DELIVERABLE D5.1

Functional specification of the Secure Docking Module

Title of Contract	Seamless Communication for Crisis Management
Acronym	SECRICOM
Contract Number	FP7-SEC-218123
Start date of the project	1st September 2008
Duration	44 months, until 30th April 2012
Date of preparation	May 2009
Author(s)	Daniel Hein, Peter Danner, Apostolos Fournaris, Martin Liebl
Responsible of the deliverable	TUG, Daniel Hein
Email	daniel.hein@iaik.tugraz.at
Reviewed by:	Cathy Walter, QinetiQ
Status of the Document:	Final
Version	1.0
Dissemination level	PU Public

Table of Contents

Abstract

The *Secure Docking Module (SDM)* is a single-chip security device that protects information. The protected information is only released to a requesting client, if the client can prove that it is in a *trusted state*. The determination of the *trusted state* and also its verification by the SDM is based on *Trusted Computing* principles. If the SDM is used to protect cryptographic key material, proof of possession of this key material allows conclusions about the state of the key-holder.

This document provides a specification of the functional interface of the SDM based on the information gathered in WP2 and the security requirements specified in WP4. It starts out with a more detailed description of the SDM concept in Sections 1, 2 and 3detail the logical interfaces of the SDM, introduces the concept of an administration host and specifies the internal data organization of the SDM. The next section (section 4) describes the session establishment protocol used to connect to the SDM and achieve an encrypted and authenticated communication channel between a host device and the SDM. The sections 5, 6, and 7 specify the commands that must be understood by the SDM grouped by their relation to the logical interfaces of the SDM. Section 7 concludes this functional specification of the SDM by considering the hardware requirements of the SDM and related security topics. The appendix provides the documentation of the high level Java interface of the SDM.

1. Introduction

1.1 The Secure Docking Module

The *Secure Docking Module (SDM)* protects cryptographic keys. In technical terms the SDM is a key storage device with local attestation (the process of establishing its state) verification capabilities. Conceptually, the SDM protects a small set of key pairs for asymmetric cryptography, but in general is capable of protecting arbitrary data up to a specific size. The SDM's key protection facilities are a standard function, which could already be implemented with today's smart cards or hardware security modules. The SDM extends this standard function by only releasing these keys to a host device if and only if this host device is in a *trusted state*. This host device is called *Trusted Docking Stations* (TDS) and is called local attestation (see Section 2.3). The relationship between SDM and TDS is depicted in Figure 1.

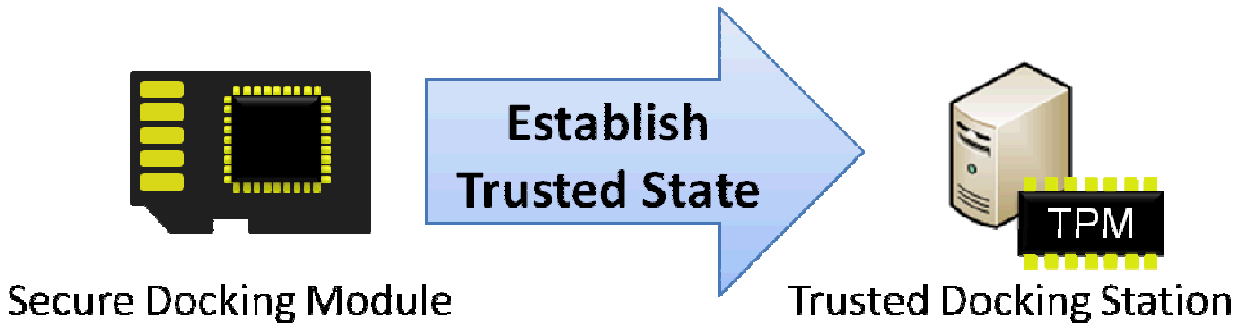

Figure 1: The relationship between the Secure Docking Module and the Trusted Docking Station

A *trusted state* is a specific software configuration. This software configuration is measured by using a *Trusted Platform Module (TPM)*. The TPM is a special security chip providing amongst other functionalities the protected capability of measuring the software configuration of its host device. A TPM must be present in the TDS. The combination of a SDM with a TDS is called a *Secure Docking Station* (SDS).

The idea of the SDM/TDS concept is that if a TDS is in a *trusted state*, it can be trusted to adhere to a specific *policy*. A *policy* is a set of rules that constrains the behavior of a device for all conceivable situations. **This gives the TDS the freedom to execute any program that can be run as part of a *trusted state***.

1.2 Motivation

In order to provide a secure communication infrastructure in accordance with the requirements of the SECRICOM project, the particularities of the infrastructure must be understood. Two main categories of communication devices are used – mobile devices, and PCs or PC derivates. These devices are constructed for very different purposes. Mobile devices have limited processing power, limited input (for example only number keys), and limited battery power. PCs on the other hand have a constant power supply (via connection to the main supply), various interfaces for input devices, memory card devices

and other kinds of extensions. To show how mobile devices and PCs are involved in the communication two use cases are explained consecutively.

1.2.1 Mobile Devices in a Messaging Communication Scenario

Mobile devices normally do not have extra slots for smart cards. They also mostly do not have wired Secure Elements and, even less, have a TPM. To be able to use a wide variety of mobile devices, common security features must be utilized. Nearly every mobile device has one or more slots to carry memory cards. The SDM will also have such a multi-purpose memory card interface, thus it can be used with these mobile devices (Figure 2). Credentials for applications and communication establishment are stored on the SDM and released only under defined conditions following special release protocols (cf. section 4).

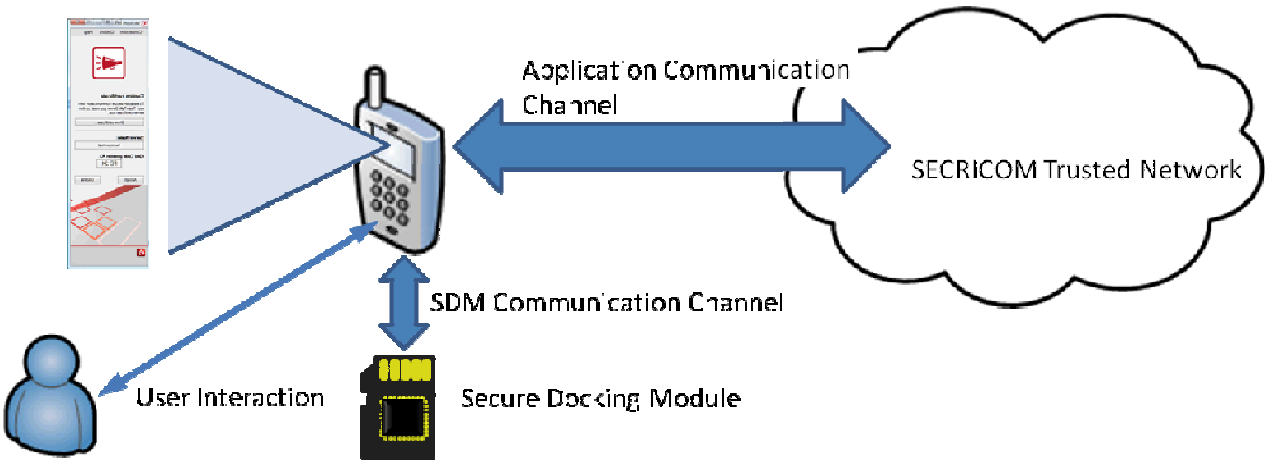

Figure 2: SECRICOM mobile application scenario

During incidents users do not have time to work with complex, time-consuming processes and they must operate under difficult conditions. Mobile applications therefore must be easy to use. Security has to be provided transparently via a SDM and the mobile devices on their part. The remaining functionality that the user actively can use is its store for credentials. Depending on the applications (and devices state(s)), in this scenario, users are releasing keys through providing passwords by requesting the SDM.

1.2.2 Trusted Secure Agent Execution Environment Scenario Using PCs as Working Platform

In environments where fixed infrastructure can be presumed, communication devices are obliged to be equipped with a TPM and to utilize it. The benefit of these hardware components is that they ensure the *trusted state* of a platform and provide the possibility to bind software to the Secure Docking Stations depicted in Figure 3. In this scenario this software is the Secure Agent Execution Environment (SAEE) running on the Secure Docking Stations, which are part of the SECRICOM Agent Platform. This enables determination of which agent is allowed to be executed on which specific SAEE in homogenous Agent Environment installations, because it is possible to predetermine the agents for the

available Secure Docking Stations by binding them to TPM identities. The establishment and utilization of trust is specified by the Trusted Computing Group and explained in chapter 2.3. In this scenario, shown in Figure 3, credentials can be used automatically. For this purpose the SAEE must communicate with the SDM and prove to it, that it is in a *trusted state*. If the SDM successfully verifies the state of the Secure Docking Station including its running SAEE, it will release the requested key material respectively credentials. This allows for automated encryption processes, e.g. the encrypted transmission of agents.

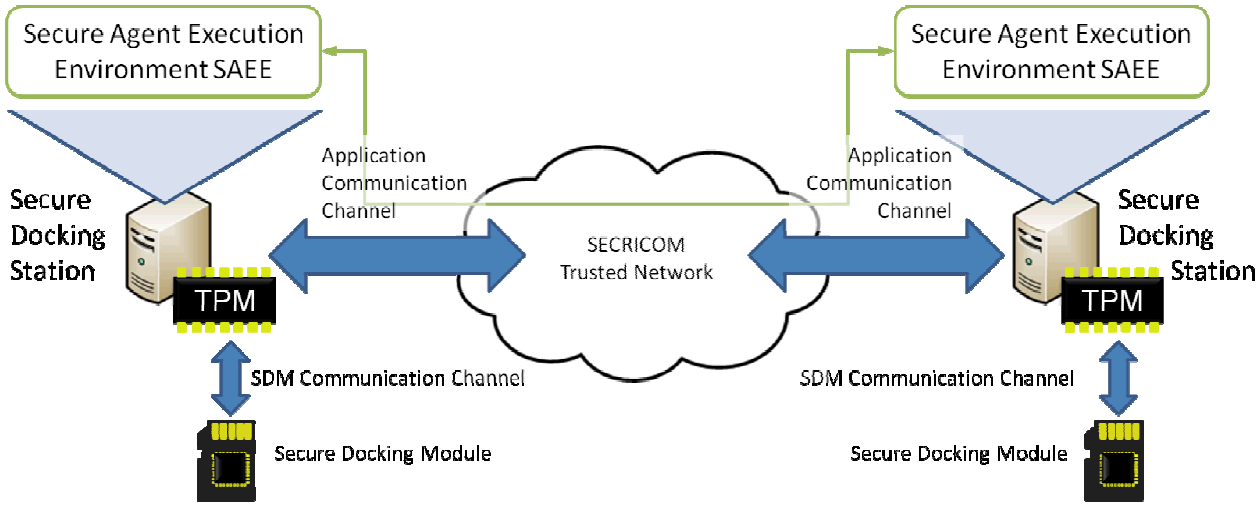

Figure 3: Agent communication scenario using SECRICOMs communication security

As one can see the SDM is a central component in SECRICOMs communications infrastructure. It has to be maintained and prepared for use. Also its interfaces and functionality have to be specified in great detail.

2. Trusted Computing and Virtualization

2.1 Measuring the Host Device

The TPM is part of the concept of *Trusted Computing*. *Trusted Computing* is driven by the *Trusted Computing Group* and tries to achieve security through providing an automatic means of establishing trust in a platform. *Trusted Computing* based on the TPM measures the platform, in this case the SDM host device, by establishing a so-called *Chain of Trust*.

The *Chain of Trust* guarantees that every software component of the protected platform is measured before it is executed. It is not possible to hide the execution of a software component. The *Chain of Trust* is an uninterrupted chain of measurements starting at the *Core Root of Trust for Measurement*. The *Core Root of Trust for Measurement* is either a BIOS component or an authenticated code module. The *Chain of Trust* is built on the principle of measuring before executing. A measurement is the SHA-1 hash of a component and its relevant configuration. Measurements are stored in so-called *Platform Configuration Registers* inside the TPM.

Figure 4 provides an illustrative example of the concept. For this example it is assumed that the Core Root of Measurement is a part of the System BIOS. Also, the example has been simplified and abstracts a few measuring steps for sake of simplicity

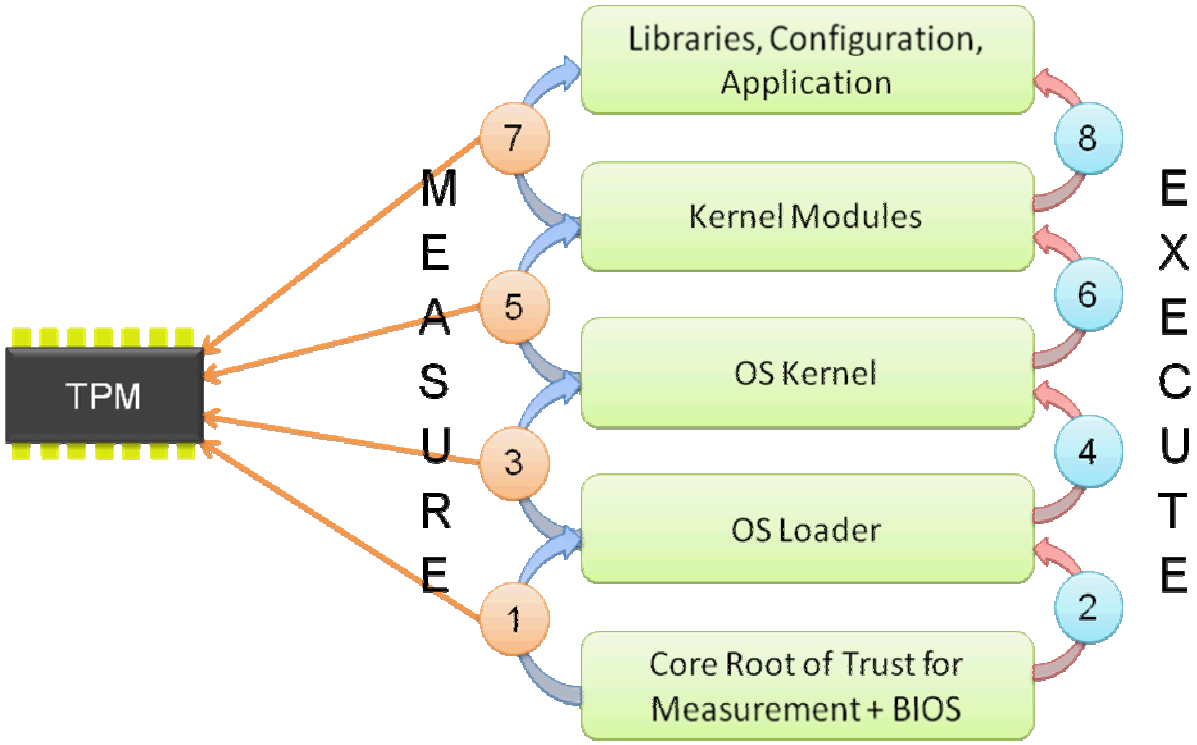

Figure 4: A sample Chain of Trust

At boot the Core Root of Trust for Measurement measures the System BIOS by computing a SHA-1 hash of it. The measurement is stored in the TPM and the BIOS is then given control of the platform. The BIOS performs its system initialization duties. After that it measures the master boot record of the boot partition, stores the measurement in a *Platform*

*Configuration Registers (*PCR) and transferring control master boot record. The master boot record performs the same procedure with the OS boot loader. This *Chain of Trust* is then continued up until application level.

The TPM standard specifies that a TPM must provide at least 24 PCRs. This number is much too small to contain a detailed measurement of all components of a system, but every component must be measured, for the *Chain of Trust* concept to work. To solve this problem a way to reuse PCR registers has to be found. Simply overwriting values in the PCRs is out of the question, as this would allow software components to hide their execution and thus break the *Chain of Trust*. This is solved by prohibiting direct writing to the PCRs. PCRs can only be extended. The *PCR Extend* operation appends a new measurement value to the value stored in a PCR and the result is hashed using the SHA-1 cryptographic hash function:

$$PCR\_EXTEND(n, v): PCR_N = SHA\text{-}1(PCR_N || v),$$

where *n* is the number of the PCR to extend and *v* is a measurement value. Due to the properties of a cryptographic hash function it is not possible to manipulate the *Chain of Trust* without detection.

On the hardware level, depending on the Core Root of Trust, building a *Chain of Trust* requires cooperation either between the TPM and the BIOS, or between the TPM, the chipset and the Central Processing Unit (CPU). On the software level each software component must adhere to the measure before execute paradigm.

2.2 *Trusted Docking Station and Virtualization*

The implementation of a *Chain of Trust* requires the cooperation of several hardware and software components. The software components include various parts of the *Operating System* (OS) of the platform. Thus, integrating the *Chain of Trust* into an existing OS infrastructure requires numerous changes. Virtualization can all help alleviating this problem. In addition it allows establishment of protected execution environments.

Virtualization provides an abstraction of a physical platform that is known as a *Virtual Machine*. The term virtualization encompasses a variety of virtualization techniques. For the purpose of this document we use the term to refer to a fully virtualized platform. A fully virtualized platform provides two important capabilities:

- It abstracts the physical characteristics of physical platform
- It provides isolations of the *Virtual Machines*

The term *Virtual Machine* is also used in conjunction with the Java programming language and the Java Runtime Environment. To avoid confusion a *Virtual Machine* in the context of hardware virtualization will henceforth be called a *compartment*.

A hypervisor or Virtual Machine Monitor (VMM) virtualizes a physical platform and enables the execution of isolated compartments. Isolation is basically achieved by granting each compartment access to the CPU, memory and interrupts, whereas the hypervisor stays in

control of the MMU. A virtualized platform is illustrated by Figure 5. Each compartment runs its own operating system and set of applications.

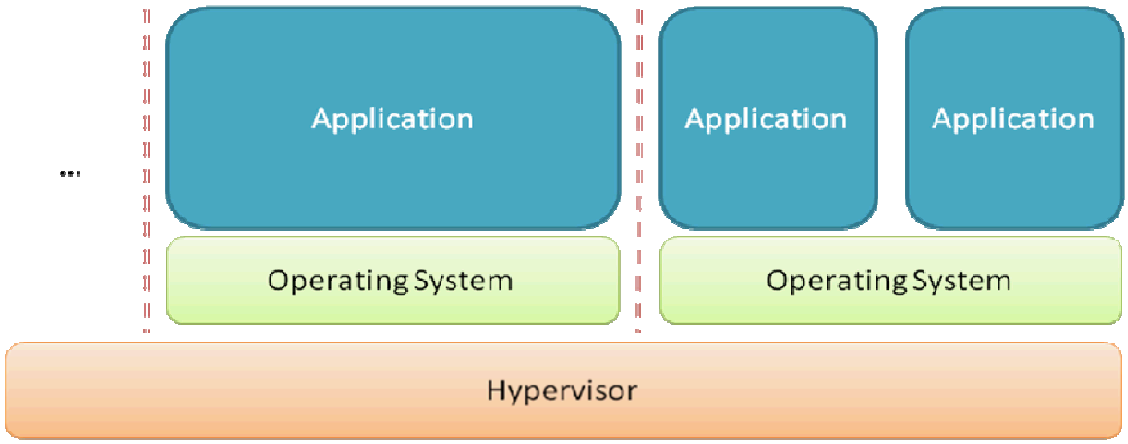

Figure 5: A schematic of a virtualized environment, where a hypervisor manages the operation of several separated virtual machines.

Concerning the SDM concept a virtualized environment has two beneficial effects: It facilitates platform state measurements by enabling the measurement of a complete compartment and the isolation allows executing services in a trusted state next to less trustworthy compartments.

The *Chain of Trust* for this virtualization based execution model requires measurement of the hypervisor. The hypervisor in turn must measure a compartment image before execution. To enable the trust decision made by the SDM, when the state of the virtualized platform is reported to it, the trusted application compartment image must be immutable. Thus, it is comparable to a CD/DVD-ROM image booted in a virtual machine. Mutable data and variable configurations must be provided by another source, for example a network share or a mounted hard disk image.

2.3 *Local Attestation*

The *Chain of Trust* ensures the accurate measurement of a platform. The benefit of this knowledge arises when this state is reported to the SDM and the SDM releases a key. The cryptographically secured process of providing this state information to a verifier (the SDM) is called *Attestation*. In the *Trusted Computing* concept, verification of the attestation information is handled by a powerful, trusted and networked entity with access to a wide array of resources. This is called *Remote Attestation*, because it is done over a network. The SDM enables *Local Attestation*. Thus, it can provide access to protected information even if no Remote Attestation service is available, but the SDM is also much more restricted both in computing power and in storage compared to a *Remote Attestation* service.

The basic *Local Attestation* process which enables the release of a key is also called Key Release Protocol and is illustrated in Figure 6. In order to release a key, the first step is for the host to request it by sending the key identifier to the SDM. The SDM randomly

generates a nonce *n*, which ensures the freshness of the signed platform configuration report generated by the host's TPM in the next step of the protocol. The platform configuration report contains the SHA-1 hash of a selected set of *PCRs* and is signed with a so-called *Attestation Identity Key (AIK)*. The SDM generated nonce *n* is also signed. This platform configuration report is named *TPM Quote*. In the next step, the SDM proceeds to verify the signature, the nonce *n*, and the actual platform configuration. If the platform configuration is equal to a previously determined *trusted state*, the key is released.

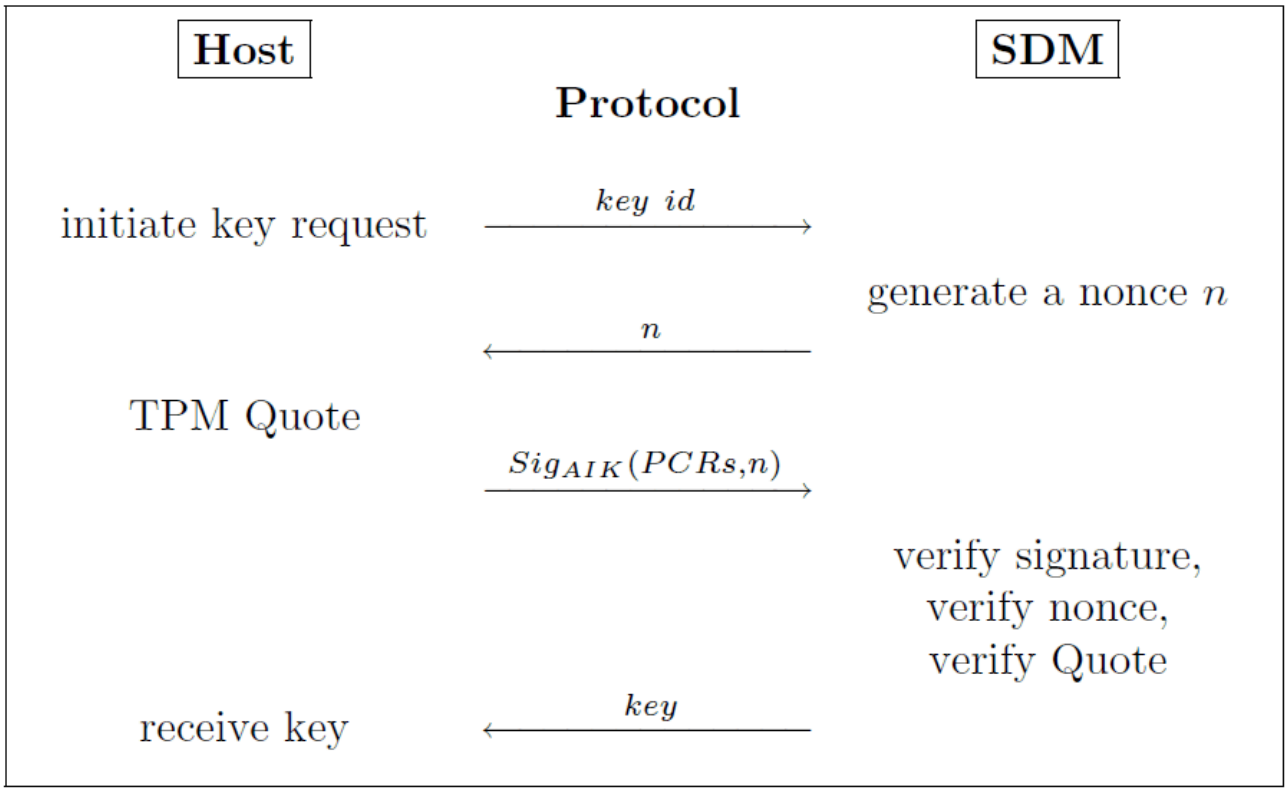

Figure 6: The Key Release Protocol

3. Secure Docking Module Interfaces

The *SDM* provides two different logical interfaces. The first, the client interface, provides the functionality required by an entity that uses the SDM. It consists of functions to verify the state of a host device, retrieve keys and generate random numbers using the SDM's random number generator. The second is the administration interface and allows an authorized entity to add, change and remove data protected by the SDM.

3.1 The SDM Client Interface

The client interface is made available through a client session with the SDM. For details on the establishment of a client session see section 4.3. For the commands available in a client session cf. to section 5.

3.2 The SDM Administration Interface

The administration interface is available during an administration session with the SDM. Administration session establishment is described in section 4.2. The available commands are enumerated in section 7. The SDM administration interface is considerable more complex than the client interface and warrants a more detailed description. Central to the administration concept of the SDM is the so-called *Administration Host*.

3.2.1 The SDM Administration Host

The SDM can only be administered by one entity. This entity is called *Administration Host*. It is possible to change the *Administration Host*. The *Administration Host* can remove itself and designate a new *Administration Host*. It is not possible to have more than one *Administration Host* at the time, nor is it possible to distinguish between different *Administration Hosts*.

The *Administration Host* can grant new entities client access to the SDM or revoke the access of existing entities. It can also update the data associated with an existing entity. The *Administration Host* can add new protected data to the SDM or remove existing protected data (cryptographic keys) from the SDM. It can also read and modify the protected data. The *Administration Host* specifies the protection mechanism for protected data that is it determines under which conditions a client entity may retrieve specific protected data from the SDM.

The *Administration Host* has full control over all data stored in the SDM; therefore the *Administration Host* must be a *TDS*. Administration session establishment must ensure that the *Administration Host* is in a *trusted state*. The *Administration Host* must be specified before an SDM can be used. For this reason, *Administration Host* injection is an early step in the SDM lifecycle.

3.2.2 The SDM States

The SDM from the time that it is fabricated to the end of its life, passes through a variety of different states. Each state has unique characteristics that specify the functionality of the SDM. The more important states are the following:

UNINITIALIZED: This is the state that characterizes the SDM directly after fabrication. In this state the SDM doesn't contain any information at all. Specifically it has no identity (unique identifier), any cryptographic proof of its identity, or any cryptographic keys or related hosts. Also, there is no administration host written inside the SDM. In this state the SDM storage elements are totally blank. The only possible command that can be addressed to the SDM is the command that inserts the unique identifier in the SDM.

PROGRAMMABLE: This state characterizes the SDM after the identity injection operation. The SDM does not contain any information apart from its unique identifier and associated cryptographic key. No other cryptographic keys or credential can be stored in the SDM since the Administration Host is not specified in this state. The only possible command that can be addressed to the SDM is the command that inserts the Administration Host in the SDM.

READY: It is the state that characterizes the SDM after it is uniquely identified (unique identifier) and an administrator host is assigned to it. At this state the SDM can be connected to an administrator host and can be filled with the necessary host and keys information along with the related credentials (TPM quote, Authorization Token). The SDM in READY state is fully operational and the full repertoire of SDM commands can be addressed to it.

3.2.3 The SDM Lifecycle

A freshly fabricated SDM is completely devoid of any information. More precisely a new SDM has no identity (unique identifier) and no cryptographic proof of its identity. Also, no administration host is specified.

The SDM Lifecycle is depicted in Figure 7. The first step in the SDM lifecycle is to inject the SDM with its unique, cryptographically corroborated identity. This operation can only be performed, when the SDM is in the UNINITIALIZED state. The successful execution of the **Inject Identity** operation irrevocably changes the state of the SDM to PROGRAMMABLE. When the SDM is in this state, it is possible to specify the Administration Host of the SDM. Once the **Inject Administration Host** command successfully executes, the SDM is again irrevocably set to the READY state.

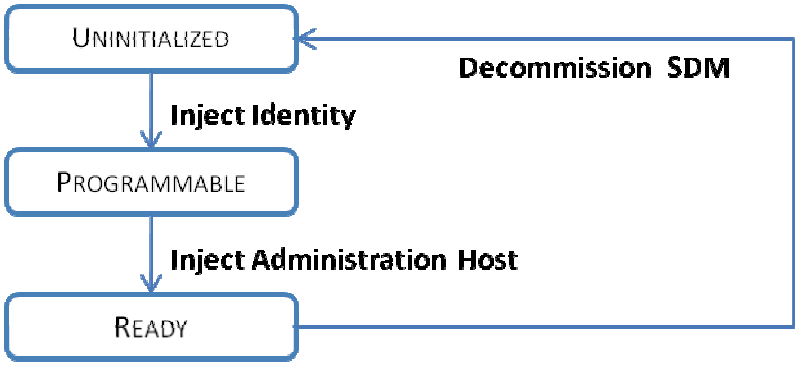

Figure 7: The SDM Lifecycle

In the READY state the SDM accepts connections from a host; it is possible to establish a client or an administrative session with the SDM. The commands that control the lifecycle of the SDM are detailed in section 6.

Deployment Procedure of the Secure Docking Module

The aim of the deployment process is that emergency service workers and other involved personnel, can get a SDM to be able to communicate to other parties using the SECRICOM infrastructure. It is essential that all communication partners in the SECRICOM network can rely on the security of the network and the trustworthiness of their communication partners and their platforms, be it a mobile device, a PC, or a Laptop. To determine the communication partners and to prevent misuse, a pre-prepared SDM is given out to each person, which needs to communicate in the SECRICOM network. All for this person necessary protected key material is stored on this SDM. Therefore, someone must know the scope of operation for each participant and hence the composition of credentials for his SDM. This is coordinated in the Command and Control (C&C) Centre.

Prerequisites

To prepare and make an SDM suitable for appliance some persons and entities must work together. They must be available or be physically present for the start of the SDM Lifecycle:

- Command and Control Centre (C&C Centre).

  This institution is the organizational unit that cares about the total amount of SDMs and the identification of users and their facilities (this can be done electronically as well as paper based). The C&C Centre is responsible for supervision of the rollout and evaluation processes concerning the SDM Start of Lifecycle phase and also takes care about the physical revocation of SDMs.

- Blank SDMs (are a physical and logistical part of the C&C Centre)

  Empty SDMs or SDMs with suitable Administration Hosts are needed (they must be changeable at the C&C Centre or any institution authorized by the C&C Centre institution).

- Administration hosts

Administration Hosts are operated by authorized persons. For the description of and requirements to Administration Hosts see section 3.2.1.

- Evaluated States for evaluated machines and key material to protect

  To bind credentials to certain platform configuration states, evaluation entities have to inspect the computers and the used and running software. The SDM implements an automatic process to enforce the trust decisions made by these qualified entities. A suitable trustworthy software bundle could be prepared beforehand, by a qualified party. This qualified party is responsible for choosing software that adheres to the required policies. Next an entity measures this software using the TPM in the platform, thus identifying the exact configuration for this specific platform, the *trusted state* of the platform. This *platform configuration* is mapped into the PCRs of the platforms TPM. The term *platform configuration* here encompasses all the software on a platform plus the actual configuration of this software. For new platforms, this must be done before connection to the infrastructure. Another way to protect key material and other credentials is to use only user-passwords. The prospective user of the SDM could provide his own password during the rollout process. This is less secure, but does not require a TPM. A third way of application of data is to inject a predefined set of data, e.g. each SDM is equipped with the same base information.

Start of the Lifecycle of an SDM

1) SDM and administration host have to be brought together (put SDM into Administration Host)

2) All necessary data for the SDM have to be gathered. Platform configurations have to be generated and sets of data have to be bound to them. The result should be a complete SDM data structure (hosts, keys, trusted platform configuration, and passwords) which can be injected in step 4.

3) Initialization Process of the SDM as described at the beginning of this section. Resulting in the READY state. This includes the personalization through application of the unique identity for the SDM.

4) Injection of the SDM data structure prepared in step 2. This includes the hosts, their associated key material, and key protection data, such as the trusted platform states and/or a password.

5) Registering a user to the SDM (paper based or electronically), maybe provision of the password(s) by the user. Hand out of the SDM.

To get a picture of how such a rollout could take place, an **example** for an SDM-Rollout is given next: An expert wants to join the to help in case of an emergency by taking snapshots of the environment. He goes to the C&C Centre where he wants to get equipment. His identity will be determined and mapped to an SDM, and therefore bound to the SDM. He comes with his own software he wants to use and gives it to the qualified administrators. The qualified administrators are setting up a clean system with predefined

and user software. The predefined software set also covers the communication software, which every user is expected to use. Then the administrators attest the platform configuration and create the data structure organization of the SDM. Sequentially each credential will then be stored on the SDM bound to the obtained platform configuration. Finally the user types in his password for the unbound credentials he gets and receives SDM and the prepared Laptop. He then is ready in case of an emergency to swarm out and contribute his part of help.

End of the Lifecycle of an SDM

The end of the Lifecycle of an SDM is the result of the predetermined decommissioning process. To decommission an SDM all its data is stripped off including its unique identification. This is necessary, because if an SDM is revoked, it could not be reused otherwise. The decommissioning process is as follows:

1) The SDM must be brought to and put in its Administration Host
2) All user data and key material will be safely removed by an administration entity. For each data structure the Administration Host has to get the list of credentials and sequentially delete the obtained list entries.
3) With a reset command, which combines three phases - the extinction of the identity and the master key pair of the SDM as well as the Administration host, together with the exchange of the state of the SDM to UNINITIALIZED – the lifecycle of the SDM is terminated.
4) Now the SDM is in the UNINITIALIZED State and ready for reuse

The C&C Centre knows about the identities of its SDMs and therefore can exactly determine if, and when a revoked unique identity can be reused within a new start of the lifecycle process of an SDM. Identity-recycling makes sense at the latest if a new incident occurs and an all-encompassing SDM rollout process starts.

3.2.4 SDM Protection Mechanisms

The requirements-conditions that a host must fulfill in order for the SDM to release its protected keys related to this host constitute the SDM protection mechanism. Each protection mechanism is characterized by the type of credentials (certificates) that a host can provide to the SDM to prove its identity and if it can be trusted or not. Since the hosts of an SDM are either equipped with a TPM (TDS systems) or not equipped with a TPM (SiFD systems) there can be identified three different possible credentials and therefore three different protection mechanisms.

Platform Configuration Protected mechanism (PCP): In this mechanism, the credential provided to the SDM by the host is a TPM quote. The TPM quote is a valid platform configuration of the host and can indicate whether or not the host is in a *trusted state*. The TPM quote value provided by the host, is compared to the value, associated with the requested key and host, stored in the SDM and if the two values match then the key is released. In order for this mechanism to be possible the host must be equipped with a TPM chip.

Authorization Token Protected mechanism (ATP): In this mechanism, the credential provided to the SDM by the host is an authentication token (password) that is given by the host's user. The authentication token is host related and constitutes the only measurement indicating that a host is authorized to receive an SDM key associated with it. The authentication token value provided by the host, is compared to the value stored in the SDM and if the two values match then the key is released. No TPM chip is required for the key release operation of this protection mechanism. The mechanism should only be utilized when the TPM use is prohibited or a TPM chip is not present on a host.

Authorized Platform Configuration Protected mechanism (APCP): This mechanism is a combination of PCP and ATP. The credential provided to the SDM by the host is both the host's valid platform configuration and an authentication token given by the host user. In APCP mechanism the authentication token value and platform configuration provided by the host, are compared to the values associated with the requested key and host, stored in the SDM and if the values match then the key is released. So, in order to have access to the SDM key materials, the host has to be in a *trusted state* and be able to provide adequate authentication of itself (through the authentication token). In order for this mechanism to be possible the host must be equipped with a TPM chip. APCP constitutes the strongest key protection mechanism among the three approaches and should be used when administered when the required security level is high, like in administration host keys.

3.2.5 The SDM Data Model

The SDM data model groups protected information (cryptographic keys) by the entities that are allowed to access the protected information, so-called hosts. Figure 8 illustrates the SDM data model. The basic data primitives used in the SDM are detailed in Table 1.

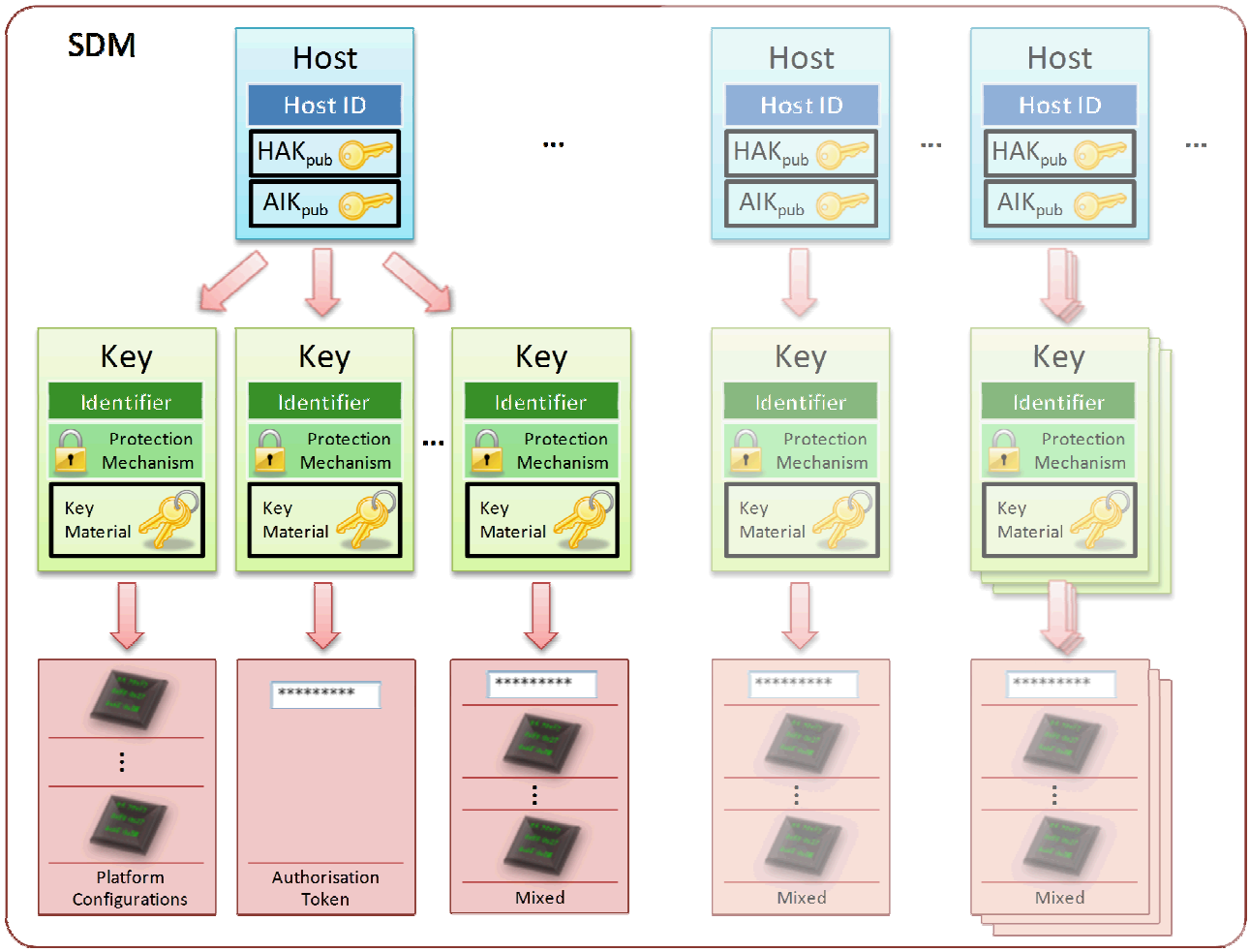

Figure 8: The SDM data model

Data Type	Size [Bytes]	Description
Unique Identifier	20	**Unique Identifiers** are used to identify different SDMs, SDM hosts and protected keys associated with a specific SDM host. SDM and SDM Host identifiers should be globally unique within the SECRICOM infrastructure. Key identifiers must be unique for keys associated with a specific host.
RSA Public Key	512	The SDM requires **RSA Public Keys** for SDM host authentication and TPM Quote signature verification. Both keys are logically part of a SDM host data structure and are always 2048-bit RSA public keys.
RSA Private Key	512	The SDM uses only one **RSA Private Key**. This is the private part of the SDM Authentication Key (SAK) which is used during host to SDM session establishment. It is a 2048-bit RSA private key.
Key Material	1024	The primary task of the SDM is to protect keys and credentials. Key material is simply a byte array with a maximum size of 1024 bytes. This key material is only released if the host can prove that it is in a specific *trusted state*, or if it can supply an authorization token, or both.
Valid Platform Configuration	20	Also called Host Configuration or simply platform configuration. Keys that use the *Platform Configuration Protected* or *Authorized Platform Configuration Protected* key protection mechanism are only released to the host if it is in a *trusted state*. A trusted state is characterized by a 20-byte SHA-1 hash. A **Valid Platform Configuration** is just such a SHA-1 hash.
Authorization Token	20	**Authorization Tokens** (passwords) grant access to keys which are protected by the *Authorization Token Protected* or *Authorized Platform Configuration Protected* protection mechanisms. An authorization token is an array of bytes with a maximum length of 20.
Random Bytes	512	A number of bytes generated using the random number generator in the SDM. The SDM can generate a maximum of 512 bytes or 4096 bits one

			go.
Key Mechanism	**Protection**	1	Either *Platform Configuration Protected*, *Authorization Token Protected*, *Authorized Platform Configuration Protected*.

Table 1: Primitive data types of the SDM

SDM Host

Name	Type	Remarks
Unique host identifier	**Unique Identifier**	Must be globally unique.
Host Authentication Key (HAK)	**RSA Public Key**	
Attestation Identity Key (AIK)	**RSA Public Key**	Optional

A host is a data structure that is composed of a **unique identifier** and a *Host Authentication Key (HAK)* and may have an associated *Attestation Identity Key (AIK)*. The HAK and the AIK are both 2048-bit **RSA Public Keys**.

The HAK is necessary to establish an authenticated and encrypted session between the SDM and a specific host (cf. section 4). The AIK is the public part of the key used by a host's TPM to sign its TPM Quotes. If a host has a TPM and the host wants to access keys that are only available if the host is in a specific *trusted state*, this AIK must be set.

In this context the term host might be a bit confusing. A host is simply any entity that has access to the private part of the HAK and which has a corresponding host entry in an SDM. A host is capable of establishing a client session with the SDM and has associated keys which can be retrieved during such a session.

In an administration session the *Administration Host* may add new hosts, update the HAK and AIK of existing hosts and can remove hosts from the SDM. Hosts can only be removed from the SDM, if they are no longer associated with any keys; that is all keys of a host must be deleted before the host itself can be deleted. The *Administration Host* can also add new keys to a host, update the data of existing keys of a host, under certain condition remove a key and list all keys associated with a host.

SDM Keys

Name	Type	Remarks
Key identifier	**Unique Identifier**	Must be unique within the context of the host the key is associated with.
Protection mechanism type	**Key Protection Mechanism**	
Key data	**Key Material**	
Password	**Authorization Token**	Only exists if the key uses the *Authorization Token Protected* or *Authorized Platform Configuration Protected* protection mechanisms.

A SDM protected key is always associated with a host and is composed of at least three elements: A **Unique Identifier**, a **Key Protection Mechanism**, the actual **Key Material**, and under certain conditions an **Authorization Token**. The key identifier must be unique within the context of the host. The key data is the actual data protected by the SDM.

The protection mechanism specifies under which conditions a key is released to the host.

- The *ATP* protection mechanism releases a key if the correct authorization token (password) is provided.
- The *PCP* protection mechanism releases a key if the host is a *trusted state*.
- The *APCP* protection mechanism requires both an authorization token and that the host is in a *trusted state*.

If the key uses the ATP or APCP protection mechanism the key also contains the **Authorization Token**. In case the key is protected by the PCP or APCP protection mechanism the key should be associated with a set of *trusted states*. If a PCP or APCP protected key is not associated with a *trusted state*, it must not be possible for a client to retrieve the key.

In an administration session the *Administration Host* can add, read and modify and remove keys. Keys must always be associated with a host. Keys can only be removed if they contain no more **Valid Platform Configurations** (*trusted states*). In the case where the key uses the ATP or APCP protection mechanism the *Administration Host* may set the **Authorization Token** of the Key. In the case where the key is protected by the PCP or ACPC mechanisms the *Administration Host* can list all **Valid Platform Configurations** associated with the key, add new **Valid Platform Configurations**, and remove **Valid Platform Configurations**. It is not possible to directly modify a **Valid Platform Configuration**, but this functionality can be implemented using remove and add commands.

Administration Host

Name	Type	Remarks
Unique host identifier	**Unique Identifier**	
Host Authentication Key (HAK)	**RSA Public Key**	
Attestation Identity Key (AIK)	**RSA Public Key**	Mandatory
Administration Configuration	**Valid Host Configuration**	*Trusted state*

The SDM administration host data structure is a special case of an SDM host. The SDM administration host is composed of a **Unique Identifier**, two **RSA Public Keys**, and a **Valid Platform Configuration**. For SDM administration host both the HAK and the AIK must be set, because host platform attestation is an integral part of the administration session establishment protocol. The administration configuration is the *trusted state* in which an *Administration Host* must be able to administer the SDM.

In an administration session the *Administration Host* may update the SDM administration host data structure and thus set a new *Administration Host*, or update the trusted state of the *Administration Host*. Special care must be taken when changing values associated with the SDM administration host. If the SDM administration host data structure contains values that are not achievable, the SDM cannot be administered any more.

4. The SDM communication protocols

The SDM is physically connected to a host computing device. This host computing device or simply host is a computing device that provides a compatible hardware interface and implements the SDM communication protocol. A host with a TPM and a measured Hypervisor that is capable of providing a measured and isolated execution environment is called Trusted Docking Station (TDS). Currently only one manufacturer ships its TPMs with an *Endorsement Key (EK)* certificate, therefore only hosts equipped with an Infineon TPM can be TDS. A TDS can host several isolated execution environments, each a potential host to the SDM. The SDM must also provide services to devices that are not equipped with a TPM. The term host or host computing device is applicable for all these devices.

The SDM must provide a session based communication protocol in order to allow communication between a host and the SDM. The SDM must allow exactly one active session between itself and a host device at any given time. It is possible for different hosts to communicate with the SDM sequentially after each other, but not concurrently. According to the "Security requirements and specification for docking module (D4.1)" the communication between the SDM and a host must be authenticated and protected against eavesdropping.

4.1 Basic session establishment

The basic protocol flow to establish authentication and a shared secret is depicted in Figure 9. The protocol is based on the Needham, Schroeder & Lowe protocol. The terms SAK and HAK in the following protocol descriptions refer to the SDM Authentication Key (SAK) and the Host Authentication Key (HAK). Both are 2048-bit RSA key pairs. SAK_{Pub} and HAK_{Pub} denote the public part of the respective key pair. The protocol exchanges two random numbers which must be used only once. Such numbers are called nonce's. The size of the nonce guarantees a very high probability that the random number generator will not generate a random number twice, under the assumption that a proper random number generator is being used. The random number generator must therefore be cryptographically secure.

Name	Type	Description
$N_{Host}/N_{SDM}/N_{TDS}$	16-byte nonce	Random numbers that must only be used once, used for symmetric key derivation
$I_{Host}/I_{SDM}/I_{TDS}$	20-byte identifier	Unique identifiers of the hosts and SDMs
IV	20-byte nonce	Initialization Vector to be used with the AES CBC-mode encryption/decryption
N_{TPM}	20-byte nonce	Nonce generated by the SDM and used by the TPM in a Quote operation
SAK_{Pub}	2048-bit RSA public	Public part of the SDM Authentication Key

	key	
HAK_{Pub}	2048-bit RSA public key	Public part of the Host Authentication Key
k	16-byte AES key	Derived from N_{Host} and N_{SDM}

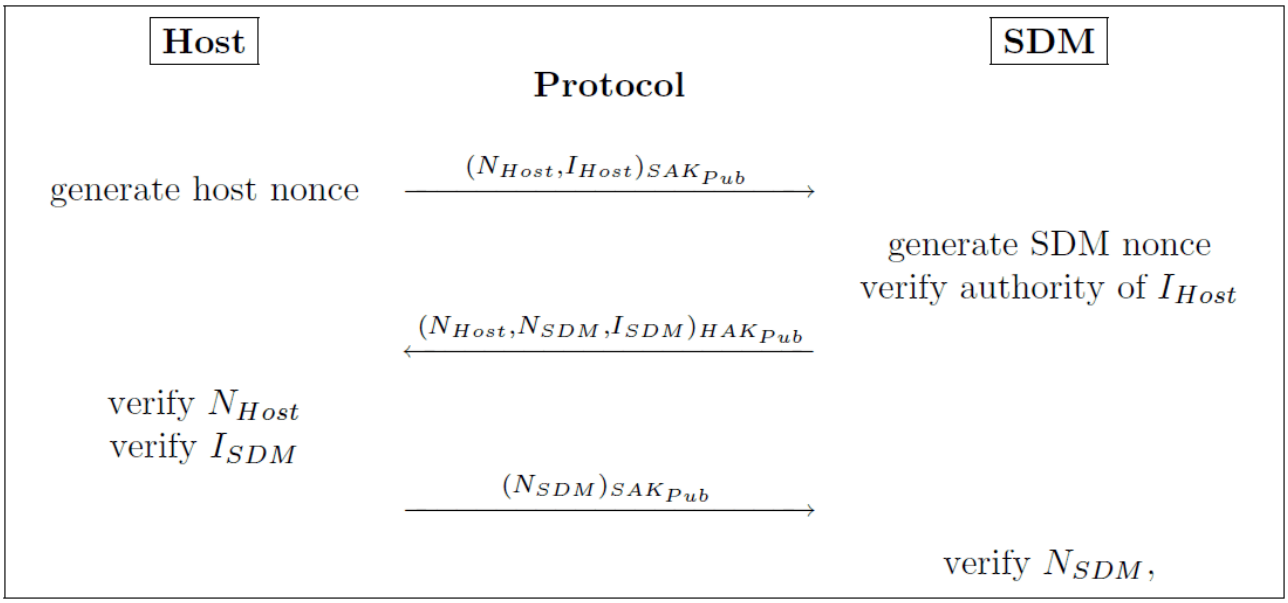

Figure 9: Authentication and secret exchange protocol based on the Needham, Schroeder & Lowe protocol

4.2 *Administration session*

The SDM provides two different services to the host. The first is the client interface which allows using the SDM key protection & release facilities. The second interface is the SDM administration interfaces which grants access to the setup and maintenance capabilities of the SDM.

As an administration session allows to change the mechanisms that govern key release it is evident that it must be protected against unauthorized access. Therefore, the SDM must mandate that the host which administrates it is in a well defined, previously specified state. This is achieved by using the same mechanism that governs key release. The protocol that establishes an administration session with the SDM is illustrated by Figure 10.

The administration interface can only be used by a TDS. The Host Authentication Key of the TDS must be exceedingly well protected, because the administration host has full access to all information stored inside the SDM.

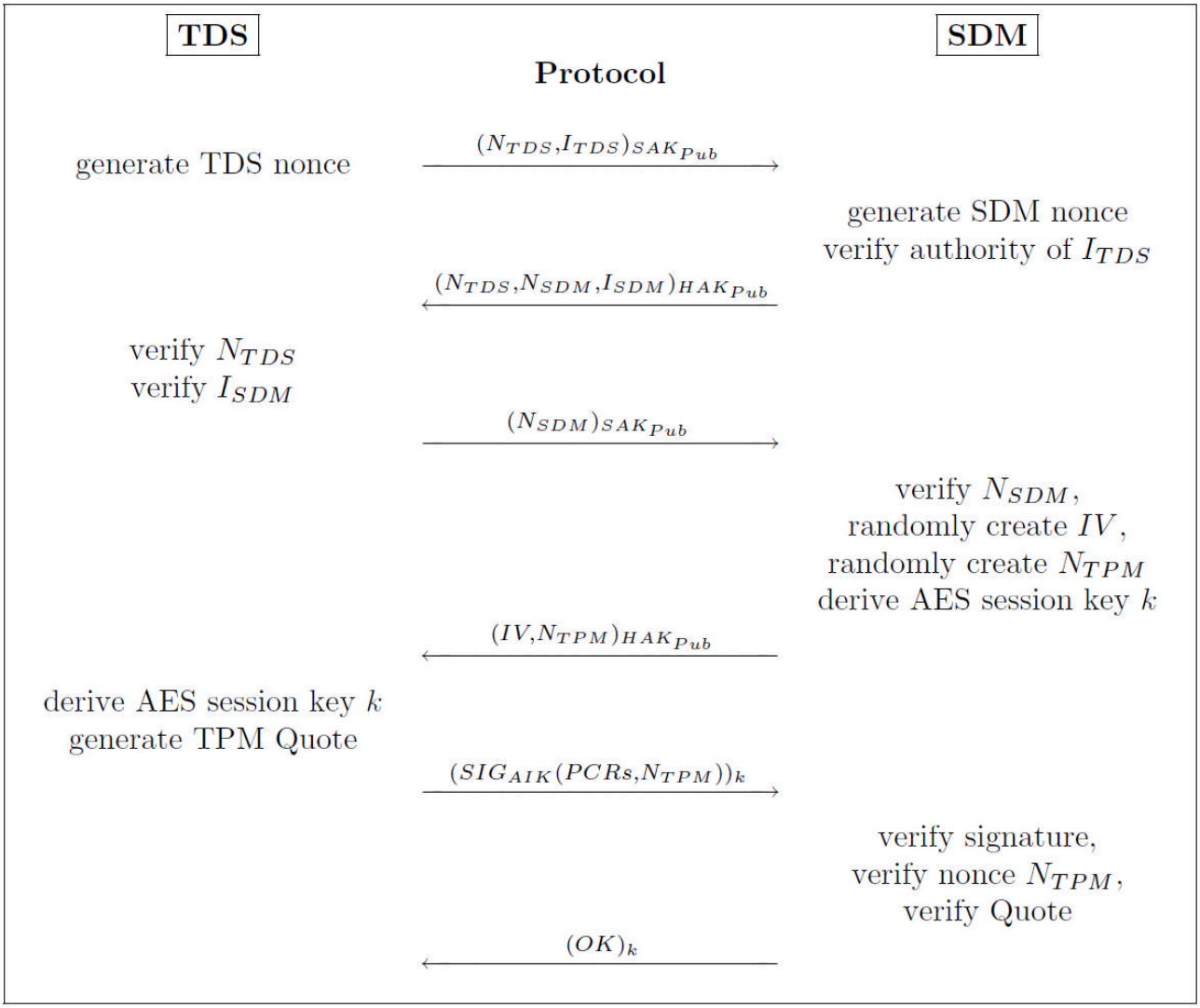

Figure 10: Administration session establishment protocol

4.3 Client Session

The primary focus of an SDM client session is to request and receive protected key material from the SDM. Keys may be protected by three different mechanisms. The first protection mechanism is based on an authorization token (password). If the client can specify the authorization token, the key is released to the client. The second mechanism is based on the host platform state. Only if the host platform state is equivalent to one of a set of previously specified states, the SDM key will be released. The last protection mechanism combines both former mechanisms: A key is only released if the host can provide both an authorization token and an attested platform configuration report which can certify that it is in a valid state.

A client session allows requesting several keys consecutively in addition to using the other capabilities the SDM provides (random number generator). The protection mechanism is specific to the keys. As the SDM must also support devices that are not equipped with a TPM it is not feasible to mandate platform attestation during establishment of a client session. The client session establishment process is shown in.

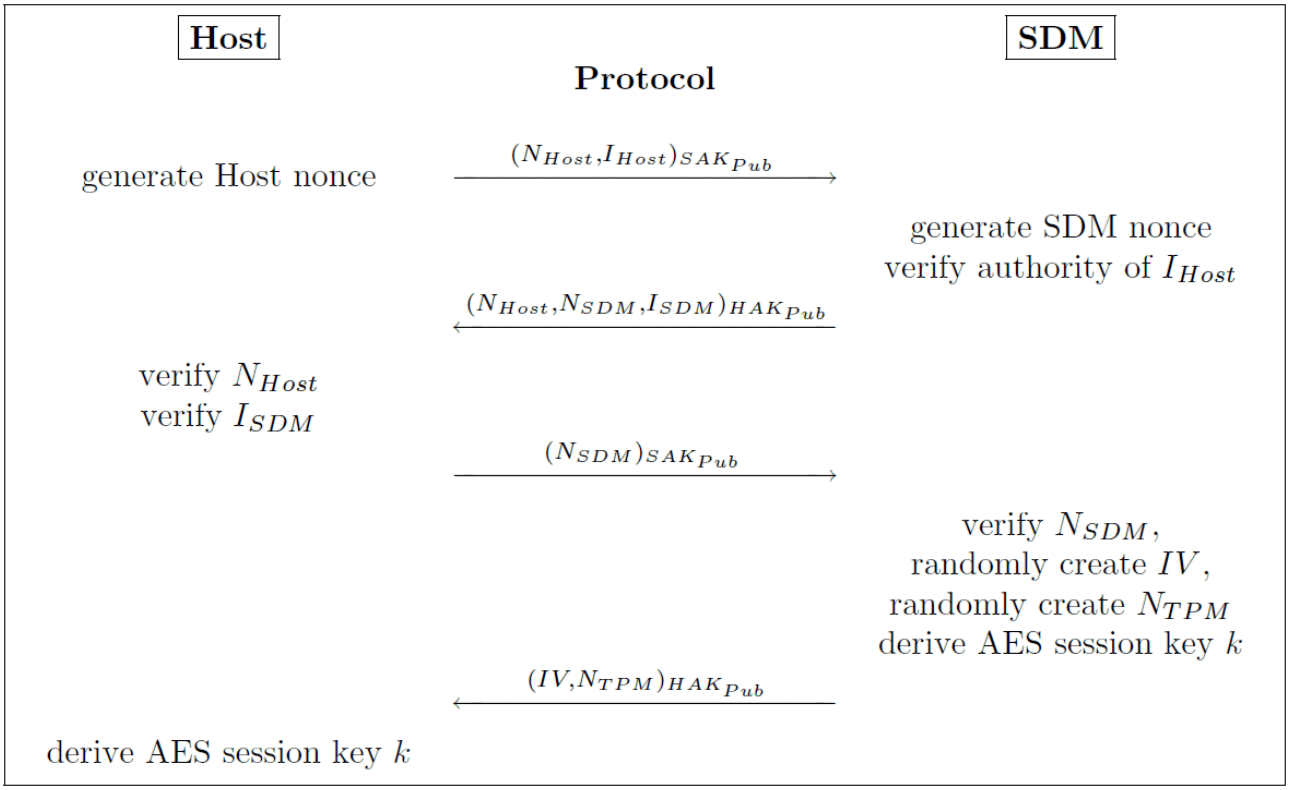

Figure 11: Client session establishment protocol

5. Client Commands

5.1 Client Session Related Commands

5.1.1 InitiateSession

`InitiateSession` indicates that a host wants to initiate a client session with the SDM. In a client session the host can query key information and test if its current platform configuration is valid.

The `InitiateSession` command is the first message in the client session establishment protocol. It sends the host principal, which uniquely identifies the host, and a host nonce to the SDM. The host nonce must be randomly generated and should not have been used for the same purpose before.

Command Structure

Name	Type	Description
Command	`byte`	INITIATE_SESSION
HostID	`SdmIdentifier`	The globally unique identifier of the host that wants to initiate a client session with the SDM.
HostNonce	`byte[16]`	A random number (nonce) generated by the host. The whole `InitiateSession` command is encrypted using the public part of the *SDM Authentication Key*. Thus, the SDM, by being able to decrypt the request and sending the random number back to the host, proofs that it possesses the private part of the *SDM Authentication Key* and thus authenticates itself.

5.1.2 EstablishSession

`EstablishSession` is the second SDM command in the client session establishment protocol and should establish the authenticity of the host. The `EstablishSession` command is encrypted using the public part of the *SDM Authentication Key*.

In an `EstablishSession` session command the host sends the SDM nonce back to the SDM. The SDM nonce is generated during the `InitiateSession` command and encrypted using the public key of the indicated host. Thus, the SDM knows that the host has access to the private part of the encryption key if the host can send back the nonce.

Command Structure

Name	Type	Description
Command	`byte`	ESTABLISH_SESSION
SDMNonce	`byte[16]`	A random number generated by the SDM and sent to the host as part of the response to the `InitiateSession` command. By sending back this number to the SDM as part of the `EstablishSession` command the host proofs that it possesses the

	private part of a specific Host Authentication Key and thus authenticates itself.

5.1.3 CloseSession

`CloseSession` terminates an existing client session with the SDM. The SDM must be in a client session for this command to work.

Command Structure

Name	Type	Description
Command	`byte`	CLOSE_SESSION

5.2 Key and Configuration Related Commands

5.2.1 CreateTpmNonce

`CreateTpmNonce` prompts the SDM to create a 20 byte nonce that must be used for a TPM Quote which is send to the SDM as part of a key release or platform state verification command.

Each SDM capability that relies on verifying the host devices state requires a TPM Quote of the host platform. To ensure the freshness of the TPM Quote the TPM Quote command allows for supplying 20 bytes of external data. This nonce must be created by the entity that relies on the TPM Quote. In this case this entity is the SDM.

Command Structure

Name	Type	Description
Command	`byte`	CREATE_NONCE

5.2.2 GetAuthKey

`GetAuthKey` retrieves an authorization token protected key from the SDM. The SDM provides three different protection mechanisms. The *Authorization Token Protected* key protection mechanism releases a key only if the correct `SdmToken`, that is a password, is specified.

Command Structure

Name	Type	Description
Command	`byte`	GET_KEY_AUTH
HostID	`SdmIdentifier`	The globally unique identifier of the host.
KeyID	`SdmIdentifier`	The key identifier. It must be unique per host.
AuthToken	`SdmToken`	The actual authorization token that unlocks the protected key.

5.2.3 GetConfigAuthKey

`GetConfigAuthKey` retrieves a platform configuration and authorization token protected key from the SDM. The SDM provides three different protection mechanisms. The *APCP* key

protection mechanism releases a key only if the platform can provide both an attested platform configuration report which certifies its valid state and the correct authorization token (password). This command is part of the SDM client interface.

Retrieving a key which is protected by a set of valid platform configurations is a two step process. First the client must request a nonce which it must then proceed to use in the TPM Quote operation that certifies the platform state. This TPM Quote can then be used with the next key release function to obtain a configuration protected key.

Command Structure

Name	Type	Description
Command	`byte`	GET_KEY_CONFIG_AUTH
HostID	`SdmIdentifier`	The globally unique identifier of the host.
KeyID	`SdmIdentifier`	The key identifier. It must be unique per host.
TPM Quote	`TpmQuote`	The TPM Quote that certifies the valid state of the platform.
AuthToken	`SdmToken`	The actual authorization token that unlocks the protected key.

5.2.4 GetConfigKey

`GetConfigKey` retrieves a key that is protected by a set of valid platform configurations. The SDM provides three different protection mechanisms. The *PCP* key protection mechanism releases a key only if the platform can certify its *trusted state*. This command is part of the SDM client interface.

Retrieving a key which is protected by a set of valid platform configurations is a two step process. First the client must request a nonce which it must then proceed to use in the TPM Quote operation that certifies the platform state. This TPM Quote can then be used with the next key release function to obtain a configuration protected key.

Command Structure

Name	Type	Description
Command	`byte`	GET_KEY_CONFIG
HostID	`SdmIdentifier`	The globally unique identifier of the host.
KeyID	`SdmIdentifier`	The key identifier. It must be unique per host.
TPM Quote	`TpmQuote`	The TPM Quote that certifies the valid state of the platform.

5.2.5 VerifyConfig

`VerifyConfig` tests if the host is in a *trusted state* that could affect the release of a specific key that is protected by a set of valid platform configurations. The SDM provides three different protection mechanisms. The *PCP* key protection mechanism releases a key only if the platform can certify its *trusted state*. This command is part of the SDM client interface.

Verifying the *trusted state* of the host platform by comparing its platform configuration with a set of valid platform configurations is a two step process. First the client must request a nonce which it must then proceed to use in the TPM Quote operation that certifies the platform state. This TPM Quote can then be used with the next verification request.

Command Structure

Name	Type	Description
Command	`byte`	VERIFY_CONFIG
HostID	`SdmIdentifier`	The globally unique identifier of the host.
KeyID	`SdmIdentifier`	The key identifier. It must be unique per host. Valid platform configurations are always associated with specific keys. It is therefore necessary to specify host identifier and key identifier to uniquely identify the key.
TPM Quote	`TpmQuote`	The TPM Quote that certifies the trusted state of the platform.

5.2.6 VerifyConfigAuth

`VerifyConfigAuth` tests if the host is in a *trusted state* and can provide an authorization token that could affect the release of a specific key. The SDM provides three different protection mechanisms. The *APCP* key protection mechanism releases a key only if the platform can prove its valid configuration and can provide the correct authorization token (password). This command is part of the SDM client interface.

Verifying the *trusted state* of the host platform by comparing its platform configuration with a set of valid platform configurations is a two step process. First the client must request a nonce that it must then proceed to use in the TPM Quote operation that certifies the platform state. This TPM Quote can then be used with the next verification request.

Command Structure

Name	Type	Description
Command	`byte`	VERIFY_CONFIG_AUTH
HostID	`SdmIdentifier`	The globally unique identifier of the host.
KeyID	`SdmIdentifier`	The key identifier. It must be unique per host. Valid platform configurations are always associated with specific keys. It is therefore necessary to specify host identifier and key identifier to uniquely identify the key.
TPM Quote	`TpmQuote`	The TPM Quote that certifies the valid configuration (trusted state) of the platform.
AuthToken	`SdmToken`	The authorization token that together with the trusted state of the platform would unlock the protected key.

5.3 *Miscellaneous Commands*

5.3.1 GetRandom

`GetRandom` requests a number of random bytes from the SDM's hardware true random number generator. The number is limited to **4096 bits** (512 bytes). This command is part of the client interface of the SDM.

Command Structure

Name	Type	Description
Command	byte	GET_RANDOM
Number	int	The number of bytes that should be generated by the SDM. The number must be between 1 and 512.

6. Initialization Commands

6.1 *InjectIdentity*

`InjectIdentity` is used on freshly fabricated SDMs to inject them with a unique identifier and a unique identity key. This is the first step in the SDM life cycle. This command gives the SDM its identity. The second step is to set an administration host using the `InjectAdminHost`. The third step is to use the administration interface to configure the SDM.

`InjectIdentity` sets the unique identifier of the SDM and the unique *SDM Authentication Key (SAK)*. The *SDM Authentication Key* is a 2048-bit RSA key-pair that is used to authenticate the SDM. The `InjectIdentity` process is not protected by an authenticated and encrypted session, as at that point the SDM does not possess any information that would allow such protection mechanisms.

This command will only be accepted by the SDM if it is in the `UNINITIALIZED` state. This is the first command that must be called on a new SDM.

Command Structure

Name	Type	Description
Command	`byte`	INJECT_IDENTITY
SDMID	`SdmIdentifier`	The globally unique identifier of the SDM.
SAK modulus	`byte[256]`	The modulus of the SDM Authentication Key. The SDM Authentication Key is a 2048-bit RSA public key-pair, where the private key is only known to the SDM. The SDM requires this key-pair for authentication.
SAK public exponent	`byte[256]`	The public exponent of the SDM Authentication Key.
SAK private exponent	`byte[256]`	The private exponent of the SDM Authentication Key.

6.2 *InjectAdminHost*

`InjectAdminHost` sets the administration host of the SDM. This command is the second command in the SDM life cycle. When an SDM is freshly fabricated the `InjectIdentity` command is issued to set the unique identifier and the *SDM Authentication Key* of the SDM. The second step is to specify an administration host. The administration host is a special SDM host that is allowed to add new hosts, new protected material, and so on. The initial administration host is set using this `InjectAdminHost` command.

`InjectAdminHost` sets the unique identifier, the public part of the *Host Authentication Key*, the public part of the *Attestation Identity Key* and a single *trusted state* of the administration host. The `InjectAdminHost` process is not protected by an authenticated

and encrypted session, as at that point the SDM does not possess any information that would allow such protection mechanisms.

This command will only be accepted by the SDM if it is in the `PROGRAMMABLE` state. This command must be called after the `InjectIdentity` command.

Command Structure

Name	Type	Description
Command	`byte`	INJECT_ADMIN_HOST
HostID	`SdmIdentifier`	The globally unique identifier of the administration host.
HAK modulus	`byte[256]`	The modulus of the public part of the Host Authentication Key. The Host Authentication Key is a RSA public key pair, where the private key is only known to the SDM host. The SDM requires the public key for administration session authentication.
HAK public exponent	`byte[256]`	The public exponent of the public part of the Host Authentication Key.
AIK modulus	`byte[256]`	The modulus of the public part of the Attestation Identity Key. The private part of the Attestation Identity Key is used by the administration host's TPM to sign the platform configuration report (TPM Quote) which is a strict requirement of the administration session establishment protocol.
AIK public exponent	`byte[256]`	The public exponent of the public part of the Attestation Identity Key.
HostConf	`HostConfiguration`	The valid configuration (trusted state) the administration host must be in to administer the SDM.

6.3 DecomissionSdm

Decomissions the SDM. This command can only be issued during an administration session. It clears the administration host data, deletes the unique identity of the SDM, and immediately closes the current administration session. A prerequisite for this command is that all protected data has been removed, i.e. the SDM is "empty".

Command Structure

Name	Type	Description
Command	`byte`	DECOMISSION_SDM

7. Administrative Commands

7.1 Administration Session Related Commands

7.1.1 InitiateAdminSession

`InitiateAdminSession` indicates to the SDM that the host wants to initiate an administration session with the SDM. An administration session enables the host to program new data, and to update and remove old keys and valid configurations of the SDM. Only hosts which are in the valid platform configuration that was specified using the `InjectAdminHost` command are allowed to program the SDM. Only one concurrent session with the SDM is possible, be it administrative or a client session.

Command Structure

Name	Type	Description
Command	`byte`	INITIATE_ADMIN_SESSION
HostID	`SdmIdentifier`	The globally unique identifier of the host that wants to initiate an administration session with the SDM.
HostNonce	`byte[16]`	A random number (nonce) generated by the host. The whole `InitiateAdminSession` command is encrypted using the public part of the *SDM Authentication Key*. Thus, the SDM, by being able to decrypt the request and sending the random number back to the host, proofs that it possess the private part of the *SDM Authentication Key* and thus authenticates itself.

7.1.2 NegotiateAdminSession

`NegotiateAdminSession` is the second SDM command in the administration session establishment protocol and should establish the authenticity of the host.

In a `NegotiateAdminSession` command the host sends the SDM nonce back to the SDM. The SDM nonce is generated during the `InitiateAdminSession` command and encrypted using the public key of the indicated host. Thus, the SDM knows that the host has access to the private part of the encryption key if the host can send back the nonce.

Command Structure

Name	Type	Description
Command	`byte`	NEGOTIATE_ADMIN_SESSION
SDMNonce	`byte[16]`	A random number generated by the SDM and sent to the host as part of the response to the `InitiateAdminSession` command. By sending back this number to the SDM as part of the `NegotiateAdminSession` command the host proofs that it possesses the private part of a specific *Host Authentication Key* and thus authenticates itself.

7.1.3 EstablishAdminSession

`EstablishAdminSession` is the final step in the administration session establishment protocol of the SDM. An `EstablishAdminSession` command sends a signed platform configuration report, a so-called TPM quote to the SDM. The TPM Quote includes a SDM generated nonce that was previously sent to the host to be included in the TPM quote. This step should proof to the SDM that the host platform is in a valid configuration (*trusted state*)

Command Structure

Name	Type	Description
Command	`byte`	ESTABLISH_ADMIN_SESSION
TPMQuote	`TpmQuote`	The platform configuration report, which includes the platform configuration and an SDM generated nonce and is signed with a specific *Attestation Identity Key*.

7.1.4 CloseAdminSession

`CloseAdminSession` terminates an existing administration session with the SDM. The SDM must be in an administration session for this command to work.

Command Structure

Name	Type	Description
Command	`byte`	CLOSE_ADMIN_SESSION

7.2 SDM Host Related Commands

A host is the SDM representation of a platform that wants to use the services of an SDM. A host object consists of a unique identifier, a *Host Authentication Key (HAK)*, and an *Attestation Identity Key (AIK)*. The unique identifier must be unique for all hosts that are part of the SDM protected infrastructure, it should be globally unique. The HAK is used to authenticate a session between a host and the SDM. The AIK is used by the host's TPM to sign a platform report. A host manages SDM protected keys; that is an SDM protected key is always associated with a host.

7.2.1 AddHost

`AddHost` either adds a new host to the SDM or updates an existing host in the SDM with new information.

Command Structure

Name	Type	Description
Command	`byte`	ADD_HOST
HostID	`SdmIdentifier`	The globally unique identifier of the host.

HAK modulus	`byte[256]`	The modulus of the public part of the *Host Authentication Key*. The *Host Authentication Key* is a RSA public key pair, where the private key is only known to the SDM host. The SDM requires the public key for session authentication.
HAK public exponent	`byte[256]`	The public exponent of the public part of the *Host Authentication Key*.
AIK modulus	`byte[256]`	The modulus of the public part of the *Attestation Identity Key*. The private part of the *Attestation Identity Key* is used by the host's TPM to sign the platform configuration report (TPM Quote). In order to release a Platform Configuration Protected key the SDM must verify these TPM Quotes and therefore requires the public part of the AIK.
AIK public exponent	`byte[256]`	The public exponent of the public part of the *Attestation Identity Key*.

7.2.2 GetHost

`GetHost` retrieves information associated with an SDM host. The associated information is the public part of the *Host Authentication Key* and the public part of the *Attestation Identity Key*.

Command Structure

Name	Type	Description
Command	`byte`	GET_HOST
HostID	`SdmIdentifier`	The globally unique identifier of the host.

7.2.3 ListHosts

`ListHosts` retrieves a list of all hosts stored in the SDM. The command returns a set of unique host identifiers.

Command Structure

Name	Type	Description
Command	`byte`	LIST_HOSTS

7.2.4 RemoveHost

`RemoveHost` removes an SDM host from the SDM. The host must be empty of all keys before it can be successfully removed!

Command Structure

Name	Type	Description
Command	`byte`	REMOVE_HOST
HostID	`SdmIdentifier`	The globally unique identifier of the host to remove.

7.2.5 SetAdministrationHost

`SetAdministrationHost` updates the existing administration host of the SDM with new information.

The administration host is the SDM representation of the platform that is allowed to administer the SDM. An administration host consists of a unique identifier, a *Host Authentication Key (HAK)*, an *Attestation Identity Key (AIK)*, and a valid platform configuration.

The unique identifier must be unique for all hosts that are part of the SDM protected infrastructure, it should be globally unique. The HAK is used to authenticate a session between the host and the SDM. The AIK is used by the host's TPM to sign a platform report. The valid platform configuration defines the *trusted state* the administration host must be in, to be allowed to administer the SDM.

Command Structure

Name	Type	Description
Command	`byte`	SET_ADMIN_HOST
HostID	`SdmIdentifier`	The globally unique identifier of the administration host.
HAK modulus	`byte[256]`	The modulus of the public part of the *Host Authentication Key*. The *Host Authentication Key* is an RSA public key pair, where the private key is only known to the SDM host. The SDM requires the public key for administration session authentication.
HAK public exponent	`byte[256]`	The public exponent of the public part of the *Host Authentication Key*.
AIK modulus	`byte[256]`	The modulus of the public part of the *Attestation Identity Key*. The private part of the *Attestation Identity Key* is used by the administration host's TPM to sign the platform configuration report (TPM Quote) which is a strict requirement of the administration session establishment protocol.
AIK public exponent	`byte[256]`	The public exponent of the public part of the *Attestation Identity Key*.
HostConf	`HostConfiguration`	The valid configuration (*trusted state*) the administration host must be in to administer the SDM.

7.3 Key Related Commands

The primary task of the SDM is to protect keys. The SDM knows three different key protection mechanisms. The first type is called *Platform Configuration Protected* or PCP in short. As the name suggest, for this protection mechanism the key is bound to a specific

set of valid platform configurations so-called *trusted states*. Only, if the host platform is in a valid configuration, the key is released.

The second type is named *Authorization Token Protected* or ATP. ATP protected keys are released if a given authorization token (password) matches the authorization token associated with the key. The last type *Authorized Platform Configuration Protected* (APCP) releases keys to the host, if the host is in a valid configuration and can supply an authorization token.

7.3.1 AddKey

`AddKey` adds a new or updates an existing SDM protected key. A key is always associated with an SDM host. Both the unique key and the unique host identifiers together not only uniquely identify the key, but also allow lookup of a key within the SDM. The SDM must be in an administration session for this command to succeed.

An SDM protected key is initialized with an identifier (`SdmIdentifier`), a protection mechanism type and the actual data that should be protected (`SdmKeyMaterial`). The actual configuration data for the key protection mechanism (valid platform configurations and/or an authorization token) must be provided using the `AddHostConfiguration` and/or the `SetAuthorizationToken` commands.

Command Structure

Name	Type	Description
Command	`byte`	ADD_KEY
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.
ProtType	`SdmKey.Type`	The protection mechanism type. The protection mechanism can either be *Platform Configuration Protected (PCP)*, *Authorization Token Protected (ATP)*, and *Authorized Platform Configuration Protected (APCP)*.
KeyMat	`SdmKeyMaterial`	The data the SDM should protect with the chosen key protection mechanism. It is expected that this data will mostly be comprised of cryptographic key material, hence the name. Currently the size of the protected data is limited to 1024 bytes.

7.3.2 GetKey

`GetKey` retrieves an existing SDM protected key from the SDM. A key is always associated with an SDM host. Both the unique key and the unique host identifiers together not only uniquely identify the key, but also allow lookup of a key within the SDM. The SDM must be in an administration session for this command to succeed.

This command retrieves the basic information associated with an `SdmKey`: the key protection mechanism and the actual protected key material. To manage the protection

mechanism data (valid platform configurations and/or an authorization token) a set of commands exists.

Command Structure

Name	Type	Description
Command	`byte`	GET_KEY
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.

7.3.3 ListKeys

`ListKeys` retrieves a list of protected keys associated with a specific host in the SDM. A key is always associated with an SDM host. Therefore, in order to list the keys, the host identifier must be specified. The SDM must be in an administration session for this command to succeed.

Command Structure

Name	Type	Description
Command	`byte`	LIST_KEYS
HostID	`SdmIdentifier`	The globally unique identifier of the host for which the list of keys should be retrieved.

7.3.4 RemoveKey

`RemoveKey` deletes an existing, empty SDM protected key from the SDM. The key must not contain any valid platform configurations otherwise the remove operation will fail. A key is always associated with an SDM host. Both the unique key and the unique host identifiers together not only uniquely identify the key, but also allow lookup of a key within the SDM. The SDM must be in an administration session for this command to succeed.

Command Structure

Name	Type	Description
Command	`byte`	REMOVE_KEY
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.

7.4 SetAuthorizationToken

`SetAuthorizationToken` sets a new or overwrites an existing authorization `SdmToken` for a specific key protected by the SDM. Both the unique key and the unique host identifiers together not only uniquely identify the key, but also allow lookup of a key within the SDM.

The key identified by the host identifier/key identifier pair must be of type ATP or APCP (see below). In contrast to valid platform configurations, a key can only have a single authorization token. The SDM must be in an administration session for this command to succeed.

Command Structure

Name	Type	Description
Command	`byte`	SET_AUTH_TOKEN
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID,KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.
AuthToken	`SdmToken`	The new authorization token (password) for the specified key.

7.5 Trusted State Related Commands

The SDM protects key material. Two of the protection mechanisms provided by the SDM only grant the release of a key, if the host platform is in a valid platform configuration. A valid platform configuration is therefore always related to a specific key. A specific key is always associated with a specific host. Therefore, to modify (add, remove) and also to test for the existence of a specific valid platform configuration, three inputs are required by the SDM; a unique host identifier, a unique key identifier and the actual valid platform configuration.

7.5.1 AddHostConfiguration

`AddHostConfiguration` adds a new valid platform configuration (*trusted state*) to the set of valid platform configurations of a specific key which employs a platform configuration based key release scheme. In order to identify the target key, both a unique host and host-unique key identifier are necessary. If the valid platform configuration that should be added with this command is already stored for the specified key, this command will have no effect. This command is part of the administrative interface and must be issued in an administrative session.

Command Structure

Name	Type	Description
Command	`byte`	ADD_HOST_CONFIGURATION
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.
ValidConf	`HostConfiguration`	A new valid platform configuration (*trusted state*) that should be added to the set of trusted states that

	govern the release of a specific key.

7.5.2 ContainsHostConfiguration

`ContainsHostConfiguration` tests if the list of valid platform configurations of a specific key which employs a platform configuration based key release scheme, contains a specific valid platform configuration (*trusted state*). In order to identify the target key, both a unique host and unique key identifier are necessary.

Command Structure

Name	Type	Description
Command	`byte`	CONTAINS_HOST_CONFIGURATION
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.
ValidConf	`HostConfiguration`	The valid platform configuration (*trusted state*) to test for in the set of trusted states that govern the release of a specific key.

7.5.3 ListHostConfigurations

`ListHostConfigurations` generates a list of valid platform configuration (*trusted state*) for a specific key which employs a platform configuration based key release scheme. In order to identify the target key, both a unique host and host-unique key identifier are necessary.

Command Structure

Name	Type	Description
Command	`byte`	REMOVE_HOST_CONFIGURATION
HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID, KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.

7.5.4 RemoveHostConfiguration

`RemoveHostConfiguration` removes a valid platform configuration from the set of valid platform configurations of a specific key which employs a platform configuration based key release scheme. In order to identify the target key, both a unique host and unique key identifier are necessary.

Command Structure

Name	Type	Description
Command	`byte`	REMOVE_HOST_CONFIGURATION

HostID	`SdmIdentifier`	The globally unique identifier of the host
KeyID	`SdmIdentifier`	The (HostID,KeyID) pair uniquely identifies a specific key within the protection of the SDM. Whereas the host identifier must be globally unique, the key identifier must be at least unique for all keys of that specific host.
ValidConf	`HostConfiguration`	The valid platform configuration (*trusted state*) to remove from the set of *trusted states*.

8. Hardware Requirements

The specification of the functions of the SDM allows inferring a list of required hardware components. The hardware specification given here is preliminary. The exact hardware configuration of the SDM will be specified as part of WP5 task **T5.2 (M9-M18): Design of the Secure Docking Module** (IFX, TUG, CEA, HIT). The list of required components that have been identified so far is as follows:

1. A processor and associated memory

   A small processor (relative to the SDM) with a moderate amount of RAM is required to control the SDM subcomponents. This combination may also implement some or all of the SDM functionality in software.

2. An RSA module

   The RSA module serves two purposes. Its primary purpose is to verify the signatures on the TPM *Quote* operations. Its secondary function is to help establishing an authenticated and encrypted communications channel between the SDM and its host computing device. To fulfil these requirements the RSA module must be capable of encryption, decryption, and creation and verification of digital signatures.

3. A SHA-1 hash module

   A TPM *Quote* is a digitally signed platform configuration report. The actual platform configuration is represented by a set of SHA-1 hashes. These SHA-1 hashes are stored in so called PCRs inside the TPM. To minimize the size of the *Quote* a SHA-1 hash of a selected set of PCRs is computed and subsequently signed by the TPM. Depending on the exact details of the key release protocol it might be necessary to recalculate the SHA-1 hash of the selected set of PCRs. The symmetric communication session key is also derived from the nonces using the SHA-1 hash module.

4. A True Random Number Generator

   The True Random Number Generator is required to generate numbers that are only used once, so called *nonces*. The key release protocol for example requires an SDM created nonce to guarantee the freshness of the Quote generated by the TPM in the TDS. Also the host to SDM communication protocol requires a randomly created Initialization Vector (IV). The random number generation functionality is also made available to hosts as part of the functional specification.

5. A symmetric cryptographic primitive

   The necessity of an authenticated and encrypted communications channel requires a symmetric cryptographic primitive to encrypt the communication. An

example of such a primitive is the Advanced Encryption Standard (AES) block cipher. The wide acceptance of AES as a cryptographic standard and the fact that there are no known exploitable weaknesses makes it AES a valid choice for this purpose.

6. A non-volatile storage for keys and platform configurations

   The primary purpose of the SDM is to protect key material for asymmetric cryptography. This key material is stored in a non-volatile memory on the SDM. The SDM must also store a set of valid platform configurations (*trusted states*).

7. A physical interface

   The physical interface defines how the SDM is physically connected to the TDS. Several considerations factor into the choice of a physical interface for the SDM. These considerations are security, compatibility and bandwidth. Thanks to the end-to-end secure channel security requirement the relevance of the physical interface to the security of the SDM is limited.

   The SDM's primary function is to protect key material. This key material must be released whenever client software requires it. Therefore, the bandwidth requirement depends on the amount of transferred data per key release and the number of key releases per power cycle. The amount of transferred data is estimated to be about a few kilo bytes depending on the size of the key material. Once the SDM releases a key, the client software has full control over it. Therefore, it can be expected that client software initiates only one key release sequence per key and only once per life cycle. For these reasons, bandwidth is no limiting factor in the selection of the physical interface of the SDM.

   The last point to consider for the interface is compatibility. The SDM should also provide limited services to mobile host devices and therefore a MicroSD interface variant of the SDM will be considered as part of the design process. For non mobile host devices, the current aim is to provide a USB interface.

8.1 Tamper Resilience

So in war, the way is to avoid what is strong and to strike at what is weak. **Sung Tzu**

As was explained in [2.1] the security of the SDM is directly bound to the security provided by the host platform's TPM. The platform attestation process which is used to prove to the SDM that the host is in a *trusted state* and which governs the release of key material has two dependencies. It relies on the Chain of Trust and the security of the attestation process.

The Chain of Trust is an uninterrupted chain of measurements starting at the Core Root of Trust for Measurement, which is either a BIOS component or an authenticated code module up to the running application. It is built on the principle of measuring before executing. For a more detailed explanation refer to section 2.1. The security Chain of Trust depends on both the PC host platform and the TPM. A PC platform provides very limited

to no protection against hardware attacks. Therefore, the aim of *Trusted Computing* is to protect a PC platform against software attacks [Gra06].

The security of the attestation mechanism depends on the Core Root of Trust for Reporting. The Core Root of Trust for Reporting is rooted in the security of the *Attestation Identity Key*. The *Attestation Identity Key* is a 2048-bit RSA key pair. Its private part is used to sign the platform configuration report (TPM Quote). The private part of the *Attestation Identity Key* never leaves the TPM unencrypted. The security of the *Attestation Identity Key* has again two dependencies. The first is that it is intractable to break the RSA asymmetric cryptography scheme and the second that the key cannot be extracted from the TPM.

As both the Core Root of Trust for Measurement and the Core Root of Trust for Reporting rely on the security of the TPM, the SDM should provide the same protection to the data it guards, as does the TPM. A *Protection Profile* [TCG08] for the TCG PC Client Specific Trusted Platform Module exists as well as a certification report [BSI08] for this *Protection Profile* by the German Federal Office for Information Security (BSI).

8.2 SDM Security Considerations

The security design of the SDM may take the Protection Profile for a TPM as a basis as some system aspects are comparable. Referenced from the TCG protection profile for PC Client Specific TPMs, the SDM can be described as hardware, firmware and/or software that implements the defined functionality. The used primitives for the SDM include cryptographic algorithms for authentication and data de-/encryption, random number generation and a hash algorithm. Primarily, there are no cryptographic services that are provided as functionality (i.e. encryption of user data, signing of user data, etc.). This eases the system design in a way, that leakage of security relevant information is somewhat restricted.

Threat definition for the TPM also fits to the security considerations for the SDM. Following is a list of threats with descriptions for the SDM.

- Compromise: in the context of an SDM this means getting access to communication keys without having the access rights therefore
- Bypass: getting access to security relevant data like keys without using the defined functionality of the SDM
- Hack Crypto:hacking the cryptographic algorithm as such
- Hack Physical: hacking of the cryptographic operation or other security relevant data with intact strength of the algorithm; these physical attacks can be attacks on non volatile memories as well as implemented logic
- Import/Export: an attacker may import erroneous or incorrect data to invalidate the communication keys of the system (DoS attack)

- Intercept: an attacker may intercept the communication to and from the SDM to gain knowledge about secrets or to try to use the gained information to perform a replay attack
- Malfunction: relevant assets may be disclosed to an attacker by malfunctions of the SDM.
- Modify: unauthorized modification of data may lead to numerous system issues.
- Replay: get hold of authentication or identification data by intercepting some communication and use this information for a so called "Replay Attack"

Similar to the threat definition, security objectives are quite similar for an SDM compared to a TPM.

- Crypto_Key_Man The SDM must manage cryptographic keys in a secure manner including releasing the keys only after a secure state has been proven to the SDM.
- DAC The SDM must control and restrict client entity access to the SDM protected capabilities and shielded location in accordance with a specified access control policy.
- Export When data (keys) are exported outside the SDM, it must securely protect the confidentiality and the integrity of the data.
- Fail_Secure The SDM must enter a secure failure mode in the event of a failure.
- I&A The SDM must identify all client entities, and shall authenticate the claimed identity before granting a client entity access to the SDM facilities.
- Limit_Actions_Auth The SDM must restrict the actions a client entity may perform before the SDM verifies the identity of the client entity.
- Single_Auth The SDM must provide a single client entity authentication mechanism and require re-authentication to prevent "replay" and "man-in-the-middle" attacks.
- Tamper_Resistance The SDM must resist physical tampering of the security functions by hostile users.

8.3 On the use of SHA-1

We are aware that some vulnerabilities of the SHA-1 hash function are known. We are, however, forced to use this hash function as long as the TPM relies on it. In case of a future change of the TPM specifications that calls for a different hash function it is strongly advised, or even a must, to change the SDM hash function as well. This should be taken

into account during the design phase. A software implementation should be considered to allow for algorithm agility.

9. Glossary

AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining
CRTM	Core Root of Trust for Measurement
IV	Initialization Vector
PCR	Platform Configuration Register
SDM	Secure Docking Module
SDS	Secure Docking Station – a combination of TDS and SDM
SHA	Secure Hash Algorithm
TCG	Trusted Computing Group
TDS	Trusted Docking Station
TPM	Trusted Platform Module
VMM	Virtual Machine Monitor (Hypervisor)

10. References

[D2.1] Kocis et al. SECRICOM – Analysis of external and internal system requirements. Deliverable report D2.1, the SECRICOM project, February 2009.

[D4.1] Simo et al. SECRICOM – Security requirements and specification for docking station module

[BSI08] Federal Office for Information Security (BSI), BSI-CC-PP-0030-2008, Certification Report, Common Criteria Protection Profile for PC Client Specific Trusted Platform Module Family 1.2; Level 2, Version 1.1 developed by Trusted Computing Group, 2008.

[Gra06] David Grawrock. The Intel Safer Computing Initiative Building Blocks for Trusted Computing. Richard Bowles, 2006.

[TCG08] Trusted Computing Group, Protection Profile, PC Client Specific Trusted Platform Module, TPM Family 1.2; Level 2, Version: 1.1; July 10, 2008

Appendix A High Level Java Library Documentation

Package at.iaik.secricom.sdm

Interface Summary		Page
SdmKey	`SdmKey` is the base interface for all types of SDM keys and provides methods to obtain the key identifier and the actual key material.	62
SdmSession	An `SdmSession` (pronounced S-D-M Session) enables a client to communicate with the SDM.	71
SdmStatus	`SdmStatus` indicates the operation status of the SDM.	77
SdmVersion	`SdmVersion` defines the version of the SDM.	82
SecureDockingModule	`SecureDockingModule` manages access to the functionalities provided by a *Secure Docking Module*.	83

Class Summary		Page
SdmIdentifier	Uniquely identifies an SDM specific object.	54
SdmKeyMaterial	`SdmKeyMaterial` is information protected by the SDM.	65
SdmRandomBytes	`SdmRandomBytes` represents an array of random bytes generated by the SDM's hardware random number generator.	69
SdmToken	`SdmToken` are authorization tokens that grant access to certain keys.	79
TpmNonce	`TpmNonce` represents a 20 bytes long nonce used in a TPM quote operation.	87
TpmQuote	`TpmQuote` represents a signed TPM quote info data structure.	89
Util	`Util` is a set of utility methods that support the SDM interface with functions like byte[] pretty printing, a factory method for TPM quote info data structures etc..	91

Enum Summary		Page
SdmKey.Type	Protection mechanism `Type` of a key.	64
SdmStatus.Status	`Status` represents the current status of the SDM.	78

Exception Summary		Page
SdmException	Indicates a problem related to the Secure Docking Module and forms the basis for the SDM exception hierarchy.	52
SdmInvalidKeyTypeException	SdmInvalidKeyTypeException indicates that an SDM operation failed because it operated on a key with the wrong protection mechanism type.	57
SdmInvalidParameterException	SdmInvalidParameterException indicates that a method of the SDM interface was called with an invalid parameter.	58
SdmIOException	SdmIoException indicates errors related to managing the persistent state of the SDM.	60
SdmObjectNotEmptyException	SdmObjectNotEmptyException indicates that removal of an object from the SDM failed because the specific object is not empty.	67
SdmUnknownKeyException	SdmUnknownKeyException indicates that an SDM operation failed because it tried to operate on a key that is not known to the SDM in the specified context.	81

Class SdmException

at.iaik.secricom.sdm

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └at.iaik.secricom.sdm.SdmException
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

SdmInvalidKeyTypeException, SdmInvalidParameterException, SdmIOException, SdmObjectNotEmptyException, SdmUnknownHostException, SdmUnknownKeyException

---

```
public class SdmException
extends Exception
```

Indicates a problem related to the Secure Docking Module and forms the basis for the SDM exception hierarchy.

Constructor Summary	Page
SdmException() Constructs a new SdmException with null as its detail message.	53

SdmException(String message) Constructs a new SdmException with the specified detail message.	53
SdmException(String message, Throwable cause) Constructs a new SdmException with the specified detail message and cause.	53
SdmException(Throwable cause) Constructs a new SdmException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	54

Constructor Detail

SdmException

```
public SdmException()
```

Constructs a new SdmException with null as its detail message. The cause cause is not initialized. A call to the initCause(Throwable) method may be used to initialize the cause after exception creation.

SdmException

```
public SdmException(String message,
                    Throwable cause)
```

Constructs a new SdmException with the specified detail message and cause.

Parameters:

message - the detail message

cause - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmException

```
public SdmException(String message)
```

Constructs a new SdmException with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

message - the detail message

SdmException

`public` **`SdmException`**`(Throwable cause)`

Constructs a new `SdmException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Class SdmIdentifier

at.iaik.secricom.sdm

`java.lang.Object`

└ **`at.iaik.secricom.sdm.SdmIdentifier`**

`public class` **`SdmIdentifier`**

`extends Object`

Uniquely identifies an SDM specific object. The SDM manages several different objects. These include hosts, keys, valid platform configurations, and authorization tokens. Some of these objects must be identified by a unique identifier. The `SdmIdentifier` performs this task.

A `SdmIdentifier` is subject to certain restrictions. Current restrictions are preliminary, and might change with the gradual development of the actual hardware specifications. Currently, an `SdmIdentifier` must not be longer than 20 bytes. Furthermore, identifiers are currently interpreted as ASCII encoded strings in the Java SDM library.

Constructor Summary	Page
SdmIdentifier`(byte[] uid)` Constructs a new `SdmIdentifier` with the specified `byte[]` unique identifier.	55
SdmIdentifier`(String uid)` Constructs a new `SdmIdentifier` with the specified `String` unique identifier.	55

	Method Summary	Page
boolean	**equals**`(Object obj)`	56
int	**getLength**`()`	56
byte[]	**getUniqueId**`()`	55
int	**hashCode**`()`	56
String	**toString**`()`	56

Constructor Detail

SdmIdentifier

```
public SdmIdentifier(String uid)
             throws SdmInvalidParameterException
```

Constructs a new `SdmIdentifier` with the specified `String` unique identifier. The given identifier must not be null and must not be longer than 20 bytes, when byte[] encoded using the ASCII `CharSet`.

Parameters:

`uid` - the unique identifier encoded as a `String`.

Throws:

`SdmInvalidParameterException` - if the given unique identifier is null or longer than 20 bytes, when ASCII encoded.

SdmIdentifier

```
public SdmIdentifier(byte[] uid)
             throws SdmInvalidParameterException
```

Constructs a new `SdmIdentifier` with the specified `byte[]` unique identifier. The given identifier must not be null and must not be longer than 20 bytes, when byte[] encoded using the ASCII `CharSet`.

Parameters:

`uid` - the unique identifier

Throws:

`SdmInvalidParameterException` - if the specified identifier is null or longer than 20 bytes.

Method Detail

getUniqueId

```
public byte[] getUniqueId()
```

Returns:

a defensive copy of the byte[] representing the unique id of the identifier.

---

getLength

```
public int getLength()
```

Returns:

the actual number of bytes required by this `SdmIdentifier`.

---

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

---

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class SdmInvalidKeyTypeException

at.iaik.secricom.sdm

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └at.iaik.secricom.sdm.SdmException
              └at.iaik.secricom.sdm.SdmInvalidKeyTypeException
```

All Implemented Interfaces:

Serializable

```
public class SdmInvalidKeyTypeException
extends SdmException
```

SdmInvalidKeyTypeException indicates that an SDM operation failed because it operated on a key with the wrong protection mechanism type.

Constructor Summary	Page
SdmInvalidKeyTypeException() Constructs a new SdmInvalidKeyTypeException with null as its detail message.	57
SdmInvalidKeyTypeException(String message) Constructs a new SdmInvalidKeyTypeException with the specified detail message.	58
SdmInvalidKeyTypeException(String message, Throwable cause) Constructs a new SdmInvalidKeyTypeException with the specified detail message and cause.	58
SdmInvalidKeyTypeException(Throwable cause) Constructs a new SdmInvalidKeyTypeException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	58

Constructor Detail

SdmInvalidKeyTypeException

```
public SdmInvalidKeyTypeException()
```

Constructs a new SdmInvalidKeyTypeException with null as its detail message. The cause cause is not initialized. A call to the initCause(Throwable) method may be used to initialize the cause after exception creation.

SdmInvalidKeyTypeException

```
public SdmInvalidKeyTypeException(String message,
                                  Throwable cause)
```

Constructs a new SdmInvalidKeyTypeException with the specified detail message and cause.

Parameters:

message - the detail message

cause - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmInvalidKeyTypeException

```
public SdmInvalidKeyTypeException(String message)
```

Constructs a new `SdmInvalidKeyTypeException` with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

`message` - the detail message

SdmInvalidKeyTypeException

`public` **`SdmInvalidKeyTypeException`**`(Throwable cause)`

Constructs a new `SdmInvalidKeyTypeException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Class SdmInvalidParameterException

at.iaik.secricom.sdm

`java.lang.Object`

`└java.lang.Throwable`

`└java.lang.Exception`

`└`at.iaik.secricom.sdm.SdmException

`└`**`at.iaik.secricom.sdm.SdmInvalidParameterException`**

All Implemented Interfaces:

Serializable

`public class` **`SdmInvalidParameterException`**

`extends` SdmException

`SdmInvalidParameterException` indicates that a method of the SDM interface was called with an invalid parameter. An invalid parameter exception is also thrown if the given parameter is null. This exception is not a runtime exception and thus forces the caller of the API explicitly consider these error cases.

Constructor Summary	Page
SdmInvalidParameterException() Constructs a new `SdmInvalidParameterException` with null as its detail message.	59

SdmInvalidParameterException(String message) Constructs a new SdmInvalidParameterException with the specified detail message.	60
SdmInvalidParameterException(String message, Throwable cause) Constructs a new SdmInvalidParameterException with the specified detail message and cause.	59
SdmInvalidParameterException(Throwable cause) Constructs a new SdmInvalidParameterException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	60

Constructor Detail

SdmInvalidParameterException

```
public SdmInvalidParameterException()
```

Constructs a new SdmInvalidParameterException with null as its detail message. The cause cause is not initialized. A call to the initCause(Throwable) method may be used to initialize the cause after exception creation.

SdmInvalidParameterException

```
public SdmInvalidParameterException(String message,
                                    Throwable cause)
```

Constructs a new SdmInvalidParameterException with the specified detail message and cause.

Parameters:

message - the detail message

cause - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmInvalidParameterException

```
public SdmInvalidParameterException(String message)
```

Constructs a new SdmInvalidParameterException with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

message - the detail message

SdmInvalidParameterException

`public` **`SdmInvalidParameterException`**`(Throwable cause)`

Constructs a new `SdmInvalidParameterException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Class SdmIOException

at.iaik.secricom.sdm

`java.lang.Object`

`└java.lang.Throwable`

`└java.lang.Exception`

`└`at.iaik.secricom.sdm.SdmException

`└`**`at.iaik.secricom.sdm.SdmIOException`**

All Implemented Interfaces:

Serializable

`public class` **`SdmIOException`**

`extends` SdmException

`SdmIoException` indicates errors related to managing the persistent state of the SDM. Information in the SDM (hosts, keys, and so on) are stored in some form of persistent storage. This exception indicates that writing or reading to/from this storage failed.

Constructor Summary	Page
SdmIOException`()` Constructs a new `SdmIOException` with null as its detail message.	61
SdmIOException`(String message)` Constructs a new `SdmIOException` with the specified detail message.	61
SdmIOException`(String message, Throwable cause)` Constructs a new `SdmIOException` with the specified detail message and cause.	61
SdmIOException`(Throwable cause)` Constructs a new `SdmIOException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	62

Constructor Detail

SdmIOException

```
public SdmIOException()
```

Constructs a new `SdmIOException` with null as its detail message. The cause cause is not initialized. A call to the `initCause(Throwable)` method may be used to initialize the cause after exception creation.

SdmIOException

```
public SdmIOException(String message,
                      Throwable cause)
```

Constructs a new `SdmIOException` with the specified detail message and cause.

Parameters:

`message` - the detail message

`cause` - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmIOException

```
public SdmIOException(String message)
```

Constructs a new `SdmIOException` with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

`message` - the detail message

SdmIOException

```
public SdmIOException(Throwable cause)
```

Constructs a new `SdmIOException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Interface SdmKey

at.iaik.secricom.sdm

All Known Subinterfaces:

ConfigurationProtectedKey, TokenProtectedKey

---

public interface **SdmKey**

SdmKey is the base interface for all types of SDM keys and provides methods to obtain the key identifier and the actual key material.Aside from its protection mechanism (see below) an SDM key consists of a unique identifier and the actual key material. The unique identifier should allow associating the protected key material with a certificate. The key material is the actual key information, which could range from credentials to access a legacy system to cryptographic keys.

There exist three protection mechanisms for SDM keys. The first type called platform configuration protected or PCP in short. As the name suggest, for this protection mechanism the key is bound to a specific set of platform states. Only, if the host platform is in a valid configuration, the key is released. The second type is named authorization token protected or ATP. Thus protected keys are released if a given authorization token (password) matches the authorization token associated with the key. The last type authorized platform configuration protected (APCP) releases keys to the host, if the host is in a valid configuration and can supply an authorization token.

See Also:

SdmIdentifier, SdmKeyMaterial

Nested Class Summary		Page
static enum	**SdmKey.Type** Protection mechanism Type of a key.	64

Method Summary		Page
SdmIdentifier	**getKeyIdentifier**()	63
SdmKeyMaterial	**getSdmKeyMaterial**()	63
SdmKey.Type	**getType**()	63

Method Detail

getKeyIdentifier

SdmIdentifier **getKeyIdentifier**()

Returns:

the unique identifier of the key

---

getSdmKeyMaterial

SdmKeyMaterial **getSdmKeyMaterial**()

Returns:

the actual key material protected by the SDM

---

getType

SdmKey.Type **getType**()

Returns:

the type of the key. Either *Platform Configuration Protected (PCP)* , *Authorization Token Protected (ATP)*, or *Authorized Platform Configuration Protected (APCP)*.

Enum SdmKey.Type

at.iaik.secricom.sdm

java.lang.Object

└ java.lang.Enum<SdmKey.Type>

└ **at.iaik.secricom.sdm.SdmKey.Type**

All Implemented Interfaces:

Comparable<SdmKey.Type>, Serializable

Enclosing class:

SdmKey

---

public static enum **SdmKey.Type**

extends Enum<SdmKey.Type>

Protection mechanism `Type` of a key. Either *Platform Configuration Protected (PCP)* , *Authorization Token Protected (ATP)*, or *Authorized Platform Configuration Protected (APCP)*.

Enum Constant Summary	Page
APCP	65
ATP	64
PCP	64

Method Summary		Page
static SdmKey.Type	**valueOf**(String name)	65
static SdmKey.Type[]	**values**()	65

Enum Constant Detail

PCP

public static final SdmKey.Type **PCP**

ATP

public static final SdmKey.Type **ATP**

APCP

public static final SdmKey.Type **APCP**

Method Detail

values

public static SdmKey.Type[] **values**()

---

valueOf

public static SdmKey.Type **valueOf**(String name)

Class SdmKeyMaterial

at.iaik.secricom.sdm

java.lang.Object

└ **at.iaik.secricom.sdm.SdmKeyMaterial**

---

public class **SdmKeyMaterial**

extends Object

SdmKeyMaterial is information protected by the SDM. The idea behind the SDM is to release key information only to platforms in a trusted state. SdmKeyMaterial represents this SDM protected information.

Currently, SdmKeyMaterial is a byte[] with a maximum length of 1024 bytes. This value is not finalized and must be adapted to the capabilities of the actual SDM hardware implementation. A lower bound for protected key should be at least 256 bytes, which is the maximum length of a 2048 bit private RSA key.

Constructor Summary	Page
SdmKeyMaterial(byte[] material) Constructs a new KeyMaterial with the specified byte[] containing the key material.	66

Method Summary		Page
boolean	**equals**(Object obj)	66
byte[]	**getKeyMaterial**()	66
int	**hashCode**()	66
String	**toString**()	66

Constructor Detail

SdmKeyMaterial

```
public SdmKeyMaterial(byte[] material)
              throws SdmInvalidParameterException
```

Constructs a new KeyMaterial with the specified byte[] containing the key material.

Parameters:

material - the key material, must not be null and must not be longer than 1024 bytes

Throws:

SdmInvalidParameterException - if the specified key material is null or longer than 1024 bytes

Method Detail

getKeyMaterial

```
public byte[] getKeyMaterial()
```

Returns:

the SDM protected key material. Key material is simply a byte[] with a maximum length of 1024 bytes.

hashCode

```
public int hashCode()
```

Overrides:

hashCode in class Object

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class SdmObjectNotEmptyException

at.iaik.secricom.sdm

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └at.iaik.secricom.sdm.SdmException
              └at.iaik.secricom.sdm.SdmObjectNotEmptyException
```

All Implemented Interfaces:

Serializable

```
public class SdmObjectNotEmptyException
extends SdmException
```

`SdmObjectNotEmptyException` indicates that removal of an object from the SDM failed because the specific object is not empty. This exception might be thrown during and SDM administration session if a host or key removal fails due to the above cause.

Constructor Summary	Page
SdmObjectNotEmptyException() Constructs a new `SdmObjectNotEmptyException` with null as its detail message.	68
SdmObjectNotEmptyException(String message) Constructs a new `SdmObjectNotEmptyException` with the specified detail message.	68

SdmObjectNotEmptyException(String message, Throwable cause) Constructs a new SdmObjectNotEmptyException with the specified detail message and cause.	68
SdmObjectNotEmptyException(Throwable cause) Constructs a new SdmObjectNotEmptyException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	68

Constructor Detail

SdmObjectNotEmptyException

```
public SdmObjectNotEmptyException()
```

Constructs a new SdmObjectNotEmptyException with null as its detail message. The cause cause is not initialized. A call to the initCause(Throwable) method may be used to initialize the cause after exception creation.

SdmObjectNotEmptyException

```
public SdmObjectNotEmptyException(String message,
                                  Throwable cause)
```

Constructs a new SdmObjectNotEmptyException with the specified detail message and cause.

Parameters:

message - the detail message

cause - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmObjectNotEmptyException

```
public SdmObjectNotEmptyException(String message)
```

Constructs a new SdmObjectNotEmptyException with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

message - the detail message

SdmObjectNotEmptyException

```
public SdmObjectNotEmptyException(Throwable cause)
```

Constructs a new `SdmObjectNotEmptyException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Class SdmRandomBytes

at.iaik.secricom.sdm

```
java.lang.Object
  └ at.iaik.secricom.sdm.SdmRandomBytes
```

```
public class SdmRandomBytes
extends Object
```

`SdmRandomBytes` represents an array of random bytes generated by the SDM's hardware random number generator. The number of bytes generated in one go is restricted to a maximum of **4096 bits** (512 bytes).

The SDM is equipped with a true random number generator which is internally used by the SDM for the creation of nonces. Mobile devices which use the SDM and are not equipped with a TPM might profit from using this source of randomness provided by the SDM. Therefore, the `SdmSession.getRandomBytes(int)` command that grants access to this facility was added to the client interface of the SDM.

See Also:

SdmSession.getRandomBytes(int)

Constructor Summary	Page
SdmRandomBytes(byte[] randomBytes) Constructs a new `SdmRandomBytes` with the specified byte[].	69

Method Summary		Page
boolean	**equals**(Object obj)	70
byte[]	**getRandomBytes**()	70
int	**hashCode**()	70
String	**toString**()	70

Constructor Detail

SdmRandomBytes

```
public SdmRandomBytes(byte[] randomBytes)
              throws SdmInvalidParameterException
```

Constructs a new `SdmRandomBytes` with the specified byte[].

Parameters:

`randomBytes` - the random bytes generated by the true random number generator of the SDM. The byte array must be between 1 and 512 bytes long.

Throws:

`SdmInvalidParameterException` - if the specified byte[] is null, empty, or longer than 512 bytes.

Method Detail

getRandomBytes

```
public byte[] getRandomBytes()
```

Returns:

the random bytes generated by the true random number generator in the SDM. The returned array is between 1 and 512 bytes long

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

---

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

---

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Interface SdmSession

at.iaik.secricom.sdm

`public interface` **SdmSession**

An `SdmSession` (pronounced S-D-M Session) enables a client to communicate with the SDM. A client in this context is a software entity that wants to use SDM functionality. The `SdmSession` specifically provides the SDM user interface in contrast to the SDM administration interface (cf. `SdmAdministrationSession`).

The `SdmSession` grants access to the following functionalities:

- Generation of random nonce values, required for platform configuration verification: `createTpmNonce()`
- Verification of the platform configuration: `verifyConfiguration(SdmIdentifier, TpmQuote)`
- Retrieval of an authorization token protected key: `getKey(SdmIdentifier, SdmToken)`
- Retrieval of a key that is only accessible in a specific platform configuration: `getKey(SdmIdentifier, TpmQuote)`
- Access to a key that requires both a specific platform configuration and an authorization token to be released to the host: `getKey(SdmIdentifier, TpmQuote, SdmToken)`
- A hardware based random number generator: `getRandomBytes(int)`

For platform state verification and key retrieval it is actually necessary to perform several steps. The first step is to generate a nonce using the `createTpmNonce()` method. This nonce must be used to generate the platform configuration report (TPM quote) required by the `verifyConfiguration(SdmIdentifier, TpmQuote)`, `getKey(SdmIdentifier, TpmQuote)`, and `getKey(SdmIdentifier, TpmQuote, SdmToken)` methods.

The SDM manages three types of keys. These types are distinguished by their protection mechanism. The lowest level of protection is to use only an authorization token (cf. `SdmToken`). The second protection level is to guarantee that the SDM host platform is in a specific platform state using the platforms TPM. The highest level of security is achieved by protecting a key with both an authorization token and the requirement of a specific state for key release.

It is very probable that an SDM will only be able to handle one session at the time. There should be a central authority on the SDM host that manages access to the SDM. The `close()` method closes the current SDM session and releases all SDM resources that depend on it. Subsequent calls to a closed session object will cause an SdmException to be thrown.

See Also:

`SecureDockingModule`, `SdmAdministrationSession`

Method Summary		Page
void	**close**() Close tells this SDM session to shut down and release all related resources.	77
TpmNonce	**createTpmNonce**() In order to guarantee the freshness of a TPM platform configuration report, a so called TPM quote, external data must be provided to the TPM.	72
SdmKey	**getKey**(SdmIdentifier keyId, SdmToken token) Retrieves an authorization token protected key from the SDM.	74
SdmKey	**getKey**(SdmIdentifier keyId, TpmQuote quote) Retrieves a specified key from the SDM by providing a TPM signed platform configuration report to the SDM.	75
SdmKey	**getKey**(SdmIdentifier keyId, TpmQuote quote, SdmToken token) Retrieves a specified key from the SDM by providing a TPM signed platform configuration report and an authorization token to the SDM.	76
SdmRandomBytes	**getRandomBytes**(int length) Generates random bytes using the true random number generator included in the SDM.	76
boolean	**verifiyConfigurationAndAuthorization**(SdmIdentifier keyId, TpmQuote quote, SdmToken token) Tests if the specified configuration and authorization token grants access to the specified key.	74
boolean	**verifyConfiguration**(SdmIdentifier keyId, TpmQuote quote) Tests if the specified configuration grants access to the specified key.	73

Method Detail

createTpmNonce

TpmNonce **createTpmNonce**()
throws SdmException

In order to guarantee the freshness of a TPM platform configuration report, a so called TPM quote, external data must be provided to the TPM. This external data is signed by the TPM together with the current platform state. Such external data should be a random number. Uniqueness is a requirement of the random number. Such a random number is called nonce. The required length of 20 bytes ensures with a very high probability (enough for practical purposes) that it will be unique, if a reliable random number generator is used. The SDM provides a hardware based random number generator.

To use an SDM capability that requires an TPM quote it is first necessary to request a nonce using this method. This nonce must be used in the next TPM quote supplied to the SDM. The current nonce is not saved between sessions.

Returns:

a nonce suitable for use with the TPM quote operation.

Throws:

SdmException - if an SDM related error occurs.

---

verifyConfiguration

```
boolean verifyConfiguration(SdmIdentifier keyId,
                            TpmQuote quote)
                     throws SdmException
```

Tests if the specified configuration grants access to the specified key. Sometimes it is useful for a client to enable the client to verify if a specific platform configuration is sufficient to release a certain key, without actually requesting release of this key. This method serves this purpose.

Performing a platform verification is a three step process. First the createTpmNonce() method must be called to create a nonce that must be used in the second step of the protocol: the creation of the TPM quote. Calling the verifyConfiguration method is actually the last step in the process.

A call to verifyConfiguration verifies the configuration of the host system by analyzing the given TPM quote. The verification is based on comparing the hash of the relevant PCRs contained in the TPM quote with a set of valid platform configurations. Before this comparison takes place, the SDM first verifies the signature over the TPM quote and if the last nonce requested with the createTpmNonce() method was used in the creation of the TPM quote.

A last note: It is possible to verify the state only for keys that are only released in a specific state, not for keys that require an authorization token. To be able to verify a key that requires both a specific state and an authorization token the verifiyConfigurationAndAuthorization(SdmIdentifier, TpmQuote, SdmToken) is provided.

Parameters:

keyId - the unique identifier of the key that should be released.

quote - the TPM signed report of the platform configuration.

Returns:

true if the given state would allow access to a platform state protected key, false otherwise.

Throws:

SdmException - if an SDM related error occurs.

See Also:

createTpmNonce()

---

verifiyConfigurationAndAuthorization

```
boolean verifiyConfigurationAndAuthorization(SdmIdentifier keyId,
                                             TpmQuote quote,
                                             SdmToken token)
                                      throws SdmException
```

Tests if the specified configuration and authorization token grants access to the specified key. Sometimes it is useful for a client to enable the client to verify if a specific platform configuration and authorization token is sufficient to release a certain key, without actually requesting release of this key. This method serves this purpose.

Performing a platform verification is a three step process. First the `createTpmNonce()` method must be called to create a nonce that must be used in the second step of the protocol: the creation of the TPM quote. Calling the `verifyConfiguration` method is actually the last step in the process.

A call to `verifyConfiguration` verifies the configuration of the host system by analyzing the given TPM quote. The verification is based on comparing the hash of the relevant PCRs contained in the TPM quote with a set of valid platform configurations. Before this comparison takes place, the SDM first verifies the signature over the TPM quote and if the last nonce requested with the `createTpmNonce()` method was used in the creation of the TPM quote.

A last note: With this method it is possible to verify the state for keys that are protected by a specific state and an authorization token. To verify a key that requires only a specific configuration the `verifiyConfigurationAndAuthorization(SdmIdentifier, TpmQuote, SdmToken)` is provided.

Parameters:

`keyId` - the unique identifier of the key that should be released.

`quote` - the TPM signed report of the platform configuration.

`token` - the authorization token that would be required to release the key.

Returns:

true if the given configuration and authorization token would allow access to a platform state protected key, false otherwise.

Throws:

`SdmException`

---

getKey

```
SdmKey getKey(SdmIdentifier keyId,
              SdmToken token)
       throws SdmException
```

Retrieves an authorization token protected key from the SDM. Key release is simple compared to the release mechanism based on platform configuration. No beforehand call to `createTpmNonce()` is necessary.

Parameters:

keyId - the unique identifier of the requested key.

token - the authorization token that grants access to the key.

Returns:

the key if authorization was successful.

Throws:

SdmException - if an SDM related problem occurs, or the authorization token is insufficient.

---

getKey

```
SdmKey getKey(SdmIdentifier keyId,
              TpmQuote quote)
       throws SdmException
```

Retrieves a specified key from the SDM by providing a TPM signed platform configuration report to the SDM.

Performing a platform verification based key release is a three step process. First the createTpmNonce() method must be called to create a nonce that must be used in the second step of the protocol: the creation of the TPM quote. Calling the getKey method is actually the last step in the process.

A call to getKey verifies the configuration of the host system by analyzing the given TPM quote. The verification is based on comparing the hash of the relevant PCRs contained in the TPM quote with a set of valid platform configurations. Before this comparison takes place, the SDM first verifies the signature over the TPM quote and if the last nonce requested with the createTpmNonce() method was used in the creation of the TPM quote.

A last note: With this method it is not possible to retrieve keys that are protected by a specific configuration and an authorization token. For this the verifiyConfigurationAndAuthorization(SdmIdentifier, TpmQuote, SdmToken) method is provided.

Parameters:

keyId - the unique identifier of the requested key.

quote - the TPM signed report of the platform configuration.

Returns:

the key if the platform configuration is valid.

Throws:

SdmException - if an SDM related problem occurs, or the platform configuration is invalid.

---

getKey

```
SdmKey getKey(SdmIdentifier keyId,
              TpmQuote quote,
              SdmToken token)
       throws SdmException
```

Retrieves a specified key from the SDM by providing a TPM signed platform configuration report and an authorization token to the SDM.

Performing a platform verification based key release is a three step process. First the `createTpmNonce()` method must be called to create a nonce. This nonce must be used in the second step of the protocol: the creation of the TPM quote. Calling this `getKey` method is actually the last step in the process.

A call to this `getKey` verifies the configuration of the host system by analyzing the given TPM quote. The verification is based on comparing the hash of the relevant PCRs contained in the TPM quote with a set of valid platform configurations. Before this comparison takes place, the SDM first verifies the signature over the TPM quote and if the last nonce requested with the `createTpmNonce()` method was used in the creation of the TPM quote.

A last note: With this method it is not possible to retrieve keys that are protected solely by a specific configuration. For this the `verifiyConfigurationAndAuthorization(SdmIdentifier, TpmQuote, SdmToken)` method is provided.

Parameters:

`keyId` - the unique identifier of the requested key.

`quote` - the TPM signed report of the platform configuration.

Returns:

the key if the platform configuration is valid.

Throws:

`SdmException` - if an SDM related problem occurs, or the platform configuration is invalid.

---

getRandomBytes

```
SdmRandomBytes getRandomBytes(int length)
                       throws SdmException
```

Generates random bytes using the true random number generator included in the SDM.

Parameters:

`length` - the number of random bytes; it must not exceed **4096 bits** (512 bytes).

Returns:

an `SdmRandomBytes` object containing the random bytes generated by the SDM's true random number generator

Throws:

SdmException - if an SDM related problem occurs

See Also:

SdmRandomBytes

---

close

```
void close()
    throws SdmException
```

Close tells this SDM session to shut down and release all related resources.

Throws:

SdmException - if an SDM related problem occurs.

Interface SdmStatus

at.iaik.secricom.sdm

```
public interface SdmStatus
```

SdmStatus indicates the operation status of the SDM. An SDM only accepts one active session at the time. To find out if the SDM is active a status request must be send to the SDM. A status request is always accepted and answered by the SDM, even if it is in an active session.

Nested Class Summary		Page
static enum	**SdmStatus.Status** Status represents the current status of the SDM.	78

Method Summary		Page
SdmStatus.Status	**getStatus**()	77

Method Detail

getStatus

```
SdmStatus.Status getStatus()
```

Returns:

the Status of the SDM

Enum SdmStatus.Status

at.iaik.secricom.sdm

java.lang.Object

└ java.lang.Enum<SdmStatus.Status>

└ **at.iaik.secricom.sdm.SdmStatus.Status**

All Implemented Interfaces:

Comparable<SdmStatus.Status>, Serializable

Enclosing class:

SdmStatus

---

public static enum **SdmStatus.Status**

extends Enum<SdmStatus.Status>

Status represents the current status of the SDM. From a clients perspective the SDM is either UNINITIALIZED, IDLE, or IN_SESSION..

Enum Constant Summary	Page
IDLE	79
IN_SESSION	79
UNINITIALIZED	78

Method Summary		Page
static SdmStatus.Status	**valueOf**(String name)	79
static SdmStatus.Status[]	**values**()	79

Enum Constant Detail

UNINITIALIZED

public static final SdmStatus.Status **UNINITIALIZED**

IDLE

public static final SdmStatus.Status **IDLE**

IN_SESSION

```
public static final SdmStatus.Status IN_SESSION
```

Method Detail

values

```
public static SdmStatus.Status[] values()
```

---

valueOf

```
public static SdmStatus.Status valueOf(String name)
```

Class SdmToken

at.iaik.secricom.sdm

```
java.lang.Object
  └ at.iaik.secricom.sdm.SdmToken
```

```
public class SdmToken
extends Object
```

SdmToken are authorization tokens that grant access to certain keys. The SDM provides three different mechanisms to protect a key. Two of them (ATP and APCP) (see SdmKey) require an authorization token. A SDM authorization token is a byte array with a maximum length of 20 bytes.

Constructor Summary	Page
SdmToken(byte[] authToken) Constructs a new SdmToken with the specified byte[] encoded authorization information.	80

Method Summary		Page
boolean	**equals**(Object obj)	80
byte[]	**getAuthorizationToken**()	80
int	**hashCode**()	80
String	**toString**()	80

Constructor Detail

SdmToken

```
public SdmToken(byte[] authToken)
         throws SdmInvalidParameterException
```

Constructs a new `SdmToken` with the specified byte[] encoded authorization information. The byte[] must not be null and must not be longer than 20 bytes.

Parameters:

`authToken` - the authorization token encoded as a byte[] of a maximum length of 20 bytes.

Throws:

`SdmInvalidParameterException` - if the specified token is null, or longer than 20 bytes.

Method Detail

getAuthorizationToken

```
public byte[] getAuthorizationToken()
```

Returns:

the authorization token, which is a byte[] with a maximum length of 20 bytes.

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

---

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

---

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class SdmUnknownKeyException

at.iaik.secricom.sdm

java.lang.Object

└ java.lang.Throwable

└ java.lang.Exception

└ at.iaik.secricom.sdm.SdmException

└ **at.iaik.secricom.sdm.SdmUnknownKeyException**

All Implemented Interfaces:

Serializable

---

public class **SdmUnknownKeyException**

extends SdmException

SdmUnknownKeyException indicates that an SDM operation failed because it tried to operate on a key that is not known to the SDM in the specified context.

Constructor Summary	Page
SdmUnknownKeyException() Constructs a new SdmUnknownKeyException with null as its detail message.	81
SdmUnknownKeyException(String message) Constructs a new SdmUnknownKeyException with the specified detail message.	82
SdmUnknownKeyException(String message, Throwable cause) Constructs a new SdmUnknownKeyException with the specified detail message and cause.	82
SdmUnknownKeyException(Throwable cause) Constructs a new SdmUnknownKeyException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	82

Constructor Detail

SdmUnknownKeyException

public **SdmUnknownKeyException**()

Constructs a new SdmUnknownKeyException with null as its detail message. The cause cause is not initialized. A call to the initCause(Throwable) method may be used to initialize the cause after exception creation.

SdmUnknownKeyException

```
public SdmUnknownKeyException(String message,
                              Throwable cause)
```

Constructs a new `SdmUnknownKeyException` with the specified detail message and cause.

Parameters:

`message` - the detail message

`cause` - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmUnknownKeyException

```
public SdmUnknownKeyException(String message)
```

Constructs a new `SdmUnknownKeyException` with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

`message` - the detail message

SdmUnknownKeyException

```
public SdmUnknownKeyException(Throwable cause)
```

Constructs a new `SdmUnknownKeyException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Interface SdmVersion

at.iaik.secricom.sdm

```
public interface SdmVersion
```

`SdmVersion` defines the version of the SDM. The versions system of the SDM distinguishes between major version and minor version. A major version change signals a change in the binary command interface of the *Secure Docking Module* hardware module, whereas a minor versions change indicates a revision of the SDM implementation.

Method Summary		Page
int	**getMajor**() The major version of the SDM hardware module.	83
int	**getMinor**() The minor version of the SDM hardware module.	83

Method Detail

getMajor

```
int getMajor()
```

The major version of the SDM hardware module. Internally the SDM uses a byte to represent this number, so the value of the version must lie between 0 and 255.

Returns:

the major version of the SDM

---

getMinor

```
int getMinor()
```

The minor version of the SDM hardware module. Internally the SDM uses a byte to represent this number, so the value of the version must lie between 0 and 255.

Returns:

the minor version of the SDM

Interface SecureDockingModule

at.iaik.secricom.sdm

```
public interface SecureDockingModule
```

`SecureDockingModule` manages access to the functionalities provided by a *Secure Docking Module*. It provides access to both the client and administrative capabilities of the SDM and manages SDM session establishment.

The Secure Docking Module is a hardware local-attestation-token. It protects secrets, and these secrets are only revealed if the host platform is in a trusted state. For a description of the functionality that is available as part of the SDM client interface see `SdmSession`.

Apart from this basic functionality it must also provide support for maintenance. Maintenance includes updating the key material and updating the valid platform configurations, as well as adding new hosts,

new key material, and new configurations. For a detailed description of the administrative interface see SdmAdministrationSession.

In order to communicate with the SDM an end-to-end secure session must be established. This requires mutual authentication between the SDM and the host platform. The `createSession` methods provided by this interface hide the complexities behind the session establishment protocols.

The `SecureDockingModule` provides the entry points into both the client and the administrative interface of the SDM. In order to use the SDM client functionality it is necessary to create an `SdmSession`. This is done with the createSession(SdmIdentifier, RSAPrivateKey) command. To administer the SDM a `SdmAdministrationSession` must be created. For this the createAdministrationSession(SdmIdentifier, RSAPrivateKey) method is provided.

It is important to note that the SDM only allows one active session. It is not possible to initiate any kind of new session with the SDM if the SDM is already in any kind of session.

See Also:

SdmSession, SdmAdministrationSession

Method Summary		Page
SdmAdministrationSession	**createAdministrationSession**(SdmIdentifier adminHostid, RSAPrivateKey hak) Prepares an administration session with the SDM which provides access to the SDM administration interface.	87
SdmSession	**createSession**(SdmIdentifier hostId, RSAPrivateKey hak) Prepares a client session with the SDM and thus grant access to the client specific SDM functionality.	86
SdmIdentifier	**getIdentifier**() Returns the unique identifier of the SDM.	85
PublicKey	**getSdmAuthenticationKey**() Returns the public part of the unique *SDM Authentication Key* pair of the SDM.	86
SdmStatus	**getStatus**() Returns the current status of the SDM.	85
SdmVersion	**getVersion**() Returns the version of the SDM.	85

Method Detail

getVersion

```
SdmVersion getVersion()
                throws SdmException
```

Returns the version of the SDM. The version of the SDM is split into a major ordinal and a minor ordinal. Both numbers are bytes, which theoretically allows for 65536 different version. A change in the major number indicates that the binary command interface of the SDM was changed, whereas a minor number change symbolizes an internal revision of the SDM.

Returns:

the `SdmVersion`

Throws:

`SdmException` - if an SDM related exception occurs

---

getStatus

```
SdmStatus getStatus()
          throws SdmException
```

Returns the current status of the SDM. From a clients perspective the SDM can only be in one of three states. It could be `UNINITIALIZED`, `IDLE`, or `IN_SESSION`. The SDM only accepts a session request if it is `IDLE`. The SDM only allows one session at any given time.

Returns:

the `SdmStatus`

Throws:

`SdmException` - if an SDM related exception occurs

---

getIdentifier

```
SdmIdentifier getIdentifier()
              throws SdmException
```

Returns the unique identifier of the SDM. In order to be able to establish a session the session initiation request must be encrypted using the public part of the *SDM Authentication Key* pair. Each SDM has its own unique authentication key and thus its own identity. In order to know with which SDM the client is communicating, this function can be called.

Returns:

the unique identifier of the SDM

Throws:

`SdmException`

---

getSdmAuthenticationKey

```
PublicKey getSdmAuthenticationKey()
                          throws SdmException
```

Returns the public part of the unique *SDM Authentication Key* pair of the SDM. Each SDM has its own *SDM Authentication Key* and thus its own identity. The public *SDM Authentication Key* is required during session establishment.

Returns:

the public part of the *SDM Authentication Key*, which is a 2048-bit RSA key pair

Throws:

SdmException

---

createSession

```
SdmSession createSession(SdmIdentifier hostId,
                         RSAPrivateKey hak)
                  throws SdmException
```

Prepares a client session with the SDM and thus grant access to the client specific SDM functionality. For details on the client specific capabilities of the SDM refer to SdmSession.

A call to the `createSession` method instructs the SDM interface library to lock access to the SDM and prepare a connection. To actually establish the SDM session a call to the connect method of the `SdmSession` is required. An SDM session requires mutual authentication between the SDM and the host. In order to enable host authentication a *Host Authentication Key (HAK)* must be specified. This HAK must be previously inserted into the SDM. The key pre-sharing is a responsibility of the SDM issuing organization.

A SDM can support multiple hosts, it is therefore necessary to instruct the SDM which host wants to establish a connection to it. Host identification and the HAK must match, otherwise session establishment fails.

Parameters:

`hostId` - identifies the host that wants to establish an SDM session

`hak` - authenticates the host that wants to establish an SDM session HAK and `hostId` must match; otherwise session establishment will fail.

Returns:

an `SdmSession` object that grants access to the SDM client specific capabilities.

Throws:

SdmException - if session establishment with the SDM fails.

See Also:

SdmSession

createAdministrationSession

```
SdmAdministrationSession createAdministrationSession(SdmIdentifier adminHostid,
                                                     RSAPrivateKey hak)
                                              throws SdmException
```

Prepares an administration session with the SDM which provides access to the SDM administration interface. For details of the administrative interface of the SDM see SdmAdministrationSession.

This method prepares a session with the SDM and locks the SDM. The establishment of an administration session a complex process. To actually connect to the TPM is is necessary to perform several steps. Some of the required operations, like the attestation of the platform, are out of scope of this API. Therefore, it is not possible to completely hide the complexity of the session establishment process. These necessary operations are covered by the SdmAdministrationSession class in more detail.

Parameters:

`adminHostid` - identifies the host that wants to connect to the SDM to perform SDM administration

`hak` - authenticates the host that wants to administer the SDM

Returns:

an `SdmAdministrationSession` object that provides access to the administration interface of the SDM.

Throws:

`SdmException` - if an SDM error occurs.

See Also:

SdmAdministrationSession

Class TpmNonce

at.iaik.secricom.sdm

```
java.lang.Object
  └ at.iaik.secricom.sdm.TpmNonce
```

```
public class TpmNonce
extends Object
```

`TpmNonce` represents a 20 bytes long nonce used in a TPM quote operation. To guarantee that a TPM quote is fresh the verifier, in this case the SDM, generates a random number 20 bytes long. This number is included in the signed platform configuration report created by the TPM quote operation. This number must only be used once, hence the name nonce.

Constructor Summary	Page
TpmNonce(byte[] nonce) Constructs new TpmNonce from the specified byte[].	88

Method Summary		Page
boolean	**equals**(Object obj)	89
byte[]	**getNonce**()	88
int	**hashCode**()	89
String	**toString**()	89

Constructor Detail

TpmNonce

```
public TpmNonce(byte[] nonce)
         throws SdmInvalidParameterException
```

Constructs new TpmNonce from the specified byte[]. This byte[] array must be exactly 20 bytes long and these bytes should be randomly generated. This constructor creates a defensive copy of the given byte array to prevent modifications.

Parameters:

nonce - the byte data that should serve as a nonce in a TPM quote operation

Throws:

SdmInvalidParameterException - if the specified byte[] is either null or not exactly 20 bytes long.

Method Detail

getNonce

```
public byte[] getNonce()
```

Returns:

the randomly generated byte[] data to serve as a nonce in a TPM quote operation.

hashCode

```
public int hashCode()
```

Overrides:

hashCode in class Object

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class TpmQuote

at.iaik.secricom.sdm

```
java.lang.Object
  └ at.iaik.secricom.sdm.TpmQuote
```

```
public class TpmQuote
extends Object
```

`TpmQuote` represents a signed TPM quote info data structure. It consists of a byte[] representing the quote info data structure itself and a byte[] containing the RSASSA-PKCS-v1.5 using SHA-1 signature over the TPM quote info structure.

Constructor Summary	Page
`TpmQuote(byte[] quoteInfo, byte[] signature)` Constructs a new `TpmQuote` from a specified TPM quote info and signature.	90

Method Summary		Page
boolean	`equals(Object obj)`	91
byte[]	`getQuoteInfo()`	90
byte[]	`getSignature()`	91
int	`hashCode()`	91
String	`toString()`	91

Constructor Detail

TpmQuote

```
public TpmQuote(byte[] quoteInfo,
                byte[] signature)
         throws SdmInvalidParameterException
```

Constructs a new `TpmQuote` from a specified TPM quote info and signature. This constructor creates defensive copies of the specified parameters. A TPM quote info must be exactly 48 bytes long.

Parameters:

`quoteInfo` - the quote info that is signed with the specified signature

`signature` - the signature over the specified TPM quote info

Throws:

`SdmInvalidParameterException` - if any of the specified parameters is null, or the TPM quote info structure is not exactly 48 bytes long, or the signature is not exactly 128 bytes long, or if the TPM quote info structure does not start with "0x01,0x01,0x00,0x00,0x81,0x85,0x79,0x84" (1100QUOT).

Method Detail

getQuoteInfo

```
public byte[] getQuoteInfo()
```

Returns:

a defensive copy of the TPM quote info data structure

---

getSignature

```
public byte[] getSignature()
```

Returns:

a defensive copy of the signature over the quote info TPM data structure

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

---

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

---

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Class Util

at.iaik.secricom.sdm

```
java.lang.Object
  └ at.iaik.secricom.sdm.Util
```

```
public class Util
extends Object
```

`Util` is a set of utility methods that support the SDM interface with functions like byte[] pretty printing, a factory method for TPM quote info data structures etc..

Constructor Summary	Page
`Util()`	92

Method Summary		Page
static String	**formatByteArray**(byte[] data, int bytesPerRow) Pretty prints a given byte array into a formated output string with a specified number of bytes per row.	92
static TpmQuote	**generateTpmQuote**(Signature sign, RSAPrivateKey aik, byte[] quoteInfo) Signs the specified TPM quote info data structure given as a byte array with the specified private RSA key and the given signature object.	93
static byte[]	**generateTpmQuoteInfo**(byte[] conf, byte[] nonce) Generates a TPM quote info data structure conforming byte array.	92

static void	**testNull**(Object obj, String name) Tests if the specified object is null.	94

Constructor Detail

Util

```
public Util()
```

Method Detail

formatByteArray

```
public static String formatByteArray(byte[] data,
                                     int bytesPerRow)
```

Pretty prints a given byte array into a formated output string with a specified number of bytes per row. Each byte is printed in hexadecimal format.

Parameters:

`data` - the byte[] to format

`bytesPerRow` - the number of bytes one line of text should contain

Returns:

a formatted string representing the given byte[]

---

generateTpmQuoteInfo

```
public static byte[] generateTpmQuoteInfo(byte[] conf,
                                          byte[] nonce)
```

Generates a TPM quote info data structure conforming byte array. A TPM quote info is the data structure that is signed by the TPM, when performing a TPM quote operation.The exact nature of the quote info structure is outlined in the TPM specification. Basically it consists of a version identifier, a data structure identifier, the SHA-1 hash of the platform configuration and a so called external value that should be a nonce to guarantee the freshness of the quote. A TPM quote info is exactly 48 bytes long.

Parameters:

`conf` - the PCR composite hash. This is 20 byte SHA-1 hash of the PCRs selected for the platform quote.

`nonce` - an externally supplied nonce that guarantees that the TPM quote is fresh.

Returns:

the quote info structure.

generateTpmQuote

```
public static TpmQuote generateTpmQuote(Signature sign,
                                        RSAPrivateKey aik,
                                        byte[] quoteInfo)
                                 throws InvalidKeyException,
                                        SignatureException,
                                        SdmInvalidParameterException
```

Signs the specified TPM quote info data structure given as a byte array with the specified private RSA key and the given signature object. This method creates a fake TpmQuote object which can be used for testing the SDM functionality. The TPM uses RSA-SSA-v1.5 with SHA-1 for quote signature creation.

Parameters:

`sign` - the signature object to use to sign the quote info data structure.

`aik` - the attestation identity key that is used for the signature

`quoteInfo` - the quote info data structure to sign

Returns:

a TpmQuote that contains both the quote info data structure as well as the signature.

Throws:

`InvalidKeyException` - if the provided key is not compatible with the specified signature object.

`SignatureException` - if signing the data fails.

`SdmInvalidParameterException` - If the given TPM quote info data structure is not exactly 48 bytes long.

testNull

```
public static void testNull(Object obj,
                            String name)
                     throws SdmInvalidParameterException
```

Tests if the specified object is null. If this is the case, an `SdmInvalidParameterException` is thrown with the following message format as exception message: `"The specified {0} is null."` . The `{0}` parameter is substituted with the specified name.

Parameters:

`obj` - the object to test if it is null

`name` - the human identifiable name of the object

Throws:

SdmInvalidParameterException - if the specified object is null

Package at.iaik.secricom.sdm.admin

Interface Summary		Page
ConfigurationProtectedKey	ConfigurationProtectedKey is an SdmKey implementing the platform configuration dependent key release protection mechanism.	95
SdmAdministrationSession	SdmAdministrationSession grants access to the administration interface of the *Secure Docking Module (SDM)*.	100
SdmHost	SdmHost manages all the information the SDM needs to communicate with a host and verify the signature on the host's TPM's platform configuration reports.	105
TokenProtectedKey	TokenProtectedKey is a key, the release of which is at least protected by an authorization token.	111

Class Summary		Page
HostConfiguration	HostConfiguration represents a valid platform configuration.	98

Exception Summary		Page
SdmUnknownHostException	SdmUnknownHostException indicates that an SDM operation failed, because it tried to operate on an unknown SDM host.	108

Interface ConfigurationProtectedKey

at.iaik.secricom.sdm.admin

All Superinterfaces:

SdmKey

```
public interface ConfigurationProtectedKey
extends SdmKey
```

ConfigurationProtectedKey is an SdmKey implementing the platform configuration dependent key release protection mechanism. ConfigurationProtectedKey is part of the administration interface of the SDM and provides methods to manage a set of valid platform configurations (HostConfigurations).

The SDM provides three different key protection mechanisms:

- *Platform Configuration Protected* keys are only released to the client if the client can proof that it is an well defined, and previously set up state (platform configuration). A TPM provides the

capabilities to measure the clients platform state and provide a signature on the clients platform configuration report, the so-called quote. The platform configuration is represented in the TPM by *Platform Configuration Registers PCRs*. A quote provides a signature on a SHA-1 hash of several selected PCRs. This SHA-1 hash is represented by a `HostConfiguration` and verified by the SDM. A `ConfigurationProtectedKey` provides facilities to add, remove and test the existence of such `HostConfigurations`.

- *Authorization Token Protected (ATP)* keys are protected by an authorization token (`SdmToken`), that is a password. The key is only released if the client can supply an authorization token that matches the authorization token in the SDM. This method is useful if the host platform does not provide the capabilities to create a signed platform configuration report, but a user is present and can provide a password.
- *Authorized Platform Configuration Protected* keys combine the two former methods. A key is only released if the platform is in a certain state and if the client can provide an authorization token.

Each key is associated with a specific `SdmHost` object. In order to look up a key in the SDM it is necessary to know both, the unique identifier of the host and the unique identifier of the key. The library implementation must hide this complexity from the client, as far as possible. An exception to this rule is that an implementation is allowed to throw exceptions that signal error conditions pertaining to missing SDM objects.

See Also:

`HostConfiguration`, `SdmKey`, `SdmToken`

Nested classes/interfaces inherited from interface at.iaik.secricom.sdm.SdmKey
`SdmKey.Type`

Method Summary		***Page***
`void`	**`addHostConfiguration`**`(HostConfiguration config)` Adds a new `HostConfiguration`.	96
`boolean`	**`containsHostConfiguration`**`(HostConfiguration config)` Tests if this key already contains the specified `HostConfiguration`.	96
`Set<HostConfiguration>`	**`getHostConfigurationIdentifiers`**`()` Lists all `HostConfiguration` known to this key.	97
`boolean`	**`removeHostConfiguration`**`(HostConfiguration config)` Removes a `HostConfiguration` from this key.	97

Methods inherited from interface at.iaik.secricom.sdm.SdmKey
`getKeyIdentifier`, `getSdmKeyMaterial`, `getType`

Method Detail

addHostConfiguration

```
void addHostConfiguration(HostConfiguration config)
                  throws SdmException
```

Adds a new `HostConfiguration`. By adding the `HostConfiguration` to the `ConfigurationProtectedKey` , it will release its protected key to any host that can proof that it is in the specified platform configuration.

Parameters:

`config` - the new `HostConfiguration` to add

Throws:

`SdmException` - if any other SDM related error occurs

`SdmUnknownHostException` - if the SDM can't find the parent host of this key

`SdmUnknownKeyException` - if the SDM can't find this key

`SdmInvalidKeyTypeException` - if the key does not support this protection mechanism

`SdmInvalidParameterException` - if the specified `HostConfiguration` is null

---

containsHostConfiguration

```
boolean containsHostConfiguration(HostConfiguration config)
                          throws SdmException
```

Tests if this key already contains the specified `HostConfiguration`.

Parameters:

`config` - the `HostConfiguration` to test for

Returns:

true if the `HostConfiguration` is known to this key, false otherwise

Throws:

`SdmException` - if any other SDM related error occurs

`SdmUnknownHostException` - if the SDM can't find the parent host of this key

`SdmUnknownKeyException` - if the SDM can't find this key

`SdmInvalidKeyTypeException` - if the key does not support this protection mechanism

`SdmInvalidParameterException` - if the specified `HostConfiguration` is null

---

removeHostConfiguration

```
boolean removeHostConfiguration(HostConfiguration config)
                         throws SdmException
```

Removes a `HostConfiguration` from this key. By removing the `HostConfiguration` the key will not be released to platforms in this platform configuration any more.

Parameters:

`config` - the `HostConfiguration` to remove

Returns:

true if the `HostConfiguration` was successfully removed, false otherwise

Throws:

`SdmException` - if any other SDM related error occurs

`SdmUnknownHostException` - if the SDM can't find the parent host of this key

`SdmUnknownKeyException` - if the SDM can't find this key

`SdmInvalidKeyTypeException` - if the key does not support this protection mechanism

`SdmInvalidParameterException` - if the specified `HostConfiguration` is null

---

getHostConfigurationIdentifiers

```
Set<HostConfiguration> getHostConfigurationIdentifiers()
                                              throws SdmException
```

Lists all `HostConfiguration` known to this key.

Returns:

an immutable list of `HostConfigurations` known to this key

Throws:

`SdmException` - if any other SDM related error occurs

`SdmUnknownHostException` - if the SDM can't find the parent host of this key

`SdmUnknownKeyException` - if the SDM can't find this key

`SdmInvalidKeyTypeException` - if the key does not support this protection mechanism

`SdmInvalidParameterException` - if the specified `HostConfiguration` is null

Class HostConfiguration

at.iaik.secricom.sdm.admin

java.lang.Object

└ **at.iaik.secricom.sdm.admin.HostConfiguration**

---

public class **HostConfiguration**

extends Object

HostConfiguration represents a valid platform configuration. A valid platform configuration is represented a 20 bytes long array of bytes. A valid platform configuration is measured by using a platforms TPM and building a chain of trust. In a platform quote operation the TPM signs a SHA-1 hash of all selected *Platform Configuration Registers* which in turn contain SHA-1 hashes that represent a platform configuration. A valid platform configuration is just such a SHA-1 hash of a set of PCRs.

During key release of a configuration protected key the SDM compares the set of valid platform configurations associated with such a key with the data send in the TPM quote operation.

Constructor Summary	Page
HostConfiguration(byte[] configuration) Constructs a new HostConfiguration with the specified configuration byte[].	99

Method Summary		Page
boolean	**equals**(Object obj)	99
byte[]	**getConfiguration**()	99
int	**hashCode**()	99
String	**toString**()	99

Constructor Detail

HostConfiguration

public **HostConfiguration**(byte[] configuration)
throws SdmInvalidParameterException

Constructs a new HostConfiguration with the specified configuration byte[]. A valid platform configuration is a 20 bytes SHA-1 hash.

Parameters:

configuration - the 20 bytes SHA-1 hash representing the valid platform configuration

Method Detail

getConfiguration

```
public byte[] getConfiguration()
```

Returns:

the 20 bytes SHA-1 hash representing the valid platform configuration

---

hashCode

```
public int hashCode()
```

Overrides:

`hashCode` in class `Object`

---

equals

```
public boolean equals(Object obj)
```

Overrides:

`equals` in class `Object`

---

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

Interface SdmAdministrationSession

at.iaik.secricom.sdm.admin

```
public interface SdmAdministrationSession
```

`SdmAdministrationSession` grants access to the administration interface of the *Secure Docking Module (SDM)*.

The SDM administration interface includes functions to add new hosts, new protected keys and new valid configurations, as well as functions to update and remove existing values.

An SDM host is an entity with a unique identifier, its own *Host Authentication Key (HAK)* and a specific *Attestation Identity Key (AIK)*. An SDM protected key is a binary data blob with an unique identifier. Such a key must be protected by either an authorization token or a specific platform configuration, or both.

These objects (hosts, keys, authorization tokens, and valid platform configurations are grouped hierarchically. A host contains keys. A key may contain valid platform configuration values, or an authorization token, or both.

The interface of the `SdmAdministrationSession` reflects this hierarchical data structure. It provides access methods to `SdmHost`s to add, get, list, and remove hosts. Hosts are identified by their unique id.

As a security measure to protect an SDM administration session, only one host in a specific configuration is allowed to administer the SDM. This administration host must be injected into the SDM after fabrication, but before issuing it to a user. An administration host specifies a certain HAK, AIK and valid platform configuration state.

In order to establish an administration session, a shared session key is negotiated to protect the communication between the host and the SDM using the premeditated HAK. After this the host must perform a TPM quote to prove its platform state. This quote must then be send to the SDM. Only if the quote is deemed valid by the SDM the session is actually established.

Method Summary		***Page***
SdmHost	**addSdmHost**(SdmIdentifier hostId, RSAPublicKey hak, RSAPublicKey aik) Adds a new host to the SDM.	*102*
void	**close**() Closes this SDM session and releases all related resources.	*104*
void	**establish**(TpmQuote quote) Establishes an administration session.	*101*
Set<SdmIdentifier>	**getHostIdentifiers**() Lists all host identifiers known to the SDM.	*102*
SdmHost	**getSdmHost**(SdmIdentifier hostId) Retrieves the `SdmHost` with the specified unique identifier from the SDM.	*103*
TpmNonce	**initiate**() Initiates an administration session establishment with the SDM.	*101*
boolean	**removeSdmHost**(SdmIdentifier hostId) Removes the `SdmHost` with the specified unique identifier from the SDM.	*103*
void	**setAdministrationHost**(SdmIdentifier id, RSAPublicKey hak, RSAPublicKey aik, HostConfiguration conf) Specifies a new administration host.	*104*

Method Detail

initiate

```
TpmNonce initiate()
         throws SdmException
```

Initiates an administration session establishment with the SDM. Calling this method opens a connection to the SDM and negotiates a shared session secret using the *Host Authentication Key* specified at creation of this `SdmAdministrationSession`. Furthermore, the SDM sends back a nonce. This nonce must be used in the next step of the session establishment process, the attestation of the platform state by the hosts TPM. To actually establish an administration session it is necessary to call `establish(TpmQuote)` with the TPM quote obtained before.

Returns:

an `TpmNonce` for use with the TPM's quote operation.

Throws:

`SdmException` - If an SDM related problem occurs.

See Also:

`establish(TpmQuote)`

---

establish

```
void establish(TpmQuote quote)
       throws SdmException
```

Establishes an administration session. After a successful call to this method an administration session is ready for use. Must be called after a successful call to the `initiate()` method.

The specified TPM quote is forwarded to the SDM. The SDM verifies the signature on the quote using the pre-shared public part of the AIK. If the signature is correct the SDM checks if the nonce is the same nonce that must have been previously obtained by a call to the initiate() method. In a last verification step the actual platform configuration contained by the quote is compared to the valid platform state of the programming host. Only if all three checks succeed, an administration session with the SDM is established.

Parameters:

`quote` - the TPM signed platform configuration report

Throws:

`SdmException` - if an SDM error occurs.

See Also:

`initiate()`

---

getHostIdentifiers

```
Set<SdmIdentifier> getHostIdentifiers()
                                throws SdmException
```

Lists all host identifiers known to the SDM. This operation could either directly communicate with the SDM or read all host identifiers at session establishment. Regardless of this implementation detail the given list should always be up-to-date and must contain all hosts stored on the SDM.

Returns:

a list of host identifiers known to the SDM.

Throws:

SdmException

---

addSdmHost

```
SdmHost addSdmHost(SdmIdentifier hostId,
                   RSAPublicKey hak,
                   RSAPublicKey aik)
            throws SdmException
```

Adds a new host to the SDM. In case that a host with the same unique identifier is already stored on the SDM the values of this host must be updated. This operation must communicate directly with the SDM and immediately store the new information on the SDM. Caching of this operation by an implementation is not allowed!

It is important to note that adding a host will only update or store values that directly pertain to the host, specifically this includes its unique identifier, its authentication key (HAK) and its AIK. Therefore, it is not possible to add an `SdmHost` object.

Parameters:

`hostId` - unique identifier of the host that should be created, or updated

`hak` - *Host Authentication Key*, must match `hostId`

`aik` - *Attestation Identity Key* the hosts uses to sign its platform quotes

Returns:

an `SdmHost` object that reflects the specified values regardless if an old host with the same identifier already existed. Such old values are lost. The `SdmHost` is likewise directly connected to the SDM and will store all changes immediately to the SDM.

Throws:

`SdmException` - if an SDM error occurs during writing of the host values.

See Also:

SdmHost

---

getSdmHost

```
SdmHost getSdmHost(SdmIdentifier hostId)
            throws SdmException
```

Retrieves the `SdmHost` with the specified unique identifier from the SDM. The thus created `SdmHost` object must be directly linked to the SDM. Specifically, any changes made to the host object must be directly written to the SDM.

Parameters:

`hostId` - unique identifier of the host

Returns:

the `SdmHost` object for the specified unique identifier, or null if no host with the specified identifier exists.

Throws:

`SdmException` - if an SDM error occurs during reading from the SDM.

See Also:

`SdmHost`

---

removeSdmHost

```
boolean removeSdmHost(SdmIdentifier hostId)
              throws SdmException
```

Removes the `SdmHost` with the specified unique identifier from the SDM. The target `SdmHost`must be empty, that is it must not contain any protected keys, before it can be removed. Calling this method on an non-empty host will cause an exception.

Parameters:

`hostId` - unique identifier of the host to remove

Returns:

true if the host existed and was empty and was successfully removed; false if the host did not exist at all.

Throws:

`SdmException` - if an SDM error occurs, or if the host is not empty.

---

setAdministrationHost

```
void setAdministrationHost(SdmIdentifier id,
                           RSAPublicKey hak,
                           RSAPublicKey aik,
                           HostConfiguration conf)
                    throws SdmException
```

Specifies a new administration host. Every SDM may have exactly one administration host. A administration host is the only host which can open an administration session to the SDM.

An administration host is similar to a normal SDM host. It must specify a unique identifier, a *Host Authentication Key*, and an *Attestation Identity Key*. In addition it must also specify the one and only valid platform configuration in which it is allowed to modify the SDM.

Parameters:

`hak` - the *Host Authentication Key*, which is necessary to authenticate the administration session

`aik` - the *Attestation Identity Key*, which the host's TPM uses to sign its platform configuration reports

`conf` - the valid platform configuration in which the host is allowed to administer the SDM

Throws:

`SdmException` - if an SDM error occurs.

---

close

```
void close()
    throws SdmException
```

Closes this SDM session and releases all related resources.

Throws:

`SdmException` - if an SDM related problem occurs.

Interface SdmHost

at.iaik.secricom.sdm.admin

```
public interface SdmHost
```

`SdmHost` manages all the information the SDM needs to communicate with a host and verify the signature on the host's TPM's platform configuration reports.

`SdmHost` consists of an identifier, a *Host Authentication Key (HAK)*, and an *Attestation Identity Key (AIK)*. The identifier must be unique among all platforms that host an SDM. The HAK is used to establish an

authenticated and confidential session between the host and the SDM. The AIK enables the SDM to verify the signature on the host's platform configuration reports. The HAK and AIK are both 2048-bit RSA public keys. In addition the SDM is capable of managing keys for several distinct hosts.

The `SdmHost` is part of the administration interface of the SDM. It is needed to load new host information, or update old host information in the SDM. Each host can be associated with a number of protected keys. `SdmHost` provides methods to list, add, get, and remove keys associated with a specific host. The add method also allows to update existing keys.

`SdmHost` is connected to the SDM. Some of the operations directly query information from the SDM. Therefore, a `SdmHost` ceases to function properly if the session that created it is closed!

See Also:

SdmKey

Method Summary		Page
SdmKey	**addKey**(SdmIdentifier keyId, SdmKey.Type type, SdmKeyMaterial keyMaterial) Adds a new protected key to the host.	107
RSAPublicKey	**getAttestationIdentityKey**()	106
RSAPublicKey	**getHostAuthenticationKey**()	106
SdmIdentifier	**getHostIdentifier**()	105
SdmKey	**getKey**(SdmIdentifier keyId) Retrieves a specific key of the current host from the SDM.	106
Set<SdmIdentifier>	**getKeyIdentifiers**()	106
void	**removeKey**(SdmIdentifier keyId) Removes a protected key from the host.	107

Method Detail

getHostIdentifier

SdmIdentifier **getHostIdentifier**()

Returns:

the unique host identifier

---

getHostAuthenticationKey

RSAPublicKey **getHostAuthenticationKey**()

Returns:

the *Host Authentication Key (HAK)*, which is a 2048-bit public RSA key

getAttestationIdentityKey

```
RSAPublicKey getAttestationIdentityKey()
```

Returns:

the *Attestation Identity Key (AIK)* the host's TPM uses to sign its platform configuration reports (TPM Quotes).

getKeyIdentifiers

```
Set<SdmIdentifier> getKeyIdentifiers()
                              throws SdmException
```

Returns:

a set of key identifiers for this specific host, retrieved from the SDM.

Throws:

`SdmException` - if an SDM related problem occurs, during retrieval of the list of keys.

getKey

```
SdmKey getKey(SdmIdentifier keyId)
       throws SdmException
```

Retrieves a specific key of the current host from the SDM.

Parameters:

`keyId` - the unique identifier of a key the SDM protects for this host. A key identifier must be unique per host.

Returns:

the `SdmKey` with the specified identifier if one exists

Throws:

`SdmException` - if an SDM related problem occurs, during retrieval of the `SdmKey`, or if the key does not exist.

addKey

```
SdmKey addKey(SdmIdentifier keyId,
              SdmKey.Type type,
              SdmKeyMaterial keyMaterial)
       throws SdmException
```

Adds a new protected key to the host. By adding the key to a specific host, only this host will be able to retrieve the key from the SDM. This method actually creates a blank `SdmKey` object from the specified key unique identifier, the key protection mechanism type and the key material that must be protected by the SDM. The information will also be written directly to the SDM. Depending on the key protection mechanism type an authorization token, a set of valid configurations, or both must be specified in order to protect key access. To do this it is possible to cast the `SdmKey` to either `ConfigurationProtectedKey` or `TokenProtectedKey` depending on the specified protection mechanism.

Parameters:

`keyId` - the unique identifier of a key the SDM protects for this host. A key identifier should be globally unique, but must be unique per host.

`type` - the protection mechanism type

`keyMaterial` - the actual key material to protect

Returns:

the freshly created `SdmKey` object which also implements the `ConfigurationProtectedKey` and/or `TokenProtectedKey` depending on the chosen protection mechanism.

Throws:

`SdmException` - if key creation in the SDM fails.

See Also:

`ConfigurationProtectedKey`, `TokenProtectedKey`, `SdmKeyMaterial`, `SdmKey.Type`

---

removeKey

```
void removeKey(SdmIdentifier keyId)
        throws SdmException
```

Removes a protected key from the host. This actually erases the key from the SDM. In order for a key object to be erasable it must be empty, that is there must not be any valid platform configurations left that are associated with the key. If the key protection mechanism does not use valid platform configurations, this does not apply. The key material associated with the key and possibly its authorization token are deleted automatically.

Parameters:

`keyId` - the unique identifier of a key the SDM protects for this host. A key identifier must be unique per host.

Throws:

`SdmException` - if removal of the `SdmKey` from the SDM fails. The main reasons for such an exception are the no key with the specified identifier can be found or the key is not empty.

See Also:

HostConfiguration, ConfigurationProtectedKey, SdmKeyMaterial

Class SdmUnknownHostException

at.iaik.secricom.sdm.admin

java.lang.Object

└ java.lang.Throwable

└ java.lang.Exception

└ at.iaik.secricom.sdm.SdmException

└ **at.iaik.secricom.sdm.admin.SdmUnknownHostException**

All Implemented Interfaces:

Serializable

---

public class **SdmUnknownHostException**

extends SdmException

SdmUnknownHostException indicates that an SDM operation failed, because it tried to operate on an unknown SDM host.

Constructor Summary	Page
SdmUnknownHostException() Constructs a new SdmUnknownHostException with null as its detail message.	109
SdmUnknownHostException(String message) Constructs a new SdmUnknownHostException with the specified detail message.	109
SdmUnknownHostException(String message, Throwable cause) Constructs a new SdmUnknownHostException with the specified detail message and cause.	109
SdmUnknownHostException(Throwable cause) Constructs a new SdmUnknownHostException with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().	109

Constructor Detail

SdmUnknownHostException

public **SdmUnknownHostException**()

Constructs a new `SdmUnknownHostException` with null as its detail message. The cause cause is not initialized. A call to the `initCause(Throwable)` method may be used to initialize the cause after exception creation.

SdmUnknownHostException

```
public SdmUnknownHostException(String message,
                               Throwable cause)
```

Constructs a new `SdmUnknownHostException` with the specified detail message and cause.

Parameters:

`message` - the detail message

`cause` - the cause. A null value is permitted and indicates that the cause is nonexistent or unknown.

SdmUnknownHostException

```
public SdmUnknownHostException(String message)
```

Constructs a new `SdmUnknownHostException` with the specified detail message. The cause cause is not initialized. A call to the initCause(Throwable) may be used to initialize the cause after exception creation.

Parameters:

`message` - the detail message

SdmUnknownHostException

```
public SdmUnknownHostException(Throwable cause)
```

Constructs a new `SdmUnknownHostException` with the specified cause and a detail message which is null if the cause is null; otherwise the detail message is cause.toString().

Parameters:

`cause` - the cause of this exception

Interface TokenProtectedKey

at.iaik.secricom.sdm.admin

All Superinterfaces:

SdmKey

```
public interface TokenProtectedKey

extends SdmKey
```

TokenProtectedKey is a key, the release of which is at least protected by an authorization token. The SDM offers three protection mechanisms, two of which require an authorization token. This decorater interface defines the method a key object must support to provide authorization token based protection. A key can be protected by both an authorization token and a valid platform configuration. In this case the key must implement both this and the ConfigurationProtectedKey interface.

See Also:

ConfigurationProtectedKey, SdmToken, SdmKey

Nested classes/interfaces inherited from interface at.iaik.secricom.sdm.SdmKey
SdmKey.Type

Method Summary		Page
void	**setAuthorizationToken**(SdmToken token) Specifies the authorization token for a key.	111

Methods inherited from interface at.iaik.secricom.sdm.SdmKey
getKeyIdentifier, getSdmKeyMaterial, getType

Method Detail

setAuthorizationToken

```
void setAuthorizationToken(SdmToken token)
                    throws SdmException
```

Specifies the authorization token for a key. This operation is write only.

Parameters:

token - the authorization token that governs release of this key. An authorization token is a byte[] with a maximum length of 20 bytes.

Throws:

SdmException - if the operation fails due to an SDM related problem.