all times, allowing the possibility of a fallback method for authentication. There may be damage to the biometric device or the biometric device may not function properly in a hostile environment. In either case, device management should allow the user to specify the preferred level of security.

# 11. Trusted User Interface

Security Class 3 level TMD's have an option to provide a Trusted User Interface (TUI) that enables secure interactions between a user and a device, including the security elements inside of the device (e.g. TPM, Smart Card, or SIM/WIM). The input/output interface between the user and the Trusted Mobile Device should be protected or hidden from interception, observation, and hacking. This section describes the Trusted User Interface from the software architecture point of view. Also please refer to the 'Trusted I/O section in "Trusted Mobile Platform Hardware Architecture Description" [TMP-HWAD], since this discussion is tightly linked to the user interface devices.

The requirements for the user interface of the TMD are as listed below:

- The Trusted mobile device shall be able to display a trusted message on the screen or display (for Security Class 2 and Class 3 devices).
- The Trusted mobile device shall provide security mechanisms that ensure that trusted messages are accurately and completely displayed. Security mechanisms to be employed may include hashing the message to be displayed and/or encryption of the message that will be displayed (for Security Class 3 device).
- The Trusted mobile device shall provide an indicator, referred to as a Trusted Mode Indicator (TMI), activated by a Trusted Mode Manager (TMM) when the displayed message is trusted. The indicator must be activated (turned on) when the trusted mode is entered and deactivated (turned off) when the TMP leaves the trusted mode and enters the un-trusted mode of operation (for Security Class 2 and Class 3 devices).
- The default state of the TMI shall be the deactivated mode (for Security Class 2 and Class 3 devices).
- The Trusted mobile device may provide security mechanisms that ensure that trusted input (i.e. from a keyboard) is received by the TMD without any modification, deletion or addition to the input data stream. Security mechanisms to be employed may include hashing or encryption of the message or data being input (for Security Class 2 and Class 3 devices).

- The Trusted Mobile Device shall preclude the possibility that trusted information being displayed or trusted data from an input device can be copied and replayed to impersonate a valid user or message (for class 1- and class 2 security devices).

The Trusted mobile device has a special output device referred to as Trusted Mode Indicator (TMI). TMI is directly controlled by the Trusted Computing Base (TCB) or the Trusted GUI Manager and securely reports the operating state of the device to the user. Generally, the Trusted mobile device has some sort of bitmap display unit, i.e. a General purpose Display Unit (GDU), and input devices. If the GDU and input devices are used by an application via an un-trusted GUI Manager, it is difficult to confirm the legitimacy of the messages displayed to the user. The TMI solves this problem. The user can easily evaluate the trustworthiness of such messages by means of the TMI. The look and feel of the user interface is also important for trustworthiness. The user interface should use an intuitively recognizable universal ergonomics design since the user of the device will not in general be a computer security specialist.

## 11.1. Trusted Mode Manager and Indicator

The Trusted User Interface (TUI) must indicate the trustworthiness of the user interface. Here we present two example architectures to achieve this requirement. The first architecture is when the TUI is provided by a Trusted GUI system. The second is when the GUI system on the device does not have sufficiently trustworthy features to be a Security Class 2 or Class 3 TMD, and then the TCB must provide a trusted GUI system.

### 11.1.1. With Trusted GUI

If the device supports a Trusted GUI System, an additional indicator which shows the trustworthiness of the device may not be needed since a GUI system has a dedicated and trusted screen region for this purpose (see Fig 11-1). The Trusted Mode Manager (TMM) will also control the mode of the user interface, either trusted or normal. The TMM may be a part of a trusted GUI system or TCB. A trusted GUI System must support following features:

- Trusted region:

The trusted region is managed by the TMM and displays the operating mode of the

device. The application screen can't override this trusted region. In trusted operation mode, the application region is also used by TMM to establish trusted interaction with the user.

- Labeled windows/applications:

Each window has a (domain) label and is displayed in the trusted region for ease of recognition by the user. An icon image in a trusted region may represent the label.

- Trusted Path:

This is an individual interface path managed by the Trusted GUI System, so other applications cannot interfere with this interface.

- Access Control.

A Mandatory Access Control Policy controls the use of the GUI system.

- Integrity.

The Trusted GUI System is a part of the TCB. This must be verified using a [5] Trusted Boot or Code Signing.

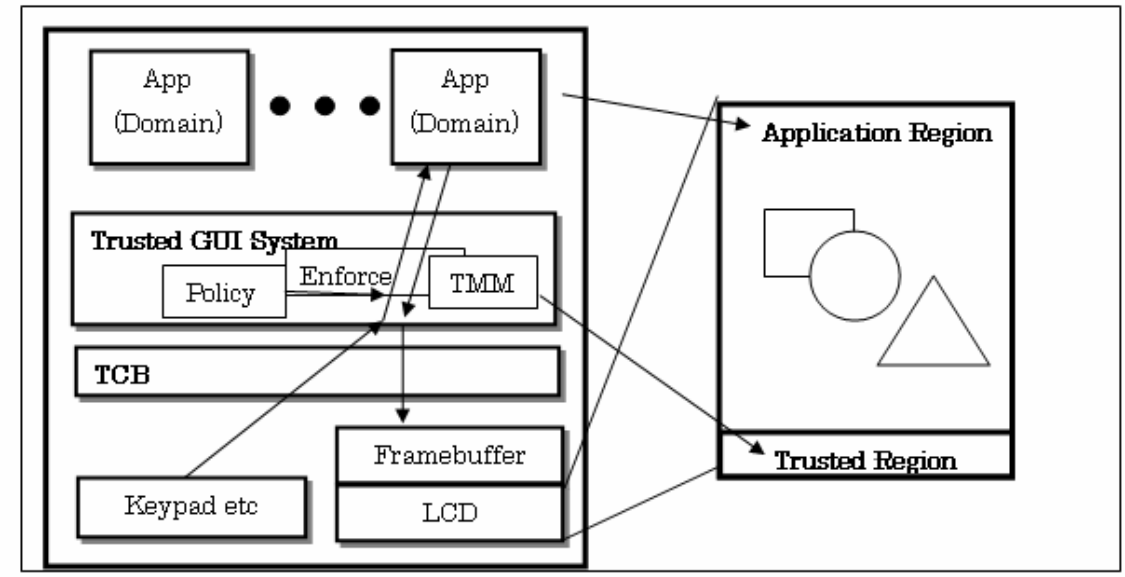


*Figure 11-1Trusted GUI System Architecture and screen example*

## 11.1.2. With Un-trusted GUI

Figure 11-2 shows an implementation example of the Trusted User Interface working with an un-trusted GUI library. The Trusted Mode Manager (TMM) within the TCB boundary manages the modes of the device operation, Trusted Mode or Normal Operating Mode. In normal mode, the un-trusted GUI library manages the frame buffer.

If a malicious application provides a false user interface, it is difficult for a user to detect.

The TMM and TMI eliminate this kind of threat (Figure 11-3). If an application wants to start a trusted communication with a user, the application sends a request to the TMM. Then the TMM activates its own trusted GUI on the GDU in place of the un-trusted GUI library, and at the same time the TMI indicates the Trusted Mode state. The application supplies messages to be displayed and requests actions from the user, such as password entry, confirmations, and selections. Additionally, the TMM checks the domain information and reports on the legitimacy of the requested application to the user. The TMM also supports trusted connections between remote services. A later section provides a further description of these connections.

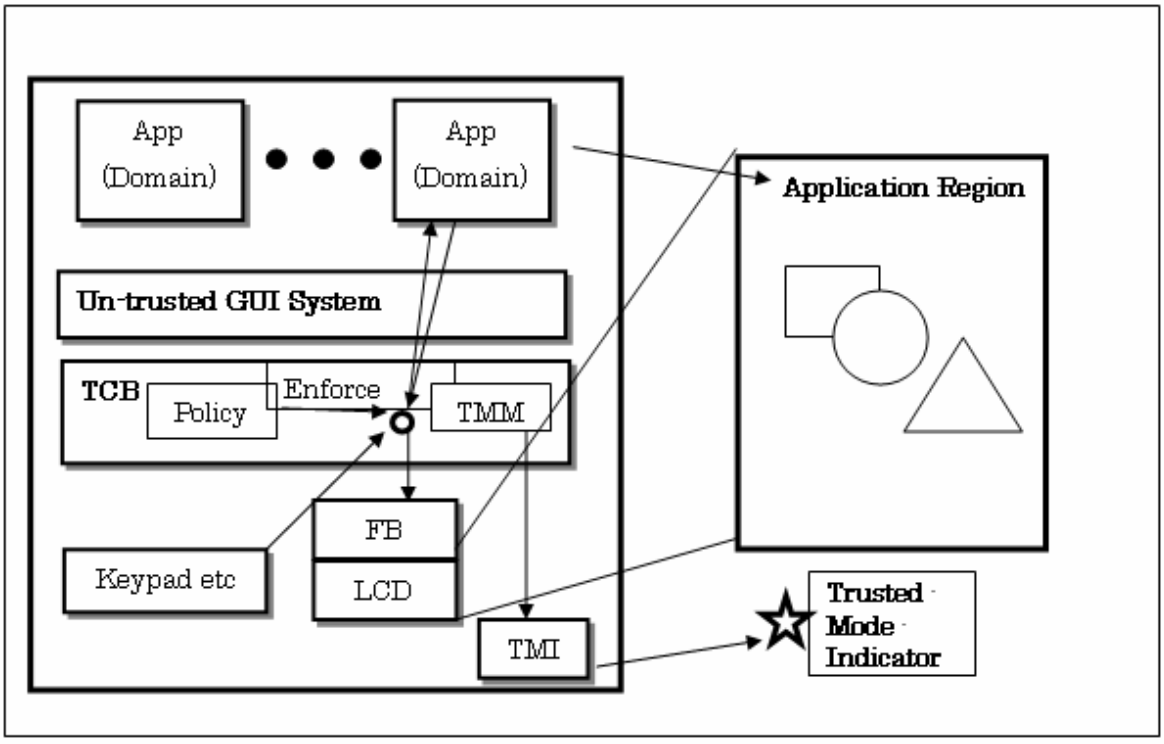


*Figure 11-2 Architecture of the Trusted User Interface with Un-trusted GUI System*

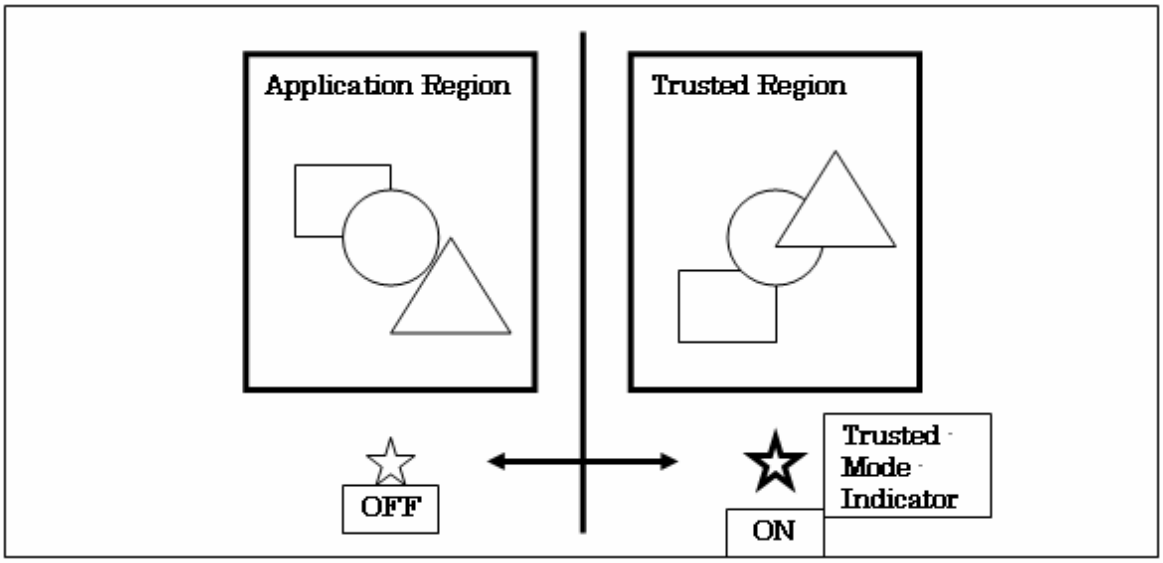


*Figure 11-3 Behavior of the Trusted Mode Indicator*

## 11.2. Operation of Trusted User Interface

This subsection describes the operations supported by the TUI.

- User Verification (e.g., PIN entry, PIN management)
  - TPM Authorization. The Authorization Secret associated with a TPM key is used for authentication between a user or service entity and the TPM, as when a user logs into the domain using a Trusted Mobile Device. To achieve this, the TMM supports the OIAP and OSAP protocols to create secure connections with the TPM and protect the user's Authorization Secret.
  - PIN for the security element (e.g., Smartcard, SIM). The TUI must provide a trusted path between the user and the security elements attached to the device.
  - GUI for Biometrics device.
  - User authentication (e.g., Login to remote/local service). To access remote entities, the TMM supports a security protocol to establish end-to-end security between a user and a remote entity.
- Signing
- Display of certificate details
- Display of security parameters.
- Device lock and unlock.
- Display access history.
- Display a warning message to the user regarding the inappropriate use of the

device.

### 11.2.1. Biometrics

Biometric devices are an alternative input method and extend the usability and security of the device. The TMM and the BioAPI framework should work together. Figure 11-4 shows the architectural diagram of the BioAPI and TUI [BioAPI]. The Biometrics Service Provider (BSP) has an optional GUI streaming call-back capability. The BSP calls the trusted GUI function supported by the TMM through this call-back. The Trusted Mobile Device that includes biometrics should support this capability.

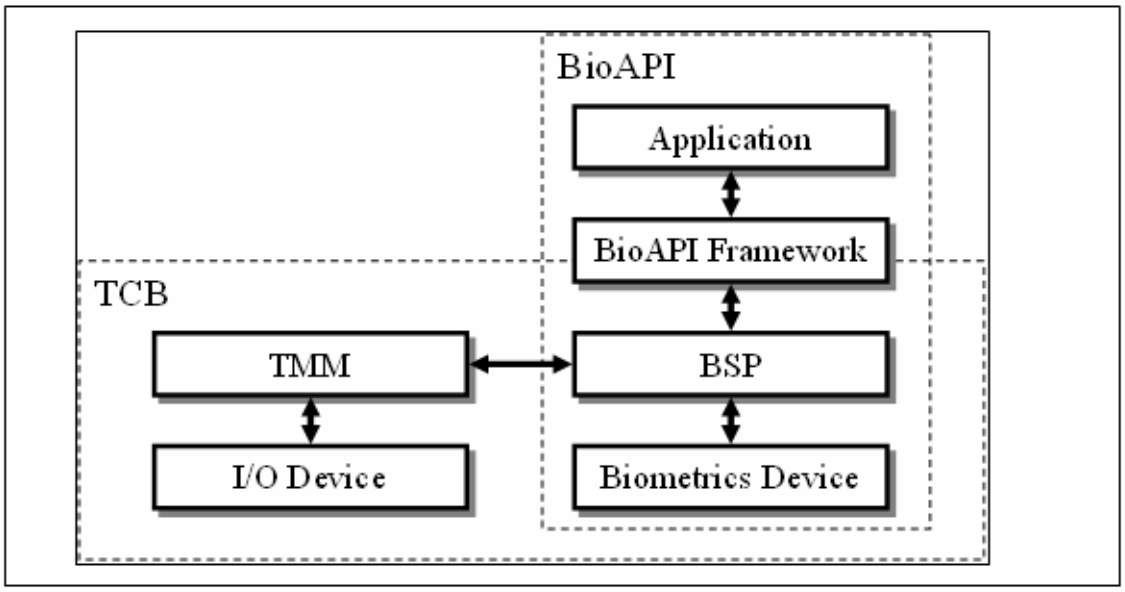


*Figure11-4 Block diagram of BioAPI and TUI*

## 11.3. Summary of Trusted UI and Security Levels

| Security Class | Class 1 | Class 2 | Class 3 |
|---|---|---|---|
| Trusted Mode Indicator (or Trusted Display Region) | No | Yes | Yes |
| Cryptographic mechanism for protecting the integrated I/O devices | No | No (rely on device-level tamper- evidence package) | No (rely on device-level tamper-resistance package) |
| Cryptographic mechanism for protecting the external I/O devices | No | Yes (if allowed to handle sensitive data) | Yes (if allowed to handle sensitive data) |
| BioAPI interface | No | Option | Option |

# 12. Provisioning / Device Management

"Provisioning" and "device management" are used to initialize or update resources in a Trusted Mobile Device, and retrieve information in a Trusted Mobile Device remotely. In this nature, those are tightly linked with a security policy.

The basis of trust in the Trusted Mobile Device is an integrity measurement of software components in a mobile device, and comparison between a predicted value and a measured value. Updating software will require updating a predicted value as well.

Security Class 1 device includes trusted boot with an integrity measurement. But due to its trust level, it is basically for a closed or semi-open platform. Provisioning and device management will be done in proprietary ways in most of cases or limited to the software in JVM that provides a security mechanism additionally

Security Class 2 and 3 devices must satisfy a set of requirements follows.

- Provide an authentication mechanism; the client or the server may send credentials to each other or challenge the other to send them.
- Provide for the integrity of messages passed between a client and a server; integrity is achieved by using HMAC (Hashed Message Authentication Code) or similar way.
- Provide confidentiality of information being transferred; use of SSL/TLS is encouraged
- Provide an access control mechanism along with an access control list (ACL). The access to resources will be based on an ACL.
- Provide an interface to update a predicted value in a secure way

## 12.1. Functions included

"Provisioning" is to manage handset parameters centrally and update parameters and software remotely. Parameters include telephone related information like roaming lists, and the required information for the data communication like a mail server address and a proxy server address. Typically, there are two types of provisioning. One is an initial provisioning to bootstrap a device from an un-configured state to a

provisioned state that is a normal operating mode. The other is continuous provisioning to a device that is in a provisioned state. The example is to download new software to provide a new service and set parameters properly.

"Device management" is the term mainly used in the IT world, and typically consists of the following functions.

- Configuration management
  - Configuration management is central management of the configuration parameters of devices.
- Software distribution
  - Software distribution includes an initial distribution of software, update of software and un-installation of software.
- Software and hardware inventory management
  - Inventory management is maintains information about the installed software, parameters for software/hardware setting, hardware vendor information including parts/serial number of a device, and hardware configuration information such as memory size up to date. This information will be referred by an administrator to distribute new software or change parameters, or by a customer service to identify the problem that a customer reports.
- Key Management
  - Key management is defined as the generation, registration, certification, storage, distribution, installation, usage, archiving, deletion, recovery, and destruction of key material and related structures and attributes in accordance with a security policy [ISO7498-2, NISTIR-4972, ISO11770-1]. Further, ISO7498-2 regards key management as part of Security Management, but does not dictate where this functionality must reside in a system or protocol stack.
- Notification (Alert)
  - Notification works in two directions. One is a notification from a server to a device sent via SMS and WAP Push [25], and the other is a notification from a client to server to inform of either an error condition or an alert threshold. A notification from a server is sometimes combined with one of device management functions to

initiate it from a server, but is not part of device management. An example of usage is to send a SMS to a device, invoke the device management program in a device to start communication with a server, and then update parameters. On the other hand, a notification from a client to a server is part of device management. This is for the early warning of an error to a network operator to prepare for an appropriate action to a device.

- Logging (Error log/History log)
  - Logging is relatively important in TMDs. Typically, logging includes an error log and a history log such as an installation log of programs. One additional important function regarding logging is a history log to include any security related incidents such as an unauthorized attempt of access important resources, any security policy changes, any attempt to call APIs in TSS, etc. Those event logs are stored in the device and retrieved from a server.
- Remote monitoring
  - This function provides monitoring of key events, screen drawn, program invocation, CPU usage, and others for remote diagnostics.
- Remote diagnostics
  - The remote diagnostics is an advanced function of remote maintenance to find the cause of errors remotely. The typical way of remote diagnostics is to invoke a program, which is delivered from a server or resides in a device remotely. This is totally device dependent functions, and is provided by a device manufacturer.

In addition to those basic device management functions, with considering the unique characteristics of mobile devices, an emergency shutdown mechanism is required for devices that are lost or stolen. This mechanism will invalidate all essential information and valuables in a lost or stolen device and restore them to another device if invalidation is successful.

Another unique aspect is [5] related functions which is part of inventory and configuration management. This is to retrieve values in PCRs remotely.

| Functions | Provisioning | Device Mgmt | Mobile/TCG | TMP Focus |
|---|---|---|---|---|
| Bootstrap | * | | | * |
| Configuration Management | * | * | | * |
| Software Distribution | * | * | | * |
| S/W and H/W Inventory | | * | | * |
| Key Management | | * | | * |
| Notification (Alert) | | * | | |
| Logging | | * | | * |
| Remote Monitoring | | * | | |
| Remote Diagnostics | | * | | |
| Emergency Shutdown | | | * | * |
| TPM Management | | | * | * |

*Table 12-1 Focused functions*

Table 12-1 shows the functions to be focused by each category. A function marked with an asterisk is a focused function. The rightmost column shows the functions to be focused on by the Trusted Mobile Platform research. There are two major reasons that specific functions are focused.

- Functions are required to build a trusted platform.
  - Bootstrap
  - Configuration Management
  - Software Distribution
  - Key Management
  - Logging
  - Emergency Shutdown
  - TPM Management
- Functions are needed for special consideration to work on a trusted platform.
  - Software and Hardware Inventory

All functions are optional for Security Class 1 devices. Even for Security Class 2 and 3 devices, all functions are optional. But, supporting Configuration Management, Software Distribution, Inventory and Logging function are recommended.

## 12.2. Management Framework

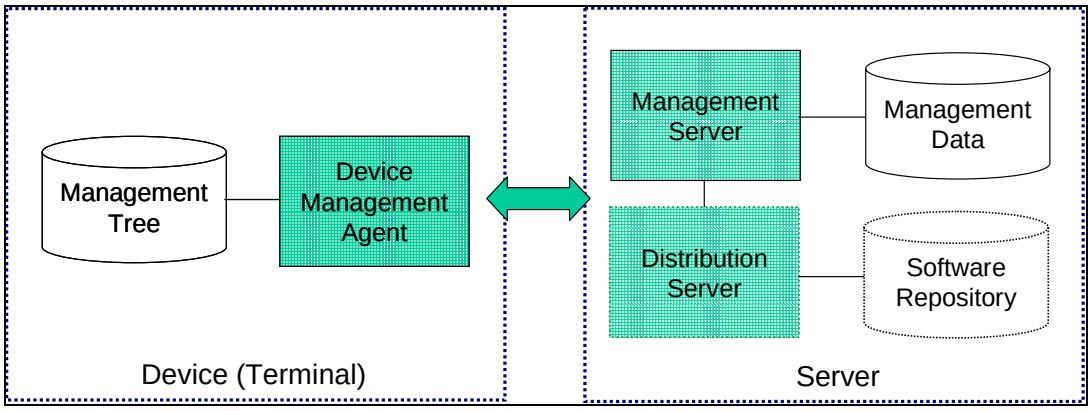


*Figure 12-1 Management Framework Overview*

The basic framework for the device management is shown above.

- **Management Server / Distribution Server**

  The management server handles several management requests working with the device management agent that is embedded into a device. The distribution server may not be a separate server, but helps the management server to distribute software to devices. Typically, an administrator submits a single request to the management server to apply it to multiple devices.

- **Device Management Agent**

  The device management agent communicates with the device management server, interprets requests and invokes device management tasks in a device to perform requests from the server. The examples of device management tasks are software distribution and configuration update. The communication between the device management agent and the device management server needs to be secured.

- **Management Data / Software Repository**

  The device management server has the management data and software repository to be delivered. Typically, the management data includes configuration parameters for a device, list of installed software, device information and user information.

- **Management Tree (in a device)**

  The management tree consists of management items (objects) and parameters (values) to them. By setting and reading values, the device management agent along with appropriate device management tasks interacts with the device.

The scope of this document is limited to define the software architecture inside a device. The design objective is to make this device management mechanism work with existing device management servers with embracing part of existing device management agent.

## 12.3. Initialization of a device

In most cases, a device is shipped from a factory in an un-configured state. A network operator or a service provider must initially provision it, but the way of the initial provisioning is totally dependent to the business model. The reference specification for this initial provisioning is Open Mobile Alliance (OMA) Provisioning Bootstrap 1.1.

Here is an overview to bootstrap a device from “un-configured” state to “provisioned state.

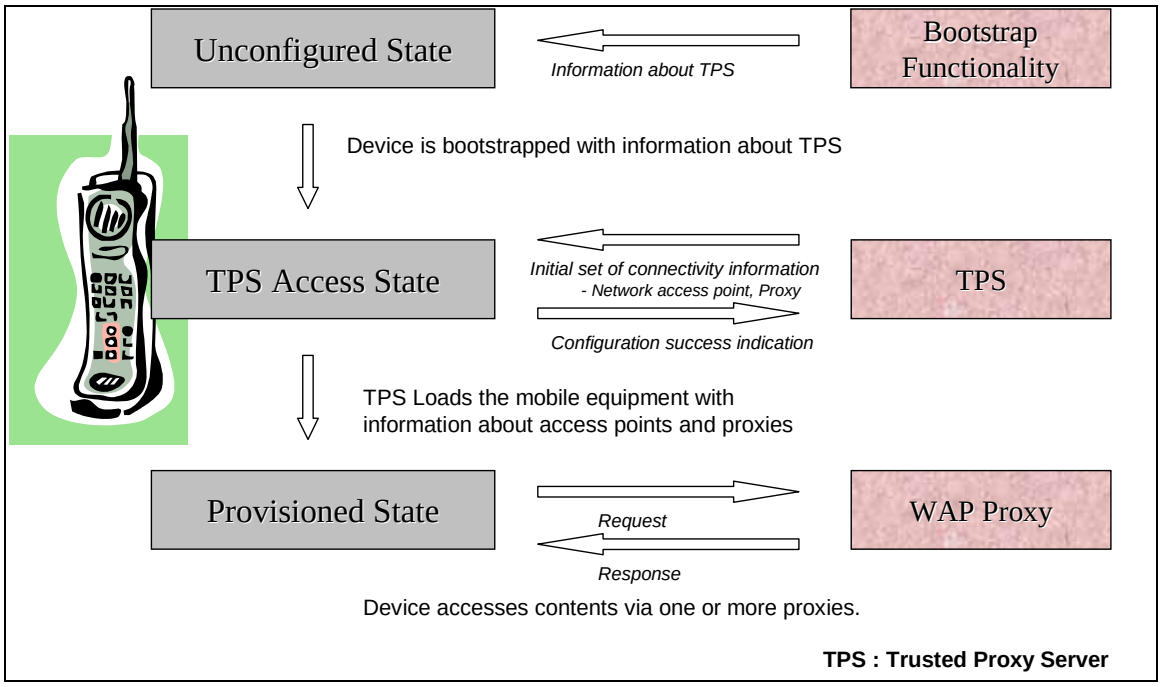


*Figure 12-2 Initial Provisioning*

Once a device goes to a provisioned state, the device management server will take over the management responsibility.

## 12.4. Configuration Management

Configuration management is needed to change a setting of parameters for OS, device drivers and applications remotely. The managed object is a single node (entity) in a management tree, and includes combinations of keys and values. By reading and writing a value of a target managed object, the configuration will be changed. Each managed object is "access controlled" so that multiple management entities can co-work for a single device.

## 12.5. Software Distribution

Software distribution is one of the most important management functions. Usually, this function is combined with software inventory management to track the software installed in the device. There are two types of software distribution. One is a user initiated download, and the other is an operator initiated download. There are several software downloading solutions available in today's mobile market. One example for user-initiated download is MIDP 2.0, which defines the protocol for downloading MIDlets. From the management point of view, both user initiated download and an operator initiated download need to be considered.

The software includes both Java program and Native program. From the management point of view, there are three types of management for software.

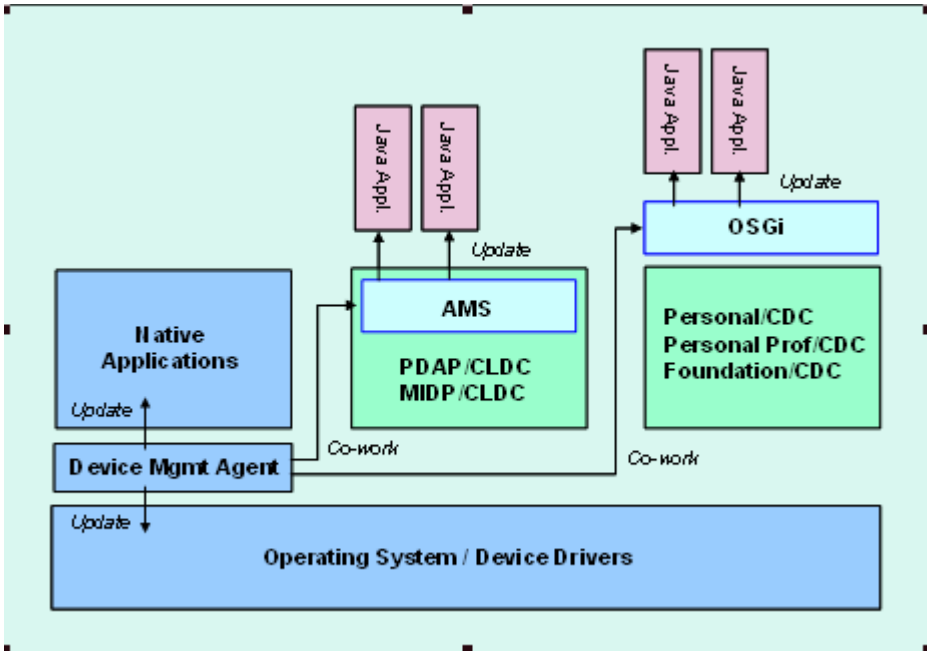


*Figure 12-3 Management Overview*

Platform management is the left-most portion of the figure is used to update the OS, device driver and other default native software. This mechanism can be used to download or update other native applications.

There are two types of Java application management corresponding to specifications in J2ME. One is the AMS (Application Management Software) specification, which is part of MIDP (Mobile Information Device Profile) and PDAP (PDA Profile), and the other is OSGi (Open Service Gateway initiative) specification that defines a life-cycle management of Java program that is called "Bundle" in OSGi's wording. MIDP and PDAP are the profile on CLDC (Connected Limited Device Capability) for the resource constraint device. OSGi reference implementation runs on Foundation Profile, Personal Profile or Personal Basis Profile on CDC (Connected Device Configuration), which is for the larger class of device. OSGi R3 defines Minimum as a proper subset of Foundation/CDC to reduce a footprint.

Both MIDP/CLDC and OSGi define their unique security model and policy. For example, MIDP 2.0 introduces the concept of trusted applications that may be permitted to use sensitive and restricted APIs. A trusted MIDlet suite is identified through signing and verification using X.509 PKI. An un-trusted MIDlet suite is a

MIDlet suite for which the origin and the integrity of the JAR file can not be trusted by the device, and run in the un-trusted domain with a restrict use of APIs. Security for trusted MIDlet suites is based on protection domains. Each protection domain defines the permissions that may be granted to a MIDlet suite in a domain. The permission consists of "Allowed" permission that explicitly allow an access to a given protected API with no user interaction, and "User" permission that requires an explicit user permission granted after the prompt given to the user.

The software distribution (update) sub-system will update the software/file/data which are not part of the device management agent itself. So, even if it is running in a complete separated domain in a Security Class 3l device, it needs to update software/file/data in the other domains. In addition to this, the software distribution (update) sub-system may update the critical portion of the platform. The TCB is one of the potential targets to be updated in critical cases. After the completion of updating software, the predicted value will be compared with a measured value to check the integrity of the software that needs to be updated. So, the device management agent or appropriate subsystems need to be part of TCB.

The Software update process is as follows:

1. The server and the client are authenticated to each other.
2. The server informs which software is to be updated.
3. The client checks whether the software specified can be updated by this server or not.
4. The client checks whether the software is currently in use or not. If not, then updating process will start. If in use, either the update is postponed or cached in a device in an appropriate format.
5. The client may back up the current image and configuration data.
6. The server sends a software image to the device.
7. The software is updated.
8. The client updates a predicted value of the software for an integrity measurement.
9. The client sends back a return code to the server.

## 12.6. Software and Hardware Inventory

The inventory function tracks the hardware information in a device and the version of

software installed in a device. The software inventory function is typically combined with the software distribution function to update old software for a new one. Similar to configuration management, the software and hardware inventory information will be retrieved from a management tree. Each managed object is "access controlled" so that multiple management entities can co-work for a single device.

## 12.7. Key Management

See Section 7.3 of [9] for requirements on key management on a TMD.

## 12.8. Logging

Logging is important for identifying problems or checking usage. A log file usually includes essential information for both the device manufacturer and user and must be encrypted. Those log files need to be retrieved from a server for diagnostic purposes. The set of requirement for a platform is as follows.

- Trusted Mobile Devices shall provide an interface to log event.
- A single record shall include, at least, date & time, user (requester), event type, event information and access right.
- Trusted Mobile Devices shall encrypt log files.
- Trusted Mobile Devices shall provide an interface to retrieve log.
- Trusted Mobile Devices shall provide an interface to flush log file.

Note: There would be different levels of logging on a TMD, such as TPM, TSS, OS, and application generated logs. However, support for a generic logging mechanism is not provided in the current version of specification and will be discussed in future versions of the specifications.

## 12.9. Auditing

At runtime, applications use auditing to generate statistics and track usage. The statistics and usage gathered can help define the data to capture. For example, in the case of auditing a high-value e-Ticket usage, as a statistic, the trusted mobile device operator (or service provider) shall use the audit data to find out how many people changed their e-Tickets during a given period. Furthermore, transaction attributes

(such as transaction and server IDs), transaction value and time shall be logged for further auditing and non-repudiation purposes. For tracking usage, the mobile device operator or service provider could track when an e-Ticket for a particular user was changed. The usage data can help track down issues in e-Ticket management and distribution.

Auditing services include the following major components, as shown in

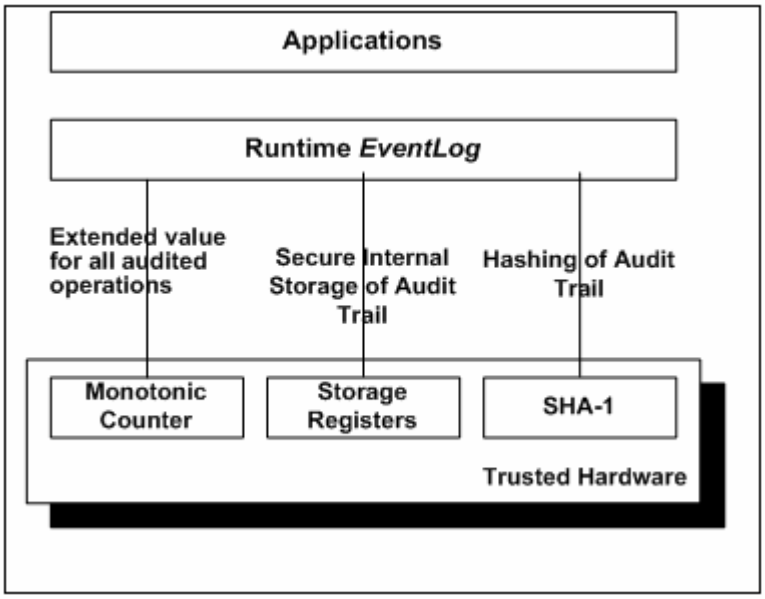


Figure *12*-4 Common Access Control Services (CACS) Architecture on Trusted Mobile Device:

- Audit trail generation and usage of the hardware-based signature and hashing capabilities.
- Secure storage of audit trail.
- Hardware-based time stamping.

To provide an owner of a trusted mobile device with the ability to determine that certain operations on the device have been executed, auditing of commands is necessary. In this specification, the audit value is a digest held internally to the TPM and externally as a log of all audited commands. With the log held externally to the TPM, the internal digest must allow the log auditor to determine the presence of attacks against the log.
It is proposed that the audit trails are signed by the TPM's RSA signing capabilities and

that they are hashed using SHA or MD5, and encrypted with a hardware-generated (private) key.

The trusted mobile device owner and service provider should have the ability to set which functions generate an audit event and to change which functions generate the event at any time. The status of the audit generation is not very sensitive information and so the command to determine the status of the audit generation is not an owner authorized command.

It is necessary to have verification of the external audit log both during a power session (i.e., while the device is powered) and across power sessions and to enable detection of partial or inconsistent audit logs throughout the lifetime of a TPM.

The Trusted Mobile Device should hold an internal record consisting of a non-volatile counter (that increments once per session, when the first audit event of that session occurs) and a digest (that holds the digest of the current session). The audit digest is volatile. It is easier to build a high endurance non-volatile counter than a high endurance non-volatile digest. This arrangement is insufficient, however, because it would allow the truncation of an audit log of any session without trace. It is therefore necessary to perform an explicit close operation on the audit session. If there is no record of a close-audit event in an audit session, anything could have happened after the last audit event in the audit log.

The essence of a trusted hardware-based audit recording mechanism consists of the following steps:

- The Trusted Mobile Platform shall contain a volatile digest used like a PCR (auditDigest), where the Integrity Metrics are digests of command parameters in the current audit session.
- An audit session shall open when the auditDigest is extended from its NULL state. This occurs whenever an audited command is executed AND no audit session currently exists, and in no other circumstances. When an audit session opens, a non-volatile counter is automatically incremented.
- An audit session closes when the Trusted Mobile Platform receives a TPM_GetAuditEventSigned command with a CloseAudit parameter asserted.

An audit session shall be considered closed if the value in the volatile digest is invalid (for whatever reason).

- TPM_GetCapability shall report the effect of TPM_Startup on the volatile digest. Note that the Trusted Mobile Platform may initialize the volatile digest on the first audit command after TPM_Startup(ST_CLEAR), or on the first audit command after any version of TPM_Startup, or may be independent of TPM_Startup.
- When the Trusted Mobile Platform signs its auditDigest, it signs the concatenation of the non-volatile counter and the volatile digest, and exports the value of the non-volatile counter, plus the value of the volatile digest, plus the value of the signature.

Note that if a TPM_SaveState is an audited command, TPM_SaveState should be issued before TPM_GetAuditEventSigned with CloseAudit asserted. This is safe because TPM_GetAuditEventSigned does not alter any parameter that is preserved by TPM_SaveState.

The Trusted Mobile Device shall maintain an audit monotonic count that is only available for audit purposes. The increment of this audit counter is under the sole control of the TPM if the TPM provides such capability or this shall be controlled by the TCB and is not usable for other count purposes. This monotonic count must be incremented by one whenever the audit digest is “extended” from a NULL state.

Each command ordinal shall have an indicator in non-volatile TPM memory that indicates if execution of the command will generate an audit event. The setting of ordinal indicator should be under control of the Mobile Device’s Owner.

The Audit Monotonic Counter (AMC) performs the task of sequencing audit logs across audit sessions. The AMC shall have no other uses other than the audit log. The TPM and the associated trusted mobile device should be matched such that the expected AMC endurance matches the expected platform audit sessions and sleep cycles. It is expected that such an AMC would not roll over. If the AMC were to roll over, and the storage of the AMC still allowed updates, the AMC could cycle and start at 0 again. Such an AMC must last for at least 7 years or at least 1,000,000 audit sessions whichever occurs first, which should be sufficient for expected mCommerce usage

scenarios, such as e-Ticket use models.

The Trusted Mobile Device shall generate an audit event in response to the execution of a function that has the audit flag set to TRUE for that function. The Trusted Mobile Platform shall maintain an extended value for all audited operations and it would keep an audit monotonic counter for the device (TCG_COUNTER_VALUE which starts at 0 and increments according to the rules of auditing).

Time stamping strengthens the security of an auditing function. The Trusted Mobile Platform should provide a very reliable, hardware-based time stamping service. The Trusted Mobile Platform shall provide a service to apply a timestamp to information blobs, such as audit logs. The timestamp, which should be provided by the trusted mobile platform, would not be an actual Universal Time Clock (UTC) value but it should be the number of timer ticks the TPM has counted. It is the responsibility of the calling application (say, an m-Commerce application) to associate the ticks to an actual UTC time, if deemed necessary.

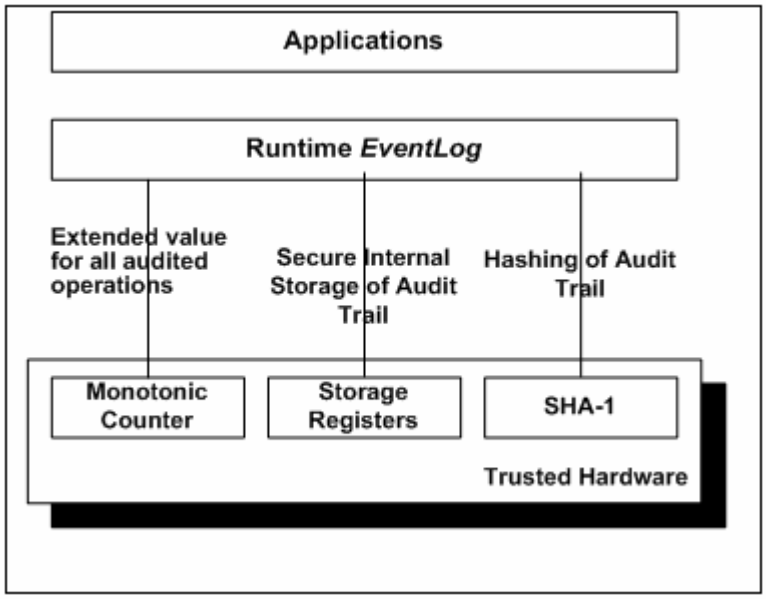


*Figure 12-5 Auditing Capabilities of Trusted Mobile Platform*

## 12.10. Remote shutdown and restoring

One of the key characteristics of a mobile device is that they are “easily lost or stolen”. On the secure platform, valuables such as digital cache and expensive digital contents will exist and can be transferred among secure platforms. In order to protect their use by unauthorized users and to protect the user's property, the following items are required.

- Properties in a device need to be encrypted. Properties include phone numbers, addresses, calendar information, and other valuable data for local applications in a device.
- Anonymous access to valuable information and content should not be allowed. Password protection or similar mechanism is required.
- Valuable content should not be transferred, even locally, without explicit permission from an owner.
- The option to make valuable information erasable and to shutdown a device remotely is required. In this case, the rights to use valuable content or value of digital cache should be transferred back to the server, and transferred to a new device if applicable.

From the Trusted Mobile Platform stand point, 1) is addressed by the encrypted file system (see section 7) 2) is a guideline for applications on TMDs, and 3) is to be addressed to DRM (Digital Rights Management) which is not a focus area for this research.

## 12.11. TPM Management

TPM Management is similar to configuration management and the inventory function. This allows the server to retrieve PCR values from the TPM in a secure way. Using TPM_Quote in a local device can retrieve these values. When this function is called, the TPM collects the values of current Platform Configuration Registers (PCRs), combines them with a nonce to generate a special structure called TCPA_QUOTE_INFO and returns the signature on this structure. This is required to ensure the platform is secured, and determine if the device management function should be started or not.

# 13. Trust Level Guidelines & Security Evaluation

As described in Section 4 of the Hardware Architecture Description Document, the trust levels defined for a Trusted Mobile Device are Security Class 1, Class 2, and Class 3. The Software Architecture characteristics associated with each level are defined in Table 13-1.

| Security Feature | Security Class | | |
|---|---|---|---|
| | Class 1 | Class 2 | Class 3 |
| TPM | -SW TPM or equivalent | - HW TPM ([5] subset) | -Integrated TPM or MCM |
| CPU architecture | -No requirement | - MMU | -HW domain separation, trusted DMA |
| Integrity and attestation | -Minimal integrity checks | - Integrity checks and source authentication (Trusted boot) | -Trusted boot<br>-Runtime integrity checks |
| Domain separation enforcement | -COTS OS separation (user account + process)<br>-Java (JAAS or OSGi) | - Hardened OS<br>- Encrypted memory system | -Secure processor architecture<br>-SW domain separation |
| Access control | - Discretionary | - Mandatory + discretionary | -Mandatory + discretionary |
| Software certification | - No certification | - CAPP EAL 2 or equivalent (No formal certification.) | -CAPP EAL 3 or equivalent |
| User authentication | -PIN | -Passphrase | -Hardware crypto<br>-Biometrics |
| Root of Trust | -None | -ROM or equivalent | -ROM or equivalent |
| SW Architecture | -No TCB | -TCB | -TCB |
| Secure storage | -No | - Through encryption | -Encryption & domain separation |

*Table 13-1 SW features by security level*

Security Class 2 and class 3 Trusted Mobile Devices must be evaluated against established public security standards. The best avenues for this certification are through the Common Criteria using EAL evaluation for software components. These certification methods are more systems and software oriented than is FIPS-140-2 and they have the added advantage of being recognized internationally. At a system level,

it may be difficult to achieve more than an EAL level 2-3 certification due to the lack of certain critical security features like HW and SW domain separation and trusted I/O. Security Class 3 level Trusted Mobile Devices will have these additional security features and should be developed with the objective of being able to achieve certification to EAL level 2-3.

Due to the complexity of the Trusted Mobile Device and the fact that different vendors will develop various components, the security evaluation should be executed in two phases. In the first phase, their developers will certify individual hardware and software components that are inside the security boundary. Items that should be subjected to individual certifications include the TPM, the secure processor, the CRTM, the Secure OS kernel, the TSS and other hardware or software components that can be readily bounded and evaluated.

Individually certified components can be assembled to construct a trusted platform, but the use of the certified components alone does not ensure that the resulting platform can be trusted. A platform level Common Criteria evaluation must also be performed. Use of individually certified components make the platform level evaluation less difficult and should result in achievement of a higher certification level. Appropriate platform level certification will be discussed in the future.

# Appendix A Implementation Options

## TPM Mandatory Functions

### TSS_Bind

Function that allows the creation of a TPM protected data, by an entity outside of the TPM. In contrast to the *TPM_Seal* function, the *TSS_Bind* function is used to only conceal data with encryption and is not applicable to other purposes such as using PCR values to determine the validity of data when decrypted.

### TPM_UnBind

Function that decrypts data that is originally encrypted for protection by the *TSS_Bind* function.

### TPM_CreateWrapKey

Function that allows the creation of a new symmetric key for TPM equivalents that use symmetric algorithms for wrapping.

### TSS_WrapKey

Function to create a migratable key from data outside of the TPM.

### TSS_WrapKeyToPcr

Function that allows similar functionality as TSS_WrapKey, with the exception that the key can be used to only seal data to a certain PCR.

### TPM_LoadKey

Function that allows the loading of a symmetric key pair into the TPM. A key must be loaded into a TPM prior to being used in any operation that the TPM performs. If the designated key is non-migratble, the TPM should check if the key is actually generated by the TPM.

### TPM_EvictKey

Function allowing the capability to unload a key from the TPM.

### TPM_GetPubKey

Function allowing the capability to get a public key from the TPM.

## Optional functions

### TPM_SaveKeyContext

Function that allows redundant keys to be flushed from TPM.

### TPM_LoadKeyContext

Function to reload cached keys into TPM again.

### TPM_Seal

Function that allows the data to be locked using the TPM protected storage mechanism. Data may be revealed only in a certain condition where the platform it resides on.

### TPM_Unseal

Function that allows data to be revealed after locked by the *TPM_Seal* function by the platform, when the platform is in the pre-defined condition.

# Appendix B Definitions and Abbreviations

For the purpose of this document, the following definitions apply:

| | |
|---|---|
| **AES** | Advanced Encryption Standard. New block cipher algorithm selected as a standard for U.S. federal government |
| **AIK** | Attestation Identity Key |
| **API** | Application Program Interface |
| **BAPI** | Biometric API |
| **BSAFE** | Library for cryptographic operations produced by RSA Security |
| **CA** | Certificate Authority. An entity that vouches a user's identity by issuing a digital certificate |
| **CBC** | Cipher Block Chaining. A mode of operation for block cipher |
| **CDSA** | Common Data Security Architecture |
| **CFS** | Cryptographic file system. File encryption mechanism incorporated into the file system, seen on several UNIX-like operating systems. |
| **CRT** | Chinese Remainder Theorem |
| **CRTM** | Core Root of Trust Measurement |
| **CryptoAPI** | Cryptographic API supplied by Microsoft's Windows Operating System |
| **CSP** | Cryptographic Service Provider |
| **DES** | Data Encryption Standard. Classical block cipher algorithm formerly selected as a standard for U.S. federal government |
| **DH** | Diffie-Hellman. A method to exchange secret data securely based on the hardness of discrete logarithm over finite multiplicative group. |
| **DSA** | Digital Signature Algorithm. One of the digital signature algorithms selected as a standard for U.S. federal government. This is based on the hardness of discrete logarithm over finite multiplicative group. |
| **ECB** | Electric Code Book. A mode of operation for block cipher |
| **ECDH** | Elliptic Curve Diffie-Hellman. A method to exchange secret data securely based on the hardness of discrete logarithm over group of elliptic curves over prime fields. |
| **ECDSA** | Elliptic Curve Digital Signature Algorithm. Digital signature algorithms based on the hardness of discrete logarithm over group of |

| | |
|---|---|
| | elliptic curves over prime fields. |
| EFS | Encrypted File System. Windows2000/XP's file encryption mechanism incorporated into the file system. |
| FIPS | Federal Information Processing Standards |
| IPSec | Internet Protocol Security. Suite of secure protocols that collectively provides confidentiality, integrity, and authentication mechanisms to IP. |
| JAAS | JAVA Authentication and Authorization Service |
| MAC | Message Authentication Code. A number derived from data and shared secret among creator and verifier(s), which can provide a means to verify the integrity of the data |
| OAEP | Optimal Asymmetric Encryption Padding. A construction method for asymmetrically encrypted data with provable security. |
| OIAP | Object Independent Authorization Protocol |
| OpenSSL | Open source version of cryptographic library developed by several volunteers on the Internet |
| OSAP | Object Specific Authorization Protocol |
| OSGi | Open Service Gateway initiative |
| PCR | Platform Configuration Register. One of the elements of TPM |
| PRNG | Pseudo Random Number Generator |
| PIN | Personal Identification Number |
| PKCS | Public Key Cryptography Standard. De-facto standard for syntax of cryptographically processed data, which is defined by RSA Security |
| PSS | Probabilistic Signature Scheme. A construction method for digital signature data with provable security. |
| RSA | Rivest-Shamir-Adleman. Asymmetric algorithm used for both encryption and signing. |
| RTM | Root of Trust for Measurement |
| SK | Storage keys. Lower level keys in TCG's storage key hierarchy tree. |
| SRK | Storage Root Key. The root key in TCG's storage key hierarchy tree. |
| SSL | Secure Sockets Layer. A security protocol invented by Netscape Communications, which protects client/server style communications over the Internet. |
| TCB | Trusted Code Base |

| | |
|---|---|
| **TCG** | Trusted Computing Group |
| **TCPA** | Trusted Computing Platform Alliance |
| **TCS** | TSS Core Services |
| **TLS** | Transport Layer Security. SSL's descendant that is under standardization in IETF |
| **TMI** | Trusted Mode Indicator |
| **TMM** | Trusted Mode Manager |
| **TMP** | Trusted Mobile Platform |
| **TPM** | Trusted Platform Module |
| **TSP** | TSS Service Provider |
| **TSS** | TPM Support Software |
| **TUI** | Trusted User Interface |
| **UICC** | UMTS Integrated Circuit Card. A smart card module equipped with 3GPP user equipment. A UICC contains at lease one USIM application |
| **USIM** | Universal Subscriber Identity Module. A UICC-stored application used for authentication between subscriber and network operator. |
| **UUID** | Universal Unique Identifier |
| **VM** | Virtual Machine |
| **WAP** | Wireless Application Protocol |
| **WIM** | WAP Identity Module |