D6.3 Collection of all SWP deliverables (with nature=R) produced during month 13-24

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Deliverable

Deliverable reference number	IST-027635/D06.3/FINAL | 1.00
Deliverable title	D6.3 Collection of all SWP deliverables (with nature=R) produced during month 13-24
WP contributing to the deliverable	WP6
Due date	Oct 2007
Actual submission date	Nov 2007

Responsible Organisation	LDV,Lehrstuhl für Datenverarbeitung, TUM
Authors	Chun Hui Suen
Abstract	
Keywords	OpenTC WP6

Dissemination level	Public
Revision	FINAL | 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

This document is a compilation of the following deliverables:

- D06a.3 Final DRM system specification (M 24)
- D06b.2 A report containing the MEITC specification and test plan (M13)
- D06b.3 Detailed design and test document (M15)
- D06c.1 High level requirements specification (for Proof-of-Concept WYSIWYS application) (M 24)
- D06d.1 EFS C/C++ API Specification (M18)
- D06e.4 Final MFA System Specification (M 18)

Note: The deliverable D06d.1 has been placed in the annex, as it is still incomplete and will be revised in the next phase of the project. It is deemed not suitable for public release and will thus not be included in the public dissemination of this document.

WP06a.3 Final DRM system specification

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Internal document

Deliverable reference number	IST-027635/D06a.3/FINAL | 1.00
Deliverable title	Final DRM system specification
WP contributing to the deliverable	WP6
Due date	Oct 2007
Actual submission date	Oct 2007

Responsible Organisation	LDV,Lehrstuhl für Datenverarbeitung, TUM
Authors	Chun Hui Suen, Florian Schreiner
Abstract	
Keywords	DRM, fair, interoperable, MPEG-21

Dissemination level	Public
Revision	FINAL | 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

List of figures

1. Introduction

This document collects the specifications of a DRM system to be developed as sample application for the OpenTC framework. These specifications define the scope of system, describe its functional requirements and its design. The design sections of this document are mainly focused on the definition of the system architecture by depicting the system modules, the function of each of them and the related interactions. Specific details of the API and communication protocol are still subject to change depending on the interfaces provides by the underlying OpenTC framework developed in workpackage 3 to 5 and implementation issues to be resolved.

The principal scope of the DRM system will be the protection of multimedia content. Generalization of the DRM system for the protection of other contents, such as personal data, secret information or medical records of the patients, would be possible through extension of the DRM system. However, specific implementation of such generalization will not be implemented in this sub-workpackage.

This document is organized into 9 sections. Section 2 describes the functional requirements in terms of use cases while sections 3, 4 and 5 include the design specifications of the system. Section 6 contains the API specification and section 7 shows the requirements for this application within the OpenTC System. Sections 8 and 9 provide glossary and references to the terms and technologies used in the DRM system. And external companion document contains the detailed API specification.

2. Use Cases

2.1 Overview

The Interoperable DRM system application scenario describes a DRM system that is based on Trusted Computing and MPEG-21 for protecting multimedia content. The system can be divided in 2 main parts: the DRM Core and the secure application.

The DRM-core runs as an independent domain that handles the content licenses and the content keys. It exposes this functionality through an application programming interface (DRM Core-API) to applications. The DRM-core is responsible for parsing licenses, deciding on whether access to requested content is allowed and managing the content keys. The core also support the management of user domains, which allows a user to play content on multiple platforms, that belong to his domain. The core handles the registration of other platforms to a domain and issues domain licenses for these peers.

The secure application is in the simplest case a media player. The application uses the DRM Core-API provided by the DRM-core to gain access to protected content. After a verification process, the application receives the content key from the DRM Core and is able to render the content.

The user can perform different actions with the secure application. Every action triggers a process between the application and the DRM Core. For the DRM system we differentiate between these 5 main use cases:

- Installation of the system
- Download content

- View / Consume Content
- Renew License
- Transfer License

In the following sections these different use cases are explained in detail. They describe step by step, what happens when the user intends to perform an action.

2.2 Description of Use Cases

Use Case Unique ID	/ UC 10 /
Title	Installation of the system
Description	The administrator installs the DRM System within the OpenTC framework.
Actors	Administrator
Preconditions	The OpenTC framework was started.
Postconditions	The DRM Core is installed/initialized.
Comment	
Normal Flow	1. The administrator installs a DRM-Core and starts it in a separate compartment. 2. The administrator installs the DRM player in a separate secure environment for rendering of the content. 3. A signed policy file establishes the following requirements: • Trusted I/O Channels: We need a secure audio and video output path for rendering content. • Access to the trusted services from the compartment, especially to the DRM Core. Access to the Core will be limited by its API. • Ability to display an application in a Window-System, which is started in the secure environment. An efficient method for video rendering should also be supported in a secure manner (for example Overlay). • Integrity measurement of all applications and plugIns that can be used to reproduce content in a secure way.

Use Case Unique ID	/ UC 20 /
Title	Download content and install license
Description	The user downloads a content.
Actors	User
Preconditions	The OpenTC framework was started and the DRM Core is running in a secure environment.
Postconditions	The content keys and the license are kept secure in the sealed storage.
Comment	
Normal Flow	1. The user downloads a container file either from a provider or another user. The file consists of the multimedia content. The downloading and the storage can be unsecured, because the data is always encrypted. The license can also be transferred in this step. It doesn't need to be protected, since it is signed by the content provider. 2. The user starts the secure environment. 3. The user starts the player application for the retrieval of the content keys. 4. The player application establishes a secured connection to the provider for exchanging the content keys. 5. The DRM Core generates an attestation identity keys (AIKs) and encryption keys using the TPM, so that the content provider can encrypt the content key. 6. The user receives the content with an embedded license. This license is forwarded to the DRM core. 7. The license for the content is checked and preprocessed in the DRM core. The encrypted content is decrypted and stored in sealed storage. The important information of the license is stored in the sealed store.

Use Case Unique ID	/ UC 30 /
Title	View / Consume content
Description	The user tells the player that he wants to view or consume the content of a protected file.
Actors	User
Preconditions	The OpenTC framework was started and the player application and DRM Core are running in a secure environment.
Postconditions	
Comment	
Normal Flow	1. The user starts a player application, which runs in the secure environment. 2. The user triggers the application to access a protected media file for rendering. 3. The player application registers with the DRM-Core. Then it asks the DRM-core through the API to enable access to the protected information by handing out the content key from the key store. 4. The Core is presented with the content's unique DII (Digital Item Identifier) along with the requested action (e.g. play, print, burn etc.) and decides on whether access is granted or not. If yes, the DII is used to query the key store for the content key. The key store itself is an encrypted file and is protected by sealing its key to a trusted system configuration. Thus, the core can only access the key store when the system is in a known trusted state. 5. Then the DRM-Core hands out the content key to the application. It poses no threat since the system and the player application are trusted.

Use Case Unique ID	/ UC 40 /
Title	Renew License
Description	Generally licenses are valid until a final date. After this date, the license expires and the user has to renew his license from a license server.
Actors	User
Preconditions	The OpenTC framework was started and the DRM Core is running in a secure environment.
Postconditions	New license is stored securely in the sealed storage.
Comment	
Normal Flow	1. The user triggers the license renewal and the player application connects to the content provider. 2. The DRM-Core performs an authentication procedure similar to that in the download procedure. 3. The player application replaces the existing license by a new one.

Use Case Unique ID	/ UC 50 /
Title	Transfer License
Description	Licenses are transferred to other computers or are translated to other DRM-Systems.
Actors	User
Preconditions	The OpenTC framework was started and the DRM Core is running in a secure environment. Manager application and target DRM system are running in a secure environment.
Postconditions	Transferred license is stored secure in target DRM system.
Comment	
Normal Flow	1. The user initiates a transfer. 2. Then the Manager application establishes a secure and authenticated connection between the two systems. The license and content key are transmitted securely. A similar authentication procedure as in the download and renew license use case is required. 3. In case a different DRM system needs to be supported, the existing license must be translated by the DRM Core. The translation may also require a re-encryption of the content. Furthermore, the translated license has to be signed by the DRM Core, which will use the TPM to enable trust to its signature. 4. The player application transmits the content itself. This is not a security problem, since the transferred data is always encrypted.

3. Design Specifications

3.1 Architecture

The diagram below shows the major components which make up the OpenTC DRM system. The entire system can be divided into 3 sections, namely applications running in userspace, the DRM Core which is running in a secure compartment, and security services provided by the operating system and compartment management. The precise separation of the system components among different secured compartments is explained in section . The following sections will explain the individual components of the system in detail.

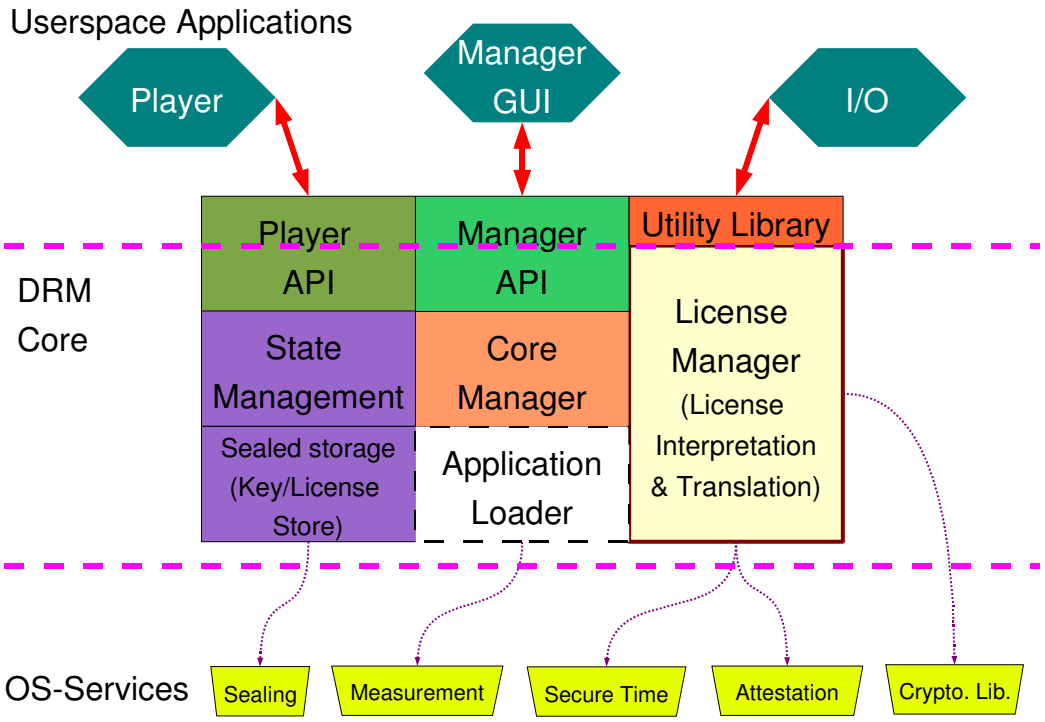


Figure 1: System Overview

3.2 Player API and Player application

The Player API is used for two purposes: the registration of a player application (Player) and the content authorization. The following sections describe these two functionalities in details.

3.2.1 Registration

Each Player who wants to access a protected content must register with the DRM Core first. During registration, the Player and DRM Core starts a secure channel which can only be opened if the core and player are running in a trusted state (provided by OpenTC framework).

Additionally, the Core gets some information about the player, e.g. the version number or process information, so that the Core can distinguish between multiple players on

the same machine. After the authentication the capability negotiation follows, where the core negotiates a common rule set with the Player instance. This rule set defines the REL commands, that both, player and core, have a common understanding of. This mechanism enables the Core to discover, which commands the player supports and in what way the Core can control the Player.

This restriction description can be done by using the subset of REL commands related to representing conditions on operations, time and state. This allows a well defined command set to be used, without defining a new standard.

After successful registration, the player is considered trustworthy to handle the protected content in a correct and predictable way.

3.2.2 Content Authorization

After a successful registration the player can request the content key for a particular protected content. This triggers a process of retrieving the associated license(s) of the selected content and interpretation of this license. The Core then comes to the decision if the player is allowed to access the content or not. This “Result” is described in a XML format similar to the MPEG-21 REL and is transmitted to the Player.

If the Result is positive and the Player is generally allowed to access the content. Together with a positive Result, the Core also transmits the content key so that the Player can decrypt the content.

Furthermore the Result may contain several additional conditions, which have to be enforced during the process of rendering. The content provider can define these conditions to specify in what way the content can be rendered. An example condition would be that the player should play only the first 10 seconds of a song. The player has to understand these conditions in order to be able to enforce it correctly.

The capabilities of the player for these conditions are negotiated during the registration, so the Core knows which conditions the player is able to enforce. For example during the registration, the player informs the core, that he is able to enforce the rule “play only the first x seconds” and the Core saves this property in an internal storage. When a license is validated and this rule should be applied for the value 10, then the Core generates a Result, which contains the rule that states “play only the first 10 seconds”.

Decryption Modules are needed, when the Player receives a positive Result and then wants to decrypt a specific content. Generally every content can use its own encryption algorithm depending on the producer of the content. If the Player wants to decrypt these contents, he needs access to all corresponding encryption libraries. This functionality is provided by the Utility Library, which the player can use to get access to a corresponding implementation of the encryption algorithm. The Utility Library is a part of the API and provides a standardized interface for essential algorithms. The mechanism within the Utility Library is explained in section 3.9.

3.2.3 Legacy Player Application

All specifications in the API are standardized and can be used by the player applications. Generally the Player should be compatible to the DRM-System to know the API of the core and how to handle content. An optional feature is to provide support for legacy players, which cannot access the API directly. Players of that kind

are not aware of the DRM Core, but maybe favored by users for whatever reason. These cases are handled by capturing the file reading operation and redirecting the request to the DRM core during the read cycle. In this way, the handling of license authentication and interpretation occurs transparent to the application.

For the player, the whole process is similar to a normal file access. The player receives the unprotected content from the socket and can render it. The file access capturing in this case converts and forwards requests through the API to the DRM Core. Since all applications, including the legacy ones, run in the secured environment, handing out the content key or the decrypted content itself is no problem, since it is guaranteed that the applications cannot compromise it.

3.3 Manager API

Manager API provides an interface to management features of the DRM Core, The provided functionalities can be divided in the following four categories:

- License Management:
  - Insertion of new licenses into the DRM Core
  - Renewal of existing licenses
  - Deletion of invalid or expired licenses
- License Transfer:
  - Generation of a Transfer Licenses to an external peer
  - Request for attestation keys
  - Generation of signed certificates
- User Management:
  - Adding and Removing users of the system
- Domain Management:
  - Registration and de-registration of Domains

License Management is used to update the license storage of the DRM Core. For a new content, the respective licenses can be added and for an expired content, the corresponding license can be renewed or removed. The License Transfer functions provide information for the acquisition and transfer of licenses to external peers. The Domain Management and User Management functions allow the Manager GUI to update the current domains and the users of the system.

The Manager API (defined in `ManagementInterface`) differs from the Player API, so that playback and administrative functions of the DRM Core are clearly separated.

3.4 Application loader

The initial loading of the DRM Core needs to be done in a secure manner. This should be handled by the compartment and device manager, which will check the integrity of the compartment image before loading the DRM Core. In addition to the main image, a secure persistent storage is used to provide secure storage for the DRM Core, that will be discussed in sections and .

3.5 Core Manager

The central component of the system is the Core Manager. It's tasks are the central management of the different parts of the DRM-Core. It coordinates the requests from the application layer and forwards them to the appropriate components. It also contains the error handling such as fail over, treatment of invalid data, error logging and exception handling.

3.6 License Manager

The core manager implements the interface to the Player and Manager GUI, and coordinates the management and enforcement of licenses. When the player wants to decrypt a protected content for a particular action, it sends a request to the core manager, with a reference to the protected content and the request parameters. This request contains the rights and the corresponding license, which have to be verified. A request may also consist of multiple licenses.

Upon request from the player, the DRM Core makes the appropriate query to the key / license storage, and sends the complete request to the license manager. In this case the License Interpreter has to verify each license and determine if the right may be granted over the content.

Depending on the license type, this is performed by the appropriate license interpreter, generating an internal representation of the license. When the license is positively authorized, the content key is retrieved from the key storage and returned to the player with an appropriate player restriction description.

When this player restriction needs to be adapted, or if a license is requested, then the query is passed to the license translation manager.

3.6.1 License Interpreter

The licenses, that are stored in the sealed storage are in an XML format. Before these licenses can be interpreted, license parsing needs to be carried out. This process maps a license into an internal representation suitable for interpretation.

The parsing process takes place in two steps. First, the formal integrity of the received data is validated, for e.g. XML-formatted licenses this includes schema- or DTD-validation.

In the second step the authenticity and integrity of the data must be validated. The most utilized approach is using digital signatures on the license, like XML dsig, together with X509 based certification chains. To leave the possibility to extend the concept to new formats, the signature checking uses the utility library as plug-in architecture for the verification.

After the parsing of the license, the interpretation can be performed. In this process the internal representation of the license is matched against the operation request from the player application. The matching returns either a positive or negative result. A positive result implies that the player application is allowed to decrypt and render the specified content. However, depending on the license, a positive result may also include additional restrictions which the player must support and enforce.

The OpenTC license model strives to support the concept of a “fair” DRM system as well. This means, that the content creator has the possibility to create licenses, that

are beneficial not only for the content provider, but also the consumer of a content.

The DRM system is designed in such way, that all participants will be treated equally, so that every participant can either act like a content consumer or a content provider. A content provider can use the system to protect his own creation against any misuse. Nevertheless the content provider can still decide to restrict the usage of the content in an “unfair” way. This decision isn't based on a technical problem, but rather a consequence of the business model. In order to have a fair usage of DRM, each participant has to consider carefully its business model. The business model should provide different added-value to the user, by granting additional rights to the user. We foresee the following rights, which would support a “fairer” usage of DRM:

- copy
- burn
- sell

With the right to “copy”, the consumer can create a limited amount of private copies. By transferring these copies, the content can be shared with a small number of OpenTC devices, which belong to the domain of the user. This domain is defined beforehand in the license, which contains a specified maximum number of devices within the domain.

In the same way, the right “burn” grants the user to save the content on a disc. “Sell” means, that a consumer can sell the content to another user. With these technical possibilities, the DRM works almost transparently to the consumer.

3.6.2 License Translation Manager

In order to support interoperability between different systems, we propose to include a license translation system, to support the translation of licenses between different license description schemes, e.g. Open Mobile Alliance (OMA) REL, Digital Video Broadcast Content Protection or Content Management DVB-CPCM. This allows content to be received from or exported to foreign DRM systems or to external devices which do not support the MPEG-21 REL license format.

This enables a seamless experience for the user, by allowing multimedia content to easily move between different interoperable systems and devices.

The parsing of the license to be translated is first performed, which creates an internal representation of the license. This is then handed to the translation engine with the required translation requirements, such as target license language and profile.

Requirements for the translation system are:

- Element name translation / adaptation
- Restructuring of license elements to a legal structure in the other language
- Contractive translation of unsupported elements

Figure 2 shows the translation between two license languages. Element renaming can be handled trivially, but restructuring and contractive translation (where an alternative description must be generated that best matches the original element) of elements not found in the original language, will require intelligent rules for such transformation.

The proposed solution is to use an expert system architecture to transform a knowledge representation of the original license into another license language. Transformation rules can be built to translate the element names, make appropriate contractive translation of elements which are not found in the target language, and an output phase which generates the output license in a different structure.

Figure 2: License Translation

3.7 State Management

The State Manager is responsible for managing system and license-related states. System states are a general framework to access information related to the DRM system (such as current player capabilities and credentials) and machine-related parameters (such as time and location). License-related states are used to store persistent information needed for license interpretation (such as playback counter).

3.8 Sealed storage

The sealed store consists of three parts: the key store, the license store and the user store.

3.8.1 Key store

A particularly important component of the core is the key store. The key store contains the keys which are used to access (namely decrypt) the protected content in the system. The DRM Core ensures that a content key is given out only when a requested action is allowed by the license. The key store is organized as a table which contains keys and unique content identifiers. The same identifiers are used in the licenses to reference content. Respective technologies are part of the MPEG-21 standard. The key store is implemented as a secure database, which is decrypted by the core when a secure environment is established. This is done with the help of the TPM, which seals the key storage master key, so that it can only be accessed when the system is in a secure state. The core itself is thus only able to retrieve the master key when the system has not been compromised.

3.8.2 License store

As described previously, License Interpretation Manager relies on an internal representation of licenses. The structure of this Internal license store is similar to the structure defined in MPEG-21. To speed up the evaluation of licenses by the License Interpretation Manager, each single syntactic object of a license, namely principal, digital item, grant and condition, is mapped to a specific internal object representation that is optimized for the evaluation process. The internal storage offers some basic search methods on the storage objects for selecting certain items based on different criteria or for matching two items against each other. The license is also stored in the secure database, to protect against any unauthorized change to the license outside of the DRM Core. Regarding the semantic of the stored elements, we strictly use values from the RDD-Standard issued within the MPEG-21 framework.

3.8.3 User store

The User store contains credentials of the user of the system. This information is needed to authenticate different consumers and to verify if a consumer is allowed to access the content. The storage contains an identifier of every user and public/private key pair for the verification of signatures. The user authentication depends also on the user management of the underlying operating system, so that the credentials might change or additional information might be needed.

3.9 Utility library

In order to support an extensible DRM system, a utility library is provided to both the player application and the DRM Core. This utility library provides a centralized mechanism in which new tools for decoding, encryption, decryption, signing, and so on, can be retrieved and made available.

The Player Application can request a decryption tool from the Utility library to be able to decode the content. The DRM Core may also need cryptographic tools, for signature verification or self-signing generated licenses (for instance, in the case of license translation from another DRM system).

The Utility library follows the concept of MPEG-21 IPMP tools, in which tools for specific functions can be identified and automatically retrieved for the target platform. This allows the DRM Core and player to support new media (new codecs) and licenses (new cryptographic tools) when newer tools become available.

An important security aspect is that this utility library itself must be verified beforehand, and must run within a secured environment. Mechanism to verify the integrity of the retrieved tools, such as tool signing, must be implemented to ensure that the tool cannot be modified to introduce security weaknesses.

3.10 OS Services

The necessary OS services required by the DRM Core are secure time, sealing, compartment measurement, attestation, cryptographic libraries. Secure time mechanism provides a trustworthy source of time, on which time-related license conditions can be verified.

Sealing of the license and key stores of the DRM Core, and measurement of the DRM

Core compartment should be performed by the OS compartment manager, prior to the starting of the DRM Core compartment.

Services to aid the attestation of the DRM Core to services on the Internet, such as the generation of AIK keys, need to be provided by the underlying framework.

Standard cryptographic libraries are also necessary in order to perform decryption and hash operations as required by the DRM Core and Player application.

4. XEN/L4 virtualization environment

4.1 Compartment architecture

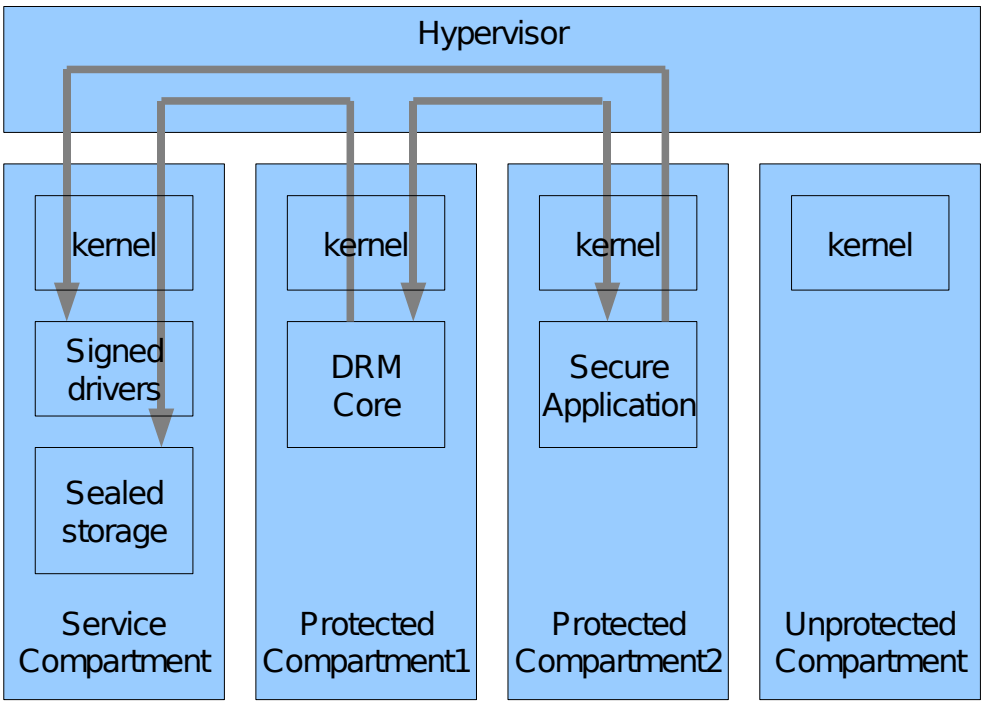


Figure 3: Virtualization of the DRM Core

In order to take advantage of the secure application isolation provided by the virtualization framework in OpenTC, higher security can be achieved by separating the player application and DRM Core into separate compartments. Figure 3 shows virtual machine partitioning of different components. The DRM Core as described in section 3 runs in a protected compartment, while the OpenTC Player runs in a different protected compartment. Since the information traffic between the DRM Core and player is not high, this is not a big performance penalty. The hypervisor, and OS components such as kernel and drivers are not described in this document.

For the rendering of the content, the player needs access to device drivers/kernel modules. This access is controlled by security policies which only allows communication with signed device drivers/kernel modules in the service compartment. This enforces the secure output path criteria. The DRM Core has access to a secure storage provided by the service compartment. Sealing is used to encrypt this storage, such that the DRM Core can only access it when the OS and the DRM Core are not modified.

4.2 Interfaces between compartments

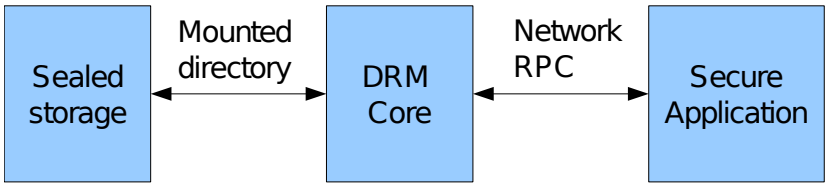


Figure 4: Interface Chain

A generic way to achieve communication between two compartments is the definition of a network RPC between them. This form is used for the connection from the secured application to the DRM Core. The security policy of the channel can be defined via an interface from the operating system. Furthermore some rules of the license may have to be applied, e.g. the content may not be rendered at the same time in more than one player application. XML-RPC [7] is the RPC protocol used in this case, as it provides a simple implementation as well as widely available cross-platform libraries for binding with many languages. This RPC interface exposes methods from the *PlayerInterface* and *ManagementInterface.*

The interface between the DRM Core and secure sealed storage is implicit, in that it is achieved by mounting secure mount points within the compartment of the DRM Core. This is controlled by the compartment and device manager in service compartment. The sealed storage is used for the storage of the licenses and the content keys.

5. Component Interaction within the DRM system

5.1 Functional parts of the DRM Core

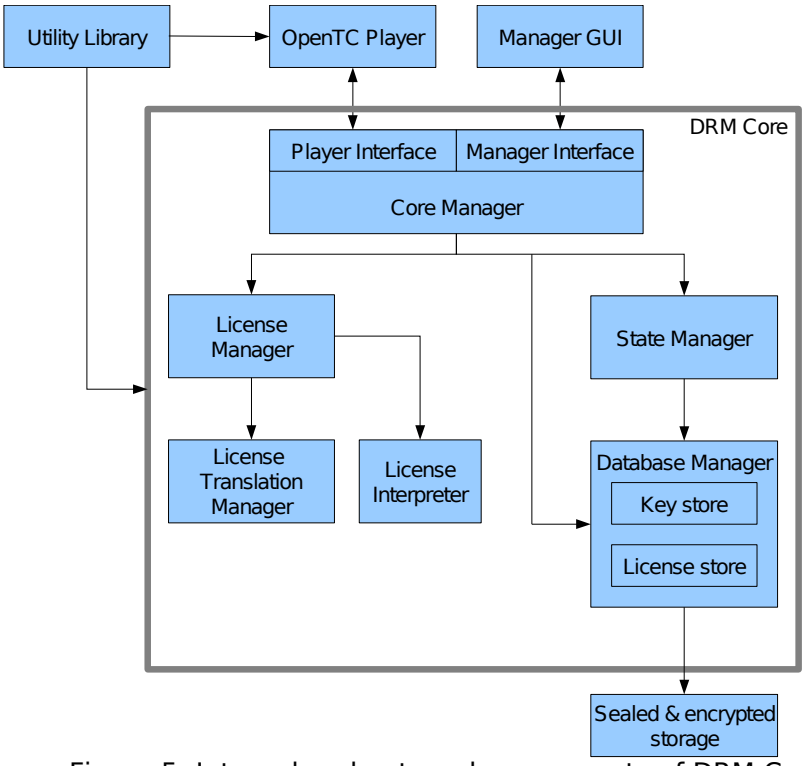


Figure 5: Internal and external components of DRM Core

The DRM Core consists of five key functional parts: The Core Manager, License Manager, License Translation Manager, State Manager and Database Manager. The Database Manager is a component that provides the access to the sealed storage. Figure 5 shows the inter-relations of the different modules.

The Core Manager provides the API's to the user level applications. The Core Manager is directly connected to the License Manager, the State Manager and the Database Manager.

The License Manager can process licenses and then decides to which component the license should be forwarded. If a license shall be interpreted, he uses the License Interpreter, which parses the license and compares it to a given set of conditions. The License Translation Manager is used, if a license has to be converted to or from other DRM-Systems. The Manager can either import or export a license from another compatible system.

The State Manager contains the current states of the applications and contents. It monitors all players that are connected to the DRM Core and provides state information about players, system and digital items.

The Database Manager has a connection to the key store and the license store. The Core Manager can request specific keys and licenses from the Database Manager,

which are then retrieved from the key store or the license store.

5.2 Sequence diagrams

5.2.1 Player API

Figure 6 shows the sequence diagram for interaction between the player and different components within the DRM Core. The player application first performs an initial

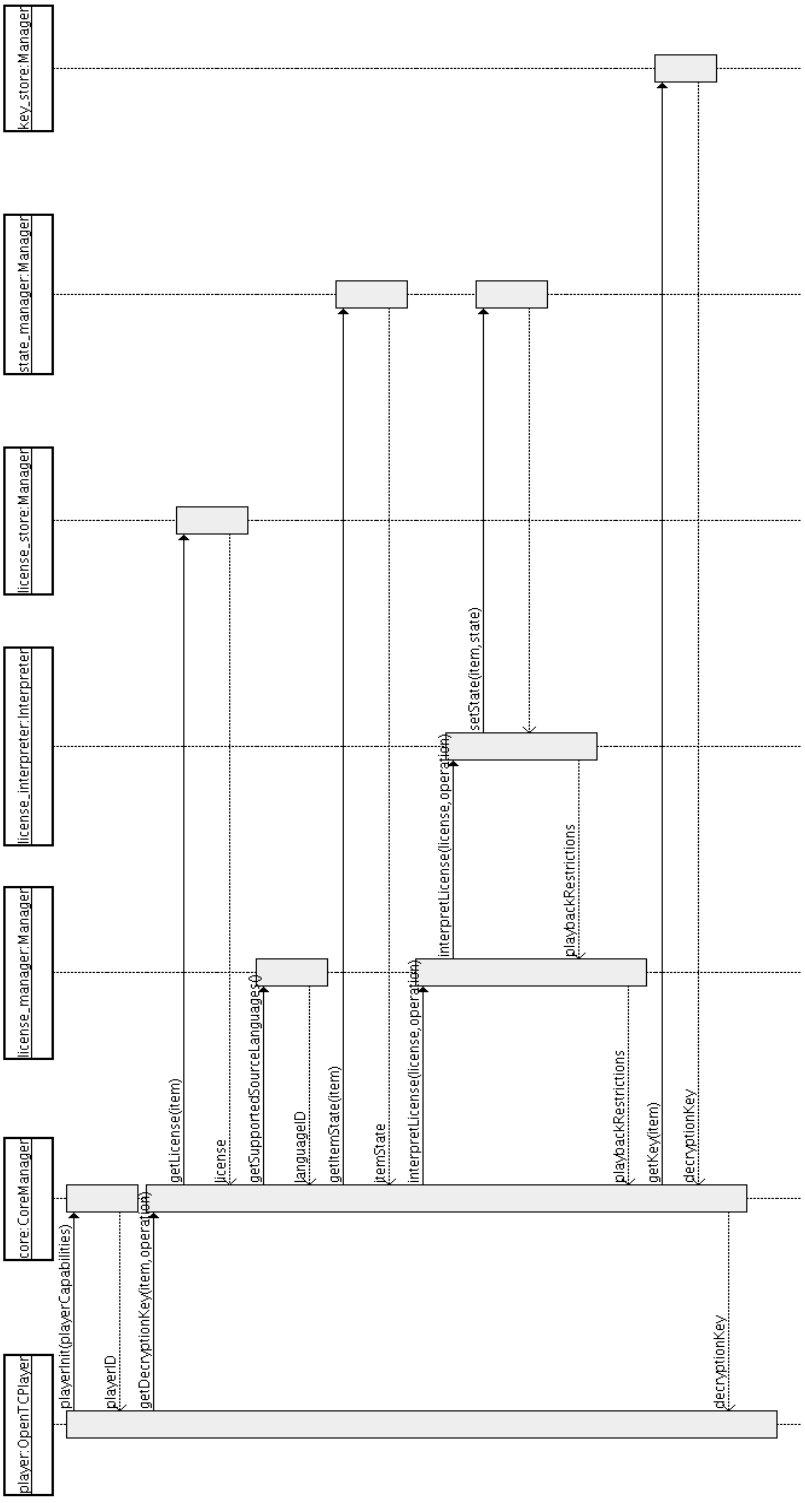


Figure 6: Sequence diagram for media playback

handshake with the DRM Core by reporting its playback capabilities, and receives as a response a `PlayerID`, which the DRM Core uses to identify different players connected to the core. Upon the player requesting to decrypt a digital item, the core manager handles the request and calls the appropriate modules within the DRM Core to process the request. Upon success, the content key is retrieved and returned to the player.

5.2.2 Management API

The functions of the Management API are processed in a similar way to the ones of the Player API. The functions called from the Management GUI are processed by the CoreManager, which passes the parameters to the corresponding component.

5.3 Interaction between different peers

5.3.1 License and Content Key transfer

A license and Content Key has to be transferred, when a content is moved from one peer to another, e.g. when a content is sold to another user.

The figure shows the sequence diagram of the key and license transfer from user B to user A.

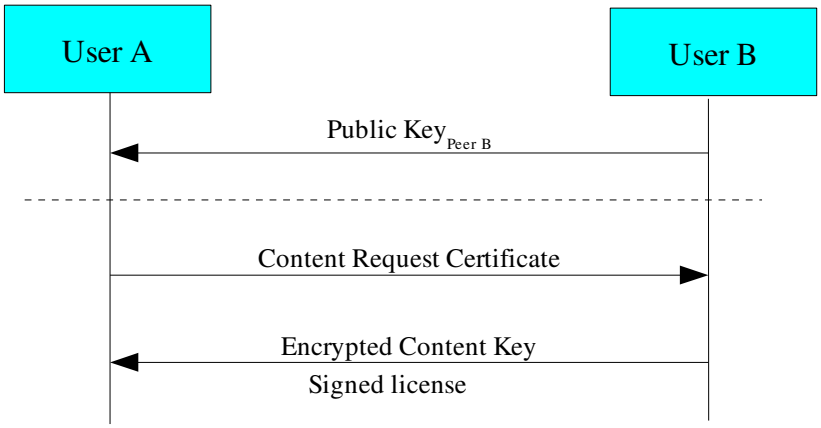


Figure 7: License transfer sequence diagram

In a first step the public key of the user B need to be transferred to user A. This step can be skipped, if the key of user B can be verified by a certificate hierarchy based on a trusted root certificate. After that, User A transmits the Content Request Certificate to User B. The Content Request Certificate contains a public encryption from User A and a content identifier to request a specific content from User B. The certificate is signed by User A using an AIK.

User B responds with the Encrypted Content key, that is encrypted with the public key of user A. User A can decrypt the key and store the key in the key storage of the DRM core. Then user B transmits the license, which is signed using his private key. User A verifies the signature and stores the license in the license store. If the transmission were successful, user B removes the license and the content key from his sealed

storage.

5.4 Domain Management

The DRM-Core supports the usage of domains, which allow users to share the content with other platforms. A content can be consumed on every platform, that belongs to the same domain. The license of the content specifies the maximum number of peers, that are allowed to join the domain. Every content has an own domain, so each content can be assigned to any other peer, when the limit of peers has not exceeded.

A peer can act either as a domain controller or as a domain client. The domain controller manages the domain and controls the number of peers, that joined the domain. The domain client is a member of the domain, who is able to play the content.

The following figure 8 shows the management of the domain clients. If a domain client wants to play a content as a domain member, it contacts the domain controller. The domain controller checks that the number of domain clients is not exceeded. After that, the domain controller registers the requesting client as a domain member and generates a domain license for the client. The domain license is a temporary license, that is issued for the domain client and that is signed by the domain controller. With this license, the domain client has the permission to play the content as a member of the domain.

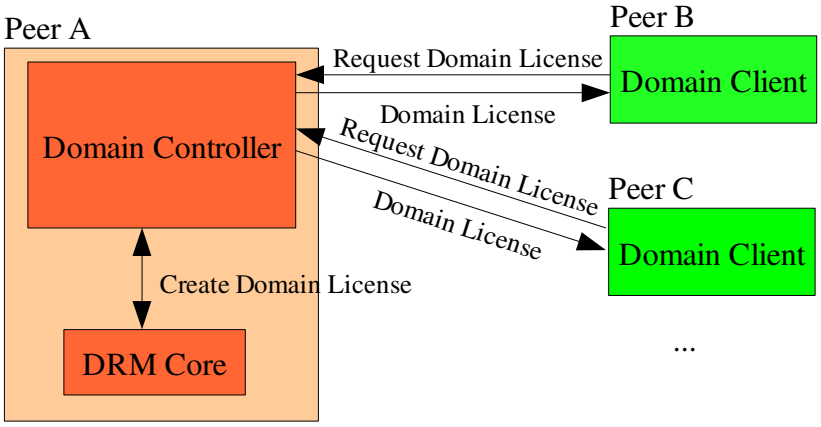


Figure 8: Domain Client Management

6. Technical Specifications

The technical API specifications is described in a separate companion document “System API Specifications”, formatted in a javadoc style. The System API Specifications describes both external and important internal API used in the DRM core.

6.1 External API

The DRM Core exposes 2 main API to the player and management software, namely, the *PlayerInterface* and *ManagementInterface*. These 2 interfaces are used by the network backend component of the DRM core, which handles communication between the core, and the player and management software respectively.

6.2 Internal API

The System API Specifications also specifies important internal API and classes. This illustrates the internal structure and organization of the DRM Core.

7. Requirements from other Partners

The secure application is generally a media player that uses the DRM Core-API to render protected content. The application needs to be secure, because it is allowed to decrypt the content. To maintain the security of the system, the player application should run in a separate compartment, whose integrity and authenticity were checked before its execution.

Furthermore the DRM system expects the presence of an underlying trusted system and requires the following services from it:

- **Secure Environment.** The DRM Core and the media player application may only execute when a secured environment is present. Thus, the underlying system must provide:
    - Memory isolation and protection of processes running in the secure environment.
    - Secure audio and video output paths to certified (signed) hardware drivers and/or hardware. No unauthorized application or service should be able to read from this output path. Optionally cryptographic protection between the driver and the hardware can also be applied when supported by the hardware.
    - A means to measure the integrity of the DRM system and associated applications. This implies the existence of a method for measuring applications before they are loaded and executed. (this is implicitly enforced by the installation policy definition)
- **Cryptographic services.** The DRM Core requires several cryptographic services which have to be provided by the underlying system:
    - A Trusted Software Stack (TSS), supporting AIK generation and sealing. AIKs are required for authentication/remote attestation purposes, while sealing is used to lock cryptographic keys to specific system configurations. The core can thus ensure that content keys are only accessible when the systems integrity is ensured. (this is done indirectly by TPA)
    - Sealed Storage. The DRM Core will use sealed storage for its license and key databases. (this is implicitly mounted by the domain builder)
    - A system-wide database of certificates of root certification authorities, along with services to verify certificates.
- **Central policy management.** Operation of the DRM Core and the media player application will be subject to an operation policy. This policy management would define policy governing communication and management functions of domains in the OpenTC framework.

8. Glossary of Abbreviations

Abbreviation	*Explanation*
API	Application programming interface
DI	Digital Item
DII	Digital Item Identifier
DRM	Digital Rights Management
DVB-CPCM	Digital Video Broadcast – Copy Protection and Content Management
dsig	Digital signature
DTD	Document Type Definition
GUI	Graphical User Interface
I/O	Input / Output
IPMP	Intellectual Property Management and Protection
MPEG	Motion Pictures Experts Group
OMA	Open Mobile Alliance
OpenTC	Open Trusted Computing
OS	Operating System
RDD	Rights Data Dictionary
REL	Rights Expression Language
TPM	Trusted Platform Module
TSS	Trusted Software Stack
UC	Use Case
XML	Extensible Markup Language

9. References

[1] MPEG: MPEG-21 Multimedia Framework Part 1: Vision, Technologies and Strategy. Reference: ISO/IEC TR 21000-1:2004. From ISO/IEC JTC 1.29.17.11.

[2] MPEG: MPEG-21 Multimedia Framework Part 3: Digital Item Identification. Reference: ISO/IEC TR 21000-3:2003. From ISO/IEC JTC 1.29.17.03.

[3] MPEG: MPEG-21 Multimedia Framework Part 4: Intellectual Property Management and Protection Components. Reference: ISO/IEC TR 21000-4. From ISO/IEC JTC 1.29.17.04.

[4] MPEG: MPEG-21 Multimedia Framework Part 5: Rights Expression Language. Reference: ISO/IEC FDIS 21000-5:2004. From ISO/IEC JTC 1/SC 29/WG 11.

[5] MPEG: MPEG-21 Multimedia Framework Part 6: Rights Data Dictionary. Reference: ISO/IEC TR 21000-6:2004. From ISO/IEC JTC 1.29.17.06.

[6] Open Mobile Alliance (2005): DRM Specification Candidate Version 2.0.
http://www.openmobilealliance.org/release_program/drm_v2_0.html

[7] XML-RPC Specification
http://www.xmlrpc.com/spec (Oct 2007)

WP06a Final DRM system specification

Companion document: System API Specification

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Internal document

Deliverable reference number	IST-027635/D06a.3/FINAL | 1.00
Deliverable title	D6a.3 Final System Specification: System API Specification
WP contributing to the deliverable	WP6
Due date	Oct 2007
Actual submission date	Oct 2007

Responsible Organisation	LDV,Lehrstuhl für Datenverarbeitung, TUM
Authors	Chun Hui Suen, Florian Schreiner
Abstract	
Keywords	DRM, fair, interoperable, MPEG-21

Dissemination level	Public
Revision	FINAL | 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table Of Content

Interface ManagementInterface

< Methods >

public interface **ManagementInterface**

This class provides a second external interface for administrative functions to the DRM Core, such as license management and controlling attestation.

Author:

chunhui

Methods

createDomainLicense

```
public License createDomainLicense(ManagerState managerState,
                                   PeerIdentity partner,
                                   Item item)
```

Request for a domain license from this core. This is usually requested from a domain member to the the target DRM core to issue a license for the client.

Parameters:

managerState - state returned during initialization
item - Digital item in question.
partner - target(principal) of new license

Returns:

A transferable license.

---

createTransferLicense

```
public License createTransferLicense(ManagerState managerState,
                                     PeerIdentity partner,
                                     Item item,
                                     boolean deleteSource)
```

Initiate a license transfer to an external peer.

Parameters:

managerState - state returned during initialization
item - Digital item in question.
partner - target(principal) of new license
deleteSource - This simultaneously removes the content key and license from the local DRM core

Returns:

A transferable license.

deleteLicense

```
public License deleteLicense(ManagerState managerState,
                             License itemLicense)
```

Generate a license indicating that the particular license of an item has been removed from the local DRM core. It returns a deauthorization notice, which is a REL prove signed by the core, that the license has been removed.

Parameters:

managerState - state returned during initialization
itemLicense - license to deauthorize

Returns:

deauthorization notice license

---

getAllLicenses

```
public java.util.List getAllLicenses()
```

Dump all licenses. (Used as a management feature to view all licenses stored in the DRM Core.

Returns:

All XML licenses in the DRM Core.

---

getAttestationKey

```
public Key getAttestationKey(ManagerState managerState,
                             Rel relType,
                             java.lang.String serverURL)
```

Obtain an attestation key to be used for downloading content.

Parameters:

managerState - state returned during initialization
playerID - same ID as given by the Core during @method playerInit
relType - REL language used.
serverURL - URL of the server to obtain license.

Returns:

an attestation key to be used for downloading content

---

getLicense

```
public java.util.List getLicense(ManagerState managerState,
                                  Item item)
```

Obtain license for a particular item.

Parameters:

managerState - state returned during initialization
item - Digital item in question.

Returns:

All XML license related to the specified digital item.

---

getLocalIdentity

```
public PeerIdentity getLocalIdentity(ManagerState managerState)
```

Get identity of local DRM core

Parameters:

managerState - state returned during initialization

Returns:

peer identity

---

insertLicense

```
public void insertLicense(ManagerState managerState,
                          License license,
                          Item item)
```

Insert license into database.

Parameters:

license - license to be inserted.
item - The digital item associated with the license.

---

managerInit

```
public ManagerState managerInit(java.lang.String userID)
```

Initialize a connection to the management interface

Returns:

ManagerState object

---

signGeneratedLicense

```
public License signGeneratedLicense(ManagerState managerState,
                                    License unsignedLicense,
                                    Item item,
                                    CoreAssignedRandomKey key)
```

Request the DRM core to sign an application generated license (in the case of application generated content).

Parameters:

managerState - state returned during initialization
unsignedLicense - application generated license
item - item reference
key - an empty CoreAssignedRandomKey Object. The actual key will be generated by the core and stored

Returns:

signature for license

Interface PlayerInterface

< Methods >

---

public interface **PlayerInterface**

This class provides the main external interface to the OpenTC player.

Methods

getDecryptionKey

```
public Key getDecryptionKey(PlayerState playerState,
                            Item item,
                            java.lang.String operation)
```

Main method to request for a decryption key for media playback.

Parameters:

playerState - same state as given by the Core during @method playerInit
item - Digital item to be played
operation - Operation requested on item

Returns:

Content decryption key if successful

---

getSupportedREL

```
public java.util.List getSupportedREL()
```

Get a list of supported REL languages on this DRM core

Returns:

List of supported REL languages

---

playerInit

```
public PlayerState playerInit(java.lang.String playerCapabilities,
                              java.lang.String userID)
```

Initialization method called by the player

Parameters:

playerCapabilities - XML capabilities description of the player

Returns:

a player state referencing this session with the player

Interface DomainClientInterface

< Methods >

---

public interface **DomainClientInterface**

Interface for domain clients.

Author:

chunhui

Methods

attachDomainController

```
public void attachDomainController(java.lang.String url)
```

Attach a new domain controller

Parameters:

url - URL of the new domain controller

---

dettachDomainController

`public void` **`dettachDomainController`**`(java.lang.String url)`

Dettach a new domain controller

Parameters:

url - URL of the new domain controller

---

getAttachedDomainURLs

`public java.util.LinkedList` **`getAttachedDomainURLs`**`()`

Get a list of all domains associated with this DRM Core.

Returns:

List of name of all domains.

---

requestDomainLicense

`public java.util.List` **`requestDomainLicense`**`(Item item)`

request a license from the domain controller.

Parameters:

item - Digital item in question.

Returns:

All XML licenses related to the specified digital item.

---

returnDomainLicense

`public void` **`returnDomainLicense`**`(java.lang.String url, License lic)`

Send a deauthorization license of a license deleted from the local DRM core, back to the domain controller. param url Domain controller param lic deauthorization license

---

Interface DbManager

< Methods >

---

public interface **DbManager**

Interface to the database engine.

Author:

chunhui

Methods

deleteLicense

`public void deleteLicense(Item item)`

Delete all licenses related to a digital item.

Parameters:

item - Digital item

---

getDecryptionKey

`public Key getDecryptionKey(Item item)`

Get the content decryption key for a digital item.

Parameters:

item - Digital item

Returns:

content decryption key

---

getItemState

`public ItemState getItemState(Item item)`

Get the item state of a digital item.

Parameters:

item - Digital item

Returns:

Item state

---

getLicense

`public java.util.List getLicense(Item item)`

Get all licenses related to a digital item.

Parameters:

item - Digital item.

Returns:

All related licenses.

setLicense

```
public void setLicense(Item item,
                       License license)
```

Store a license and item state linked to a digital item.

Parameters:

item - Digital item
license - license
state - item state

---

setState

```
public void setState(Item item,
                     ItemState state)
```

Store the item state linked to a digital item, without changing its license.

Parameters:

item - Digital item
state - item state

Interface LicenseManager

< Methods >

---

public interface **LicenseManager**

Interface to the license manager. This is the manager which

Author:

chunhui

Methods

getSupportSourceLanguage

```
public Rel getSupportSourceLanguage()
```

Get supported REL language which can be parsed.

Returns:

REL language

---

getSupportTargetLanguages

```
public java.util.List getSupportTargetLanguages()
```

Get supported REL languages which can be generated during a license translation.

Returns:

List of REL languages

---

interpretLicense

```
public java.lang.String interpretLicense(java.util.List licenseGroup,
                                         Item item,
                                         java.lang.String operation)
                          throws OtcException
```

Authorize an operation performed on a digital item, based on the group of licenses

Parameters:

licenseGroup - List of licenses
item - digital item
operation - operation

Returns:

REL restriction for the operation

Throws:

de.tum.ldv.opentc.model.ex.OtcException -

---

translateLicense

```
public License translateLicense(License sourceLicense,
                                Rel targetLicenseLang,
                                java.lang.String targetRestrictions)
```

Translates a license from one REL language to another.

Parameters:

sourceLicense - input license
targetLicenseLang - target REL language
targetRestrictions - translation restrictions.

Returns:

translated license

Class LicenseManagerImpl

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.license.LicenseManagerImpl
```

All Implemented Interfaces:
LicenseManager

< Constructors > < Methods >

---

public class **LicenseManagerImpl**
extends java.lang.Object
implements LicenseManager

Constructors

LicenseManagerImpl

```
public  LicenseManagerImpl()
```

Methods

getSupportSourceLanguage

```
public Rel getSupportSourceLanguage()
```

---

getSupportTargetLanguages

```
public java.util.List getSupportTargetLanguages()
```

---

interpretLicense

```
public java.lang.String interpretLicense(java.util.List licenseGroup,
                                         Item item,
                                         java.lang.String operation)
```

---

translateLicense

```
public License translateLicense(License sourceLicense,
                                Rel targetLicenseLang,
                                java.lang.String targetRestrictions)
```

Interface InterpreterInterface

< Methods >

---

public interface **InterpreterInterface**

Interface to REL interpreter

Author:

chunhui

Methods

getSupportedInterpretedLanguages

```
public java.util.List getSupportedInterpretedLanguages()
```

Return the list of supported languages this interpreter can parse.

Returns:

List of REL languages

---

getSupportedSourceLanguage

```
public Rel getSupportedSourceLanguage()
```

Return the supported language this interpreter can parse.

Returns:

List of REL languages

---

interpretLicense

```
public java.lang.String interpretLicense(java.util.List licenseGroup,
                                          Item item,
                                          ItemState state,
                                          SystemState system,
                                          java.lang.String operation)
```

Authorize an operation on a digital item, given a group of licenses, iten state and system state.

Parameters:

licenseGroup - List of licsenses
item - Digital item
state - Item state
system - System state
operation - Operation on digital item

Returns:

REL restriction of operation

Class MPEG21Interpreter

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.license.interpreter.MPEG21Interpreter
```

All Implemented Interfaces:
InterpreterInterface

< Constructors > < Methods >

---

public class **MPEG21Interpreter**
extends java.lang.Object
implements InterpreterInterface

Constructors

MPEG21Interpreter

```
public  MPEG21Interpreter()
```

Methods

getSupportedInterpretedLanguages

```
public java.util.List getSupportedInterpretedLanguages()
```

---

getSupportedLanguage

```
public Rel getSupportedLanguage()
```

---

getSupportedSourceLanguage

```
public Rel getSupportedSourceLanguage()
```

---

interpretLicense

```
public java.lang.String interpretLicense(java.util.List licenseGroup,
                                         Item item,
                                         ItemState state,
                                         SystemState system,
                                         java.lang.String operation)
```

Interface StateManager

< Methods >

---

public interface **StateManager**

State manager

Author:

chunhui

Methods

checkCurrentUse

```
public int checkCurrentUse(Item item,
                           java.lang.String operation)
```

Check current operation count

Parameters:

item - digital item
operation - operation on item

Returns:

current count of operation

---

checkLimit

```
public int checkLimit(Item item,
                      java.lang.String operation)
```

Check limits of counter

Parameters:

item - digital item
operation - operation on item

Returns:

maximum count of operation

---

countUse

```
public int countUse(Item item,
                    java.lang.String operation)
```

Increment counter for operation on digital tiem

Parameters:

item - digital item
operation - operation on item

Returns:

current count

Package de.tum.ldv.opentc.manager.core

Class Summary

CoreManager

The CoreManager coordinates the requests from the application layer and forwards them to the appropriate components.

de.tum.ldv.opentc.manager.core

Class CoreManager

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.core.CoreManager
```

All Implemented Interfaces:
ManagementInterface, PlayerInterface

< Constructors > < Methods >

public class **CoreManager**
extends java.lang.Object
implements ManagementInterface, PlayerInterface

The CoreManager coordinates the requests from the application layer and forwards them to the appropriate components. It implements the ManagerInterface and PlayerInterface

Author:
chunhui

Constructors

CoreManager

```
public  CoreManager(java.lang.String localIdentity)
```

Methods

addDomainController

```
public void addDomainController(ManagerState managerState,
                                java.lang.String url)
```

createDomainLicense

```
public License createDomainLicense(ManagerState managerState,
                                   PeerIdentity partner,
                                   Item item)
```

createTransferLicense

```
public License createTransferLicense(ManagerState managerState,
                                      PeerIdentity partner,
                                      Item item,
                                      boolean deleteSource)
```

---

deleteLicense

```
public License deleteLicense(ManagerState managerState,
                             License itemLicense)
```

---

getAllLicenses

```
public java.util.List getAllLicenses()
```

---

getAttestationKey

```
public Key getAttestationKey(ManagerState mgrState,
                             Rel relType,
                             java.lang.String serverURL)
```

---

getDecryptionKey

```
public Key getDecryptionKey(PlayerState playerState,
                            Item item,
                            java.lang.String operation)
```

---

getLicense

```
public java.util.List getLicense(ManagerState mgrState,
                                 Item item)
```

---

getLocalIdentity

```
public PeerIdentity getLocalIdentity(ManagerState managerState)
```

---

getSupportedREL

```
public java.util.List getSupportedREL()
```

---

insertLicense

```
public void insertLicense(ManagerState mgrState,
                          License lic,
                          Item item)
```

---

managerInit

```
public ManagerState managerInit(java.lang.String userID)
```

---

playerInit

```
public PlayerState playerInit(java.lang.String playerCapabilities,
                              java.lang.String userID)
```

---

removeDomainController

```
public void removeDomainController(ManagerState managerState,
                                   java.lang.String url)
```

---

signGeneratedLicense

```
public License signGeneratedLicense(ManagerState managerState,
                                    License unsignedLicense,
                                    Item item,
                                    CoreAssignedRandomKey key)
```

Package de.tum.ldv.opentc.manager.user

Class Summary

User

UserManager

---

de.tum.ldv.opentc.manager.user

Class User

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.user.User
```

< Constructors > < Methods >

---

public class **User**
extends java.lang.Object

Constructors

User

```
public  User(java.lang.String UserID)
```

Methods

getUserName

```
public java.lang.String getUserName()
```

---

de.tum.ldv.opentc.manager.user

Class UserManager

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.user.UserManager
```

< Constructors > < Methods >

public class **UserManager**
extends java.lang.Object

Constructors

UserManager

```
public  UserManager()
```

Methods

addUser

```
public User addUser(java.lang.String userID)
```

---

listUsers

```
public java.util.List listUsers()
```

---

removeUser

```
public void removeUser(User userObj)
```

Package de.tum.ldv.opentc.manager.util

Interface Summary

IPMPTool

Interface for IPMP tool

Class Summary

Ipmp

Class to manage all IPMP tools in the system

---

de.tum.ldv.opentc.manager.util

Interface IPMPTool

< Methods >

---

public interface **IPMPTool**

Interface for IPMP tool

Author:

chunhui

Methods

getDescription

`public java.lang.String` **`getDescription`**`()`

Get tool description

Returns:

tool description

---

getName

`public java.lang.String` **`getName`**`()`

Get String name of this tool

Returns:

Name of tool

---

process

```
public java.lang.Object process(Item item,
                                java.lang.Object obj)
```

"Action" method of this tool to process a target object

Parameters:

item - Digital item to be processed
obj - processing parameter object

Returns:

result object

---

de.tum.ldv.opentc.manager.util

Class Ipmp

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.manager.util.Ipmp
```

< Constructors > < Methods >

---

public class **Ipmp**
extends java.lang.Object

Class to manage all IPMP tools in the system

Author:

chunhui

Constructors

Ipmp

```
public  Ipmp()
```

Methods

getTools

`public static java.util.List` **`getTools`**`()`

Get a particular tool based on a string search

Parameters:

toolDescription -

Returns:

list of IPMP objects registered as IPMP tools

Package de.tum.ldv.opentc.model

Interface Summary

ItemIdentifier

Class Summary

CoreAssignedRandomKey

Int32Identifier

Item

Reference to a digital Item.

ItemState

Item state object

Key

Generic Key object

License

License object

Mpeg21Rel

PeerIdentity

Rel

REL language type

RestrictedKey

Content key with an associated usage restriction

---

de.tum.ldv.opentc.model

Class CoreAssignedRandomKey

```
java.lang.Object
    |
    +--Key
        |
        +--de.tum.ldv.opentc.model.CoreAssignedRandomKey
```

< Constructors >

---

public class **CoreAssignedRandomKey**
extends Key

Constructors

CoreAssignedRandomKey

```
public  CoreAssignedRandomKey(int length,
                              int kType)
```

---

de.tum.ldv.opentc.model

Class Int32Identifier

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.Int32Identifier
```

All Implemented Interfaces:
ItemIdentifier

< Constructors > < Methods >

---

public class **Int32Identifier**
extends java.lang.Object
implements ItemIdentifier

Constructors

Int32Identifier

```
public  Int32Identifier(int id)
```

Methods

equal

```
public boolean equal(ItemIdentifier item)
```

---

getInt32ID

```
public int getInt32ID()
```

---

toString

```
public java.lang.String toString()
```

Overrides:

toString in class java.lang.Object

---

de.tum.ldv.opentc.model

Class Item

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.Item
```

< Constructors > < Methods >

---

public class **Item**
extends java.lang.Object

Reference to a digital Item.

Author:

chunhui

Constructors

Item

```
public  Item(ItemIdentifier id)
```

Constructor

Parameters:

id - ID of this item.

Methods

getID

`public ItemIdentifier getID()`

Read the ID of this item.

Returns:

ID of item

---

de.tum.ldv.opentc.model

Interface ItemIdentifier

< Methods >

---

public interface **ItemIdentifier**

Methods

equal

`public boolean equal(ItemIdentifier item)`

---

toString

`public java.lang.String toString()`

Overrides:

toString in class java.lang.Object

---

de.tum.ldv.opentc.model

Class ItemState

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.ItemState
```

< Constructors > < Methods >

---

public class **ItemState**
extends java.lang.Object

Item state object

Author:

chunhui

Constructors

ItemState

`public` **`ItemState`**`(ItemIdentifier item_ID)`

Constructor

Parameters:

item_ID -

Methods

getItemID

`public ItemIdentifier` **`getItemID`**`()`

Get digital item ID

Returns:

Item ID

---

getStateProperty

`public java.lang.Object` **`getStateProperty`**`(java.lang.String key)`

Get Item state property

Parameters:

key -

Returns:

stored value

---

setStateProperty

```
public void setStateProperty(java.lang.String key,
                             java.lang.Object data)
```

Set Item state property

Parameters:

key -
data -

---

de.tum.ldv.opentc.model

Class Key

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.Key
```

Direct Known Subclasses:
CoreAssignedRandomKey, RestrictedKey

< Fields > < Constructors > < Methods >

---

public class **Key**
extends java.lang.Object

Generic Key object

Author:
chunhui

Fields

AES_TYPE

```
public static final int AES_TYPE
```

Constructors

Key

```
public  Key(int length,
            int kType,
            byte[] key)
```

Constructor

Parameters:

length - number of bytes
kType - key type
key - byte array of key data

Key

```
public  Key(int length,
            int kType,
            java.lang.String keyInHex)
```

Constructor

Parameters:

length - number of bytes
kType - key type
key - String of key data in hexadecimal string notation

Methods

getKType

```
public int getKType()
```

Get key type

Returns:

key type

---

getKey

```
public byte[] getKey()
```

Get key data

Returns:

byte array of key

---

getLength

```
public int getLength()
```

get length of key

Returns:

length of key

---

de.tum.ldv.opentc.model

Class License

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.License
```

< Constructors > < Methods >

---

public class **License**
extends java.lang.Object

License object

Author:

chunhui

Constructors

License

```
public  License(Rel rel,
                java.lang.String text)
```

Constructor

Parameters:

rel - REL language type
text - String of license

Methods

getRel

`public Rel getRel()`

Get REL language type

Returns:

REL language type

---

getText

`public java.lang.String getText()`

Get string text of license

Returns:

license string

---

de.tum.ldv.opentc.model

Class Mpeg21Rel

```
java.lang.Object
    |
    +--Rel
        |
        +--de.tum.ldv.opentc.model.Mpeg21Rel
```

< Constructors > < Methods >

---

public class **Mpeg21Rel**
extends Rel

Constructors

Mpeg21Rel

`public Mpeg21Rel()`

Methods

getMpeg21Rel

```
public static Rel getMpeg21Rel()
```

---

de.tum.ldv.opentc.model

Class PeerIdentity

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.PeerIdentity
```

< Constructors >

---

public class **PeerIdentity**
extends java.lang.Object

Constructors

PeerIdentity

```
public  PeerIdentity()
```

---

de.tum.ldv.opentc.model

Class Rel

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.Rel
```

Direct Known Subclasses:
Mpeg21Rel

< Constructors > < Methods >

---

public abstract class **Rel**
extends java.lang.Object

REL language type

Author:
chunhui

Constructors

Rel

```
public  Rel()
```

Methods

getName

```
public java.lang.String getName()
```

Get name of this REL type

Returns:

REL name

---

de.tum.ldv.opentc.model

Class RestrictedKey

```
java.lang.Object
    |
    +--Key
         |
         +--de.tum.ldv.opentc.model.RestrictedKey
```

< Constructors > < Methods >

---

public class **RestrictedKey**
extends Key

Content key with an associated usage restriction

Author:

chunhui

Constructors

RestrictedKey

```
public  RestrictedKey(Key k,
                      java.lang.String usageRestriction)
```

Construction

Parameters:

k - normal key object
usageRestriction - Usage restriction in REL description

Methods

getUsageRestriction

```
public java.lang.String getUsageRestriction()
```

get usage restriction

Returns:

Usage restriction String

Package de.tum.ldv.opentc.model.ex

Class Summary

KeyNotFound

LicenseAuthorizationFailed

NoLicenseInCore

OtcException

---

de.tum.ldv.opentc.model.ex

Class KeyNotFound

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--OtcException
                |
                +--de.tum.ldv.opentc.model.ex.KeyNotFound
```

All Implemented Interfaces:
java.io.Serializable

< Constructors >

---

public class **KeyNotFound**
extends OtcException

Constructors

KeyNotFound

```
public  KeyNotFound()
```

---

de.tum.ldv.opentc.model.ex

Class LicenseAuthorizationFailed

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--OtcException
                |
                +--de.tum.ldv.opentc.model.ex.LicenseAuthorizationFailed
```

All Implemented Interfaces:
java.io.Serializable

< Constructors >

---

public class **LicenseAuthorizationFailed**
extends OtcException

Constructors

LicenseAuthorizationFailed

```
public  LicenseAuthorizationFailed(java.lang.String reason)
```

---

de.tum.ldv.opentc.model.ex

Class NoLicenseInCore

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--OtcException
                |
                +--de.tum.ldv.opentc.model.ex.NoLicenseInCore
```

All Implemented Interfaces:
java.io.Serializable

< Constructors >

---

public class **NoLicenseInCore**
extends OtcException

Constructors

NoLicenseInCore

```
public  NoLicenseInCore()
```

---

de.tum.ldv.opentc.model.ex

Class OtcException

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--de.tum.ldv.opentc.model.ex.OtcException
```

All Implemented Interfaces:
java.io.Serializable

Direct Known Subclasses:
KeyNotFound, LicenseAuthorizationFailed, NoLicenseInCore

< Constructors >

---

public class **OtcException**
extends java.lang.Exception

Constructors

OtcException

```
public  OtcException()
```

Package de.tum.ldv.opentc.model.state

Class Summary

ManagerState

PlayerState

State associated with a connected media player

SystemState

Sytem states in DRM core.

---

de.tum.ldv.opentc.model.state

Class ManagerState

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.state.ManagerState
```

< Constructors >

---

public class **ManagerState**
extends java.lang.Object

Constructors

ManagerState

```
public  ManagerState()
```

---

de.tum.ldv.opentc.model.state

Class PlayerState

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.state.PlayerState
```

< Constructors > < Methods >

---

public class **PlayerState**
extends java.lang.Object

State associated with a connected media player

Author:

chunhui

Constructors

PlayerState

```
public  PlayerState(int playerId,
                    java.lang.String playerCapabilities)
```

Constructor

Parameters:

playerId - ID given by initialization.
playerCapabilities - player capabilities

PlayerState

```
public  PlayerState(java.lang.String playerCapabilities)
```

Constructor. playerID is automatically assigned in increasing sequence.

Parameters:

playerCapabilities - player capabilities

Methods

getPlayerCapabilities

```
public java.lang.String getPlayerCapabilities()
```

Get player capabilities.

Returns:

player capabilities

---

getPlayerId

```
public int getPlayerId()
```

get player ID.

Returns:

player ID

getProperty

```
public java.lang.String getProperty(java.lang.String key)
```

Get property of player state

Parameters:

key - key of property

Returns:

data of stored property

setProperty

```
public void setProperty(java.lang.String key,
                        java.lang.String data)
```

Set property into player state.

Parameters:

key - key of property
data - data of property

---

de.tum.ldv.opentc.model.state

Class SystemState

```
java.lang.Object
    |
    +--de.tum.ldv.opentc.model.state.SystemState
```

< Constructors > < Methods >

public class **SystemState**
extends java.lang.Object

Sytem states in DRM core.

Author:

chunhui

Constructors

SystemState

```
public  SystemState()
```

Methods

getState

`public static SystemState getState()`

get system state.

Returns:

system state

---

getSystemDate

`public java.util.Date getSystemDate()`

Get system date.

Returns:

system date

INDEX

I

K

L

M

N

O

P

R

S

T

U

WP06b.2 MEITC Specification and Test Plan

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Deliverable

Deliverable reference number	IST-027635/D6b.2/ Final / 1.00
Deliverable title	WP06b.2 MEITC Specification and Test Plan
WP contributing to the deliverable	WP 6
Due date	Oct 07
Actual submission date	28 Oct 07

Responsible Organisation	TUBITAK
Authors	Görkem Çetin, Kadir İmamoğlu, Volkan Erol
Abstract	This internal deliverable is the specification and test plan for MEITC system
Keywords	

Dissemination level	Public
Revision	

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

List of figures

1 Introduction

1.1 Purpose

The purpose of this document is to describe the Software Requirement Specifications (SRS), the use cases and test plan of the Message Exchange Infrastructure for Trusted Computing (MEITC) system which is a sub-workpackage of the Open Trusted Computing (Open TC) project to be developed by TUBITAK-UEKAE. Open TC is a European Union Sixth Framework Programme Project which was started after the FP6 IST4 call.

1.2 Scope

The system developed will be a fully secure message exchange infrastructure for Linux Operating System by using the Trusted Platform Module (TPM) and the Trusted Software Stack (TSS), built over a virtualization layer. This infrastructure will ensure confidentiality, authentication, non-repudiation and data integrity on the installed base. In this document, the functionality and system requirements specifications of the MEITC's five base components also will be defined. These five base components are the following.

- MEITC Database Server
- MEITC Mail Server
- MEITC Web Server
- MEITC Trusted Log Server
- MEITC Certificate Service Provider

MEITC Database Server: A database server will host users' mailboxes. All e-mail headers, user information and quota information will be kept in this database.

MEITC Mail Server: This component will handle all the e-mail traffic and it will use the Trusted Log and the Certificate Service Provider (CSP) to implement the security services for the messages, namely, integrity checking and non-repudiation.

MEITC Web Server: This component will be the front-end for users and the system administrators. Users and system administrators of MEITC will connect to this web server via their web based browsers to compose or read e-mail messages.

MEITC Trusted Log Server: This component guarantees the integrity checking of e-mails and also the non-repudiation: it holds a record for each e-mail that includes data about the message (i.e. the sender and the recipient addresses, etc.), the digest calculated over the message and optionally the details of the remote attestation of the various components.

MEITC Certificate Service Provider: This component will hold users' digital certificates and keys for signing and encrypting e-mails. It can use the TPM as crypto device for asymmetric operations and also other hardware signing devices. Symmetric encryption will be done by using the cryptographic trusted services developed within OPEN TC Work package 5. The user and the CSP keys will be sealed to the state of the CSP in order to be released only if the system integrity is provided.

The server running the Trusted Log Server (LS) and Certificate Service Provider (CSP) is also responsible to provide measurement values of the compartments, and check this value before other servers are up and running. See Figure 1 for more information on the MEITC system implementation.

1.3 Definitions, acronyms, and abbreviations

1.3.1 Definitions

Appendix 1 in the section 5 (Annex 1) contains definitions for words used within this document.

1.3.2 Acronyms

Appendix 2 in the section 6 (Annex 2) contains commonly used acronyms used in this report.

1.4 References

- IEEE Recommended Practice for Software Requirements Specifications IEEE Std 830-1998
- PET (Private Electronic Transaction) Use Case Document
- OPEN TC D02.1 Requirements Definition and Specification IST-027635 / D02.1 / Final | 1.00
- OPEN TC Annex I - “Description of Work”
- Siani Pearson (ed.): Trusted Computing Platforms: TCPA Technology in Context, Prentice Hall PTR2003
- TCG Specification Architecture Overview

1.5 Overview

This document is prepared in accordance with the IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, and extended by including use cases.

In the following sections of this documentation the software requirement specifications of the MEITC system will be explained in a more detailed manner. Second section (Overall Description) of this document gives a general description of the MEITC. In this section, we will explain, respectively, product perspective, product functions, user characteristics, constraints, assumptions and dependencies. In the third section we will define the specific requirements in order to facilitate our work in the design step. The specific requirements that we will talk about are respectively, external interfaces, functions, performance requirements, logical database requirements, design constraints and software system attributes.

2 Overall Description

2.1 Product perspective

In this part, the software specification requirements defined for the MEITC system will be explained in a more detailed manner. We will define also under which constraints the MEITC system will be developed. The interfaces, which will connect the internal and external components with the MEITC, will be explained. Additional requirements

such as memory constraints, additional operations and site adaptation requirements will also be defined here.

This project is not self-contained system. It depends to a larger system which will be developed in OPEN TC. This larger system which is a trusted environment must be ready to support the MEITC system.

The users of the MEITC system will use an unmodified web based browser (Mozilla Firefox, Konqueror, Internet Explorer etc.) to access their accounts. In the client side of the MEITC system users need to have a Trusted Platform Module (TPM) enabled computer for remote attestation purposes. In a MEITC environment, users will be able to read, send and delete e-mail messages securely.

2.1.1 System interfaces

All of the machines in the system will work with TPM support. Access to the web server will be done through a web based browser. In order to guarantee the trustworthiness of the whole system, web browser and web server will communicate on a trusted channel by using HTTP on top of the conventional TLS/SSL protocols.

2.1.2 User Interfaces

Users will connect to MEITC system via their web based browsers. Each page of the messaging system will be prepared using JSP and HTML. There will be two types of users: system administrators and unprivileged user. System administrators will have administrative privileges to arrange users accounts. The other user type will not have administrative privileges, and instead they only have access to their messages in their own message boxes.

User interface of MEITC should have user-centered design, in which tasks are easily followed and executed by the end-user. User interface should also satisfy the general requirements of customer as the software evolves. End-users of MEITC will see a simple webmail that will give them the ability to read, send and delete e-mails (or other actions) they would like to take.

2.1.3 Hardware interfaces

Minimum recommended hardware for MEITC system in server side is as follows:

- At least 2 Ghz Intel or AMD processor
- Infineon TPM 1.2 enabled mainboard
- At least 2 GB of RAM
- 100 GB hard disk. More disk space is needed to handle more inboxes.
- Ethernet card with a bandwidth of at least 100 Mbps
- SVGA colour monitor; minimum 800x600 screen resolution, 1024x768 recommended; minimum of 16 bit colours

On the client side, web based browser requires the following hardware specifications.

- Infineon TPM 1.2 enabled mainboard
- Enough free disk space in hard disk
- 128 MB or more RAM

- SVGA colour monitor; minimum 800x600 screen resolution, 1024x768 recommended; minimum of 16 bit colours

2.1.4 Software interfaces

The following software products will be installed on the main MEITC server components.

MEITC Web Server

- Any Linux distribution with kernel 2.6
- TSS Version: 1.2
- PKI
- Tomcat version >= 5.0
- Apache Web Server version >= 2.0
- Open SSL
- OPEN TC Crypto Utilities
- OPEN TC Measurement Service
- OPEN TC Attestation Service

MEITC Database and Mail Server (also includes CSP and Trusted Log Server for the first prototype)

- Any Linux distribution with kernel 2.6
- MySQL Version >= 5.0
- Postfix Version >= 2.3
- OPEN TC Measurement Service
- OPEN TC Sealing Service

On the client side the main required software product is a web based browser with Hyper Text Markup Language (HTML) version 4.0, Java and Javascript enabled. The following web based browsers are recommended.

- Mozilla Firefox, version 1.0 or higher
- Microsoft Internet Explorer (MSIE), version 5.0 or higher

2.1.5 Communications interfaces

Default communication protocol for data transmission between servers and the client is Transmission Control Protocol / Internet Protocol (TCP/IP). At the upper level Hyper Text Transfer Protocol (HTTP, default port=80) and Secure Socket Layer (SSL, default port = 443) will be used for communication between the web server and client. Current implementation will depend on the output of WP5 remote attestation deliverables.

Access to the web server will be done through a web based browser: in order to guarantee the trustworthiness of the whole system, the browser and the web server will communicate on a trusted channel by using HTTP on top of the conventional TLS/SSL protocols enabled for the mutual platform authentication. General structure of the MEITC system and communication among MEITC components can be visualized in the figure below:

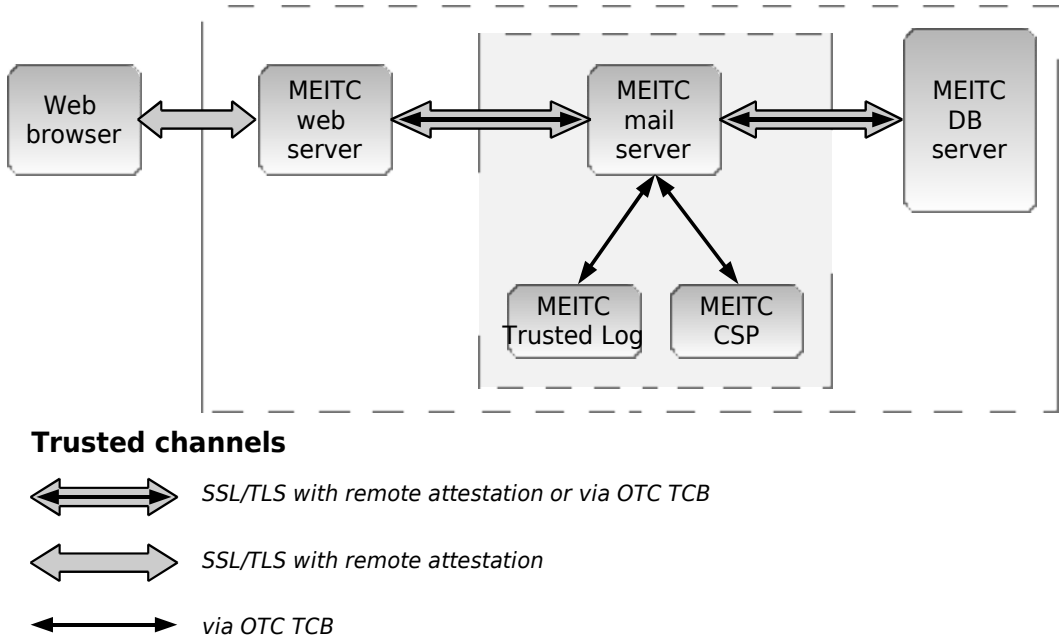


Figure 1:The general structure of the MEITC and communications among MEITC components.

In this model, the CSP and TL will be run on the dom0, and all other servers (DB, MS, WS) will be run on domU, where it's assumed that they are isolated from each other using Xen virtualization layer technology. MEITC prototype will be based on Xen.

2.1.6 Memory constraints

The server system will be able to run 4 operating systems (instances or compartments) as once, so the memory would be sufficient for memory-intensive applications. If the system requires more memory, necessary memory must be added to the system. On the other hand, depending on the configuration, the number of compartments can easily be reduced in order to benefit from low memory and CPU power.

The client computer, which runs the web browser, should have enough physical memory to run this program.

2.1.7 Operations

Following items are a list of operations end users and administrators can do.

- User can login to the message exchange system
- User can read a message
- User can delete a message
- User can send a message
- User can ask for a certificate
- Administrator can define a user account

- Administrator can remove a user account
- Administrator can change / reset a user password
- Administrator can manage CA (i.e generate, revoke or accept certificates and certificate requests)

2.2 Product functions

MEITC message exchange system is an application, which will ensure the “trusted” message exchange between the predefined users in the system by using the trusted computing modules. There are five base components in the system. These components are respectively, MEITC Database Server (DB), MEITC Mail Server (MS), MEITC Web Server (WS), MEITC Trusted Log Server (LS) and MEITC Certificate Service Provider (CSP). These components' functionalities are explained in the section 1.2 (Scope) of this document.

The user names should be generated in CSP and in database, to be authenticated by the system. Communication between client system and web server will be ensured in a “trusted” manner by using attestation services and encryption.

Following operations can be made by end users:

- Read an email
- Send an e-mail
- Delete existing e-mails
- Request a certificate
- View a certificate

A user friendly and interactive interface with a help system will be designed for users in order to do these operations easily.

The operations, that the system administrator can do, are defined respectively as:

- Define a new user account
- Delete an existing user account
- Reset / change users' passwords
- Accept a certificate
- Revoke a certificate

A system administrator interface will also be defined to realize the defined actions.
If necessary, additional roles can be defined in the following parts of the project life cycle.

2.3 User characteristics

We have foreseen that the users, who will use the MEITC system, should have an existing knowledge about how to use Internet or an e-mail client software like Thunderbird, Outlook etc. The potential users of the MEITC system will be organisational users. In many organisations, the computer user profile is more that the expected ones for our application. When designing the system, necessary precautions will be applied to realize a usable and intuitive user interface, which can be utilized by users who don't have specific computer knowledge.

For the system administrator we have foreseen that, they know at least what trusted computing is. Similarly, we have considered that the administrator knows how to define user accounts or change the existing users' definitions.

For these two user profiles (i.e users and administrators), intuitive help menus will be created in order to facilitate the work of the users.

2.4 Assumptions and dependencies

We assume that all the software modules which will be developed in WP5 will be ready during the development phase. This will ensure that MEITC implementation will work in the desired manner. The delay of these modules' implementation can affect the MEITC implementation process.

The following figure shows the architecture of the MEITC system and the relationship between the main components of the MEITC. In the first phase of the project, two components holding CSP, web server, DB server and mail server will be provided.

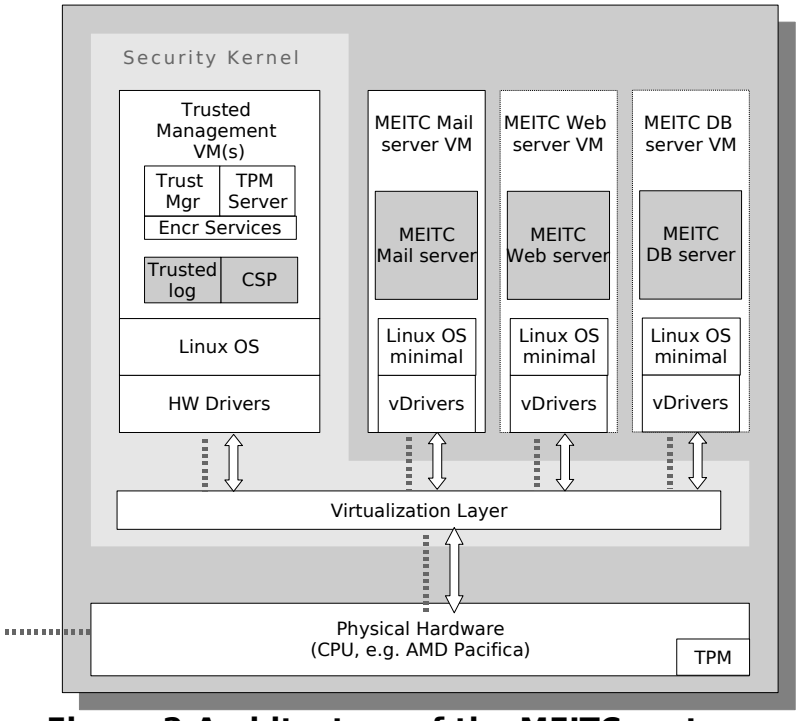


Figure 2:Architecture of the MEITC system

3 Use Cases and Specific requirements

In this part of the SRS document, we will explain the MEITC application use cases specific requirements in a manner that can help the design phase of the project. We will consider the relations of MEITC with other modules developed in the Open TC project. In this part, the inputs and the outputs of the system will also be examined.

3.1 External interfaces

MEITC will work over stacks mentioned in the previous paragraph and will communicate with other components via trusted channels. For the MEITC CSP, openssl crypto service will be used. This CSP will in turn evolve into a certificate authority (CA) which will manage user keys together with certificates. Private keys will be used for signing the e-mails and public keys will be used for encrypting. End users will be able to validate signed e-mails.

3.2 Use Cases

In this section, we will see what the users and the system administrators can do when using the MEITC system.

General assumptions and requirements

The assumptions below describe the security aspects of the environment in which the result will be used or is extended to be used. These assumptions are heavily based on PET Banking Demonstrator Use Cases document.

AR 10: Correct hardware

The underlying hardware is non-malicious and behaves as expected. Optionally, the correct properties of the hardware can be attested by platform certificate

AR 20: Trusted Administrator

Standard services for compartment administration and platform management must be trusted to act in accordance with the wishes of users, since they have to access security-critical information.

AR 30: Physical attacks

Physical attacks against the underlying hardware platform must not happen.

AR 40: Xen based system

The MEITC system will benefit from virtualization, so, a Xen based system is already installed on the system, featuring dom0 (the hypervisor) and domU (the virtual machines).

AR 50: Trusted bootloader

A trusted bootloader, specifically tGRUB, is required to be used to measure the integrity of the system.

AR 60: TPM driver

A TPM driver is required to reach the TPM module on the mainboard.

AR 70: Trusted Software Stack (TSS)

A trusted software stack (v1.x) is required to use the TPM driver.

UC name	**UC 10: MEITC system startup**
Primary actors	System administrator
Stakeholders and interest	All MEITC servers
Assumptions	1. The MEITC components are installed on a trusted computing base 2. A script is used which will allow all servers to identify their status 3. The TPM ownership is already taken by the system administrator
Postconditions	All the servers have remotely attested and are up and running
Main flow	1. System is powered on by the administrator 2. dom0 is checked and booted by tGRUB 3. dom0 starts up the DB, WS and MS servers on different domU compartments 4. dom0 checks running status of the domU compartments
Alternative flow	2.a If integrity checking process fails, the system halts 5.a.1 If one of the compartments do not boot properly, then the system administrator makes sure that the corresponding domU starts 5.a.2 Operation continues with step 3
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 11: Taking TPM Ownership**
Primary actors	System administrator
Stakeholders and interest	All MEITC servers
Assumptions	1. The MEITC components are installed on a trusted computing base 2. The ownership of the system's TPM has not yet been taken.
Postconditions	TPM Ownership has been taken.
Main flow	1. The ownership of the system's TPM is properly taken.
Alternative flow	1. The system administrator takes the ownership of the system. As part of this process, he specifies the owner password.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 12: Remote Attestation**
Primary actors	System administrator
Stakeholders and interest	All MEITC servers
Assumptions	1. The MEITC components are installed on a trusted computing base 2. The TPM ownership is already taken by the system administrator. (UC 11) 3. The known-good values of any compartment is known by remaining compartments.
Description	ClientA is the name of the user requesting trusted channel. ClientB is the name of the user answering this request.
Postconditions	ClientA and ClientB attested each others trusted state.
Main flow	1. ClientA tries to connect to ClientB. 2. In order to allow the target computer to attest the state of the source computer, ClientA computes a quote of its state. 3. ClientB verifies the quote from ClientA. 4. In order to allow the source computer to attest the state of the target computer, the target client computes a quote of its state. 5. ClientA verifies the quote from ClientB. 6. Attestation succeeds.
Alternative flow	3.a.1. Due to modifications of the source computer, the attestation fails 5.a.1. Due to modifications of the target computer, the attestation fails
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 15: Establishing a trusted channel**
Primary actors	Trusted channel application
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC components are installed on a trusted computing base 2. Attestation configuration files have been prepared. 3. The TPM ownership is already taken by the system administrator (UC 11). 4. MEITC System has been started (UC 10).
Description	1. The listener compartment is named “LC” and the requesting compartment is named “RC”.
Postconditions	Secure trusted channel is established.
Main flow	1. Application on LC reads the configuration file. 2. Application on LC starts the listener. 3. Application on LC waits and tries to detect possible incoming connections. 4. If an incoming connection is detected, application on LC opens an SSL connection to RC. 5. LC makes a successful remote attestation with RC using UC 12 and a session is started. 6. LC establishes a secure channel for communication
Alternative flow	5.a.1. If attestation (UC 12) fails go to step3. 5.a.2. If attestation already has been made go to step 6. 5.a.3. If session timeouts go to step 3.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 40: Add a new user**
Primary actors	System administrator
Stakeholders and interest	MEITC web server, MEITC database server, web browser
Assumptions	The system administrator is logged in to the system.
Postconditions	User created.
Main flow	1. System administrator uses the web interface to choose “add user” operation. 2. System administrator enters to the web interface for user details. 3. Web server demands from the database if the entered username is already defined. 4. Database server responds that the user is not already defined. 5. User information is registered to the database server. 6. Database server informs the web server that new user is created. 7. Web server informs administrator that the new user is created.
Alternative flow	4.a.1 If the user is already defined the database server sends an error message to web server. 4.a.2 Web server displays the message on the web browser. 4.a.3 Web browser demands from the system administrator to enter users' informations one more time. 4.a.4 The operations continue with the step 2.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 50: Delete an existing user**
Primary actors	System administrator
Stakeholders and interest	MEITC web server, MEITC database server, web browser
Assumptions	The system administrator is logged in to the system.
Postconditions	User is deleted
Main flow	1. System administrator uses the web interface to choose “delete user” operation. 2. System administrator enters the username to be deleted to the web interface. 3. Web server demands from the database whether this username is already defined. 4. Database server responds the username is already defined. 5. Web server sends a confirmation request to the web browser. 6. Web browser requests confirmation from the system administrator. 7. System administrator confirms the operation. 8. Web browser sends the confirmation to the web server. 9. Web server sends the delete operation to the database server. 10. User is deleted from the database and a message is sent to the web server. 11. Web server passes this message to the web browser. 12. Web browser displays that the selected user is deleted.
Alternative flow	4.a.1 If the user is not already defined, database server sends to the web server an error message. 4.a.2 Web server displays this message on web browser. 4.a.3 Web browser requests from the system administrator to select another username. 4.a.4 The operation continues with the step 2.

	7.a.1 System administrator doesn't confirm the operation. 7.a.2 Operation is interrupted.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 60: User authenticates via MEITC**
Primary actors	User
Stakeholders and interest	All MEITC servers and web browser on client side
Assumptions	1. User is already defined in MEITC system. 2. A web browser is installed in the client side platform.
Postconditions	User is authenticated via MEITC
Main flow	1. User opens web browser in the client side. 2. Web browser establishes a trusted channel with the MEITC Web server as explained in UC 12: a mutual remote attestation is executed. 3. User enters her username and password. 4. Web browser sends username and password to the web server. 5. Web server sends username and password to the mail server. 6. Mail server asks the database server for the username and password 7. Database server returns username and password. 8. Mail server checks username and password with the database server.
Alternative flow	10.a. If the authentication process fails, operation stops.
System requirements	See assumptions in 2.5
Open issues	1. How to implement the mutual remote attestation is still an issue

UC name	**UC 100: Accessing user's inbox**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running 2. The user is authenticated as in UC 60.
Postconditions	User accesses her inbox
Main flow	1. WS connects to MS for accessing the mail inbox data of the user 2. MS gets the inbox data from DB server 3. DB server gives the user data to MS 4. MS sends the data to WS 5. WS forwards the data to the client 6. User chooses next operation
Alternative flow	1.a.1 If MS is not properly functioning, then web server gives an appropriate error message and goes back to login page 2.a.1 If DB is not properly functioning, then web server gives an appropriate error message and goes back to login page
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 110: Sending an e-mail**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running 2. User is authenticated as in UC 60. 3. User can read mails as explained in UC 100.
Postconditions	User sends an e-mail
Main flow	1. User composes the e-mail and selects the signing and encryption options. 2. Client sends the e-mail data to the WS. 3. WS sends the e-mail data to the MS. 4. MS sends the e-mail data to the CSP for signing and encrypting the e-mail. 5. MS generates the signature for the e-mail by using the sender's private key and/or encrypts it by using the public keys of the recipient. 6. MS sends the e-mail transmission information to trusted log server (LS). 7. LS stores a record that contains details of the e-mail and the digest calculated over the message bytes; this secure record is held for non-repudiation purposes. 8. MS sends the e-mail data to the DB. 9. DB stores the signed and/or encrypted e-mail to the sender's and the recipients' mailboxes. 10. MS sends the acknowledge of the operation and the update of the mailbox to the WS 11. WS forwards the acknowledge to the web browser.
Alternative flow	1. In any of the steps above, if the corresponding (affected) server is not functioning properly, then the WS sends a reply showing the error to the user.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 120: Deleting an e-mail**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running, not necessarily with all compartments. 2. The user is authenticated as in UC 60. 3. The user can read her e-mail as explained in UC 100. 4. The user has her inbox open.
Postconditions	The user deletes the selected e-mail
Main flow	1. User selects the appropriate e-mail to be deleted 2. User clicks on the Delete button 3. WS sends this information to MS 4. MS deletes the e-mail and informs DB 5. E-mail is deleted from the DB
Alternative flow	1. In any of the steps above, if the corresponding (affected) server is not functioning properly, then the WS sends a reply showing the error to the user.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 200: Requesting a certificate**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running. 2. User logins to the certificate manager.
Postconditions	User requests a certificate
Main flow	1. User requests for a new certificate. 2. This request is stored in the DB repository. 3. The request process is continued by the administrator in UC 230
Alternative flow	
System requirements	See assumptions in 2.5
Open issues	Normally this should be done by administrator. Usual certificate requesting mechanisms will be investigated.

UC name	**UC 210: Revoking a certificate**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running 2. User has a certificate 3. User logins to the certificate manager.
Postconditions	Certificate is revoked
Main flow	1. User asks for revoking his certificate. 2. This request is stored in the DB repository. 3. The revocation process is continued by the administrator in UC 240
Alternative flow	
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 220: Viewing a certificate**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. MEITC system is running. 2. User logins to the certificate manager. 3. User already has a certificate.
Postconditions	User views the selected certificate.
Main flow	1. User clicks on the menu item in order to view the certificate.
Alternative flow	None
System requirements	See assumptions in 2.5
Open issues	All users should be able to see other's certificates.

UC name	**UC 230: Accepting a certificate request**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. User has requested a certificate as in UC 200. 2. MEITC system is running. 3. Administrator logins to the certificate manager.
Postconditions	Certificate request is granted and certificate is generated.
Main flow	1. Administrator views the certificate request. 2. Administrator accepts the certificate request. 3. Certificate is generated by CSP and stored on the database. 4. When the user logs in again, he'll be informed that certificate is generated.
Alternative flow	3.a. System administrator rejects certificate request.
System requirements	See assumptions in 2.5
Open issues	None

UC name	**UC 240: Accepting a certificate revoke request**
Primary actors	User
Stakeholders and interest	All MEITC servers
Assumptions	1. User has issues a certificate revocation request as in UC 210. 2. MEITC system is running. 3. Administrator logins to the certificate manager.
Postconditions	Certificate revocation request is accepted and certificate is revoked.
Main flow	1. Administrator views the certificate revocation request. 2. Administrator accepts the certificate revocation request. 3. Certificate is revoked generated by CSP and revocation information is stored on the database. 4. Certificate is deleted from the DB.
Alternative flow	3.a. System administrator rejects certificate revocation request.
System requirements	See assumptions in 2.5
Open issues	This process can optionally automatically be issued by the system, immediately after user request, without administrator intervention.

3.3 Performance requirements

In this step of the project, it is not possible to give numerical performance requirements. Normally, the application focuses on the security of messaging system. Therefore the performance of the system is not considered as an important issue for the project yet.

3.4 Design constraints

In this section, the standards that the MEITC have to comply with and the hardware constraints to be obeyed will be examined.

3.4.1 Standards compliance

Generally, the components developed in the OPEN TC project will support TPM 1.2 and

TSS 1.2. In this case the TSS implementation written for Linux will have to comply with the standards prepared by the Trusted Computing Group (TCG). TSS developed in Open TC project will support the version 1.2 of the standards, and so will MEITC. For hardware compliance, MEITC will utilize Infineon TPM 1.2 supported mainboards.

3.5 Software System Attributes

Software system attribute requirements for the MEITC system are as follows.

3.5.1 Reliability

Software developed for MEITC system must run without any critical problems on installed base. All requests in the software system must run correctly and finish requests without any error. When an error occurs in the system, accurate knowledge should be given to the user. The minimal configurations for computers must be provided in the MEITC systems. Reliability shall depend on the reliability of hardware devices. Failure of one will render the other useless for this application. For this project, all systems in server side will be running on OpenSUSE. For the initial prototype, Pardus Linux will be used.

3.5.2 Availability

This system is designed to run 24 hours a day and be readily available to the user. System administrator makes necessary controls for the availability of the MEITC system. Administrator of the system must be sure of running web, e-mail, database and CSP services correctly. In any problems if necessary computers can be restarted by the system administrator and loss or corrupted data can be restored from backups.

3.5.3 Security

To make a secure system, unnecessary ports in the MEITC system should be closed for internal and external access. For the end users connecting to the system via browser necessary ports like HTTP (80), OpenSSL (443) and TSS communication should be open.

3.5.4 Maintainability

User administration and application running in the MEITC system will be via web browser based interface. Necessary OpenSSL certificates which will be used in the system should be installed for each user computers separately for a smooth run.

MEITC system should be backed up during predefined intervals. In this way we can decrease data loss and corruption when system collapses. Database, Trusted Log and CSP server backups must be taken frequently.

3.5.5 Portability

All the applications developed are portable to other Linux distributions since they will be provided with RPM packages together with source code. MEITC is not portable itself to other platforms.

4 Test plan

MEITC test plan will include unit tests, module test, integration test and usability test during the project development phase. “MEITC Detailed Design and Test Document” includes an extended test document which is based on this test plan. All tests except

usability test will be accomplished by TUBITAK staff who whote the application itself.

5 Appendix - 1 : Definitions

Table 1 contains definitions for words used within this document.

Table 1. Definitions

Terminology	***Definition***
Apache	A general purpose web server
Browser	Software used to view hypertext documents
Integrity Measurement	Integrity measurement is the process of obtaining metrics of platform characteristics that affect the integrity (trustworthiness) of a platform; storing those metrics; and putting digests of those metrics in PCRs
Java	A programming language created by Sun Microsystems
MySQL	MySQL is a database management system
Postfix	Postfix is an open source mail transfer agent
Sealing	Sealing provides assurance that a protected messages are only recoverable when the platform is functioning in a very specific known configuration.
Remote Attestation	Remote attestation is a way to prove to a challenger that you're truly running the software on your computer that you say you are.

6 Appendix - 2 : Acronyms

Commonly used acronyms used in this report are given in Table 2.

Table 2. Acronyms

Acronym	***Terminology***	***Definition***
CSP	Certificate Service Provider	CSP is the certificate service provider for MEITC
DB	Database	DB is a collection of information organized in such a way that a computer program can quickly select desired pieces of data
HTML	Hyper Text Markup Language	HTML, a subset of Standard Generalized Mark-Up Language (SGML) for electronic publishing, the specific standard used for the World Wide Web (WWW)
HTTP	Hyper Text Transfer Protocol	HTTP, the actual communications protocol that enables web browsing
LS	Log Server	LS is the machine to which log events are sent by the system
MEITC	Message	MEITC is a secure message exchange environment.

Acronym	***Terminology***	***Definition***
	Exchange Infrastructure for Trusted Computing	
OPEN TC	Open Trusted Computing	OPEN TC consortium is an R&D project focusing on the development of trusted and secure computing systems based on open source software.
Open SSL	Open Secure Socket Layer	OpenSSL is a popular package to add cryptographic security to applications communicating over a network
JSP	Java Server Pages	Java Server Pages (JSP) are normal HTML with Java code pieces embedded in them. A JSP compiler is used to generate a Servlet from the JSP page.
PKI	Public Key Infrastructure	PKI is a secure method for exchanging information. PKI uses a public/private key, to encrypt IDs, documents, or messages. It starts with a certificate authority (CA), which issues digital certificates. Digital certificates or digital IDs authenticate the identity of people and organizations over a public system such as the Internet.
SRS	Software Requirements Specification	SRS is a complete description of the behaviour of the system to be developed.
SVN	Subversion	SVN is a tool which allows development teams to safely coordinate and track software source code changes.
TC	Trusted Computing	Trusted computing is a combination of software and hardware supporting applications to ensure that data cannot be accessed unless the user's system is operating as expected and has not been tampered with.
TCB	Trusted Computing Base	TCB is a part of a platform that is assumed to perform its tasks correctly, even if the platform itself is corrupted.
TCP/IP	Transmission Control Protocol / Internet Protocol	TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network.
TPM	Trusted Platform Module	The TPM is a micro controller that stores keys, passwords and digital certificates.
TSS	Trusted Software Stack	The TSS is a software specification that provides a standard API (Application Programming Interface) for accessing the functions of the TPM.

Acronym	***Terminology***	***Definition***
WS	Web Server	A web server is a computer that stores web documents and makes them available for the rest of the world to see through the world wide web.

WP06b.3 MEITC Detailed Design and Test Document

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Deliverable

Deliverable reference number	IST-027635/D6b.2/ Final / 1.00
Deliverable title	WP06b.3 MEITC Detailed Design and Test Document
WP contributing to the deliverable	WP 6
Due date	Oct 07
Actual submission date	28 Oct 07

Responsible Organisation	TUBITAK
Authors	Görkem Çetin, Kadir İmamoğlu, Volkan Erol
Abstract	This internal deliverable is the detailed design and test plan for MEITC system
Keywords	

Dissemination level	Public
Revision	

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

List of figures

1 Introduction

This detailed design document (DDD) is the high level design of MEITC application (D06b3). Prior to this document, a report containing the MEITC specification has been created and submitted which in turn includes the use case scenarios, requirements and test cases complying IEEE specification requirements document.

Detailed design document introduces how MEITC system will work and various interfaces between different components of MEITC. Since low level use-case diagrams have been mentioned in a previous deliverable (D06b2), these will not be considered again to be included in this document.

Generally speaking, MEITC system should be capable and have an easy to use interface. For each additional level of complexity that is added, there should be a compelling reason. Clients and servers are different components for obvious reasons which has been detailed in D06b2. This framework includes more than one compartment installations with different capabilities plus many frameworks, utilities and drivers.

1.1 Purpose

This document defines design, coding standards and tools. Programmers should obey the standards and use the applications and tools mentioned in Part 2 of this document. During the implementation of the design, documentation for each component is produced.

1.2 Definitions and abbreviations

MEITC: Message Exchange Infrastructure for Trusted Computing

GUI: Graphical user interface

DDD: Detailed design document

MS: Mail server

WS: Web server

LS: Log server

CSP: Certificate service provider

DB: Database server

HTTP: Hypertext transfer protocol

TLS: Transport layer security

SSL: Secure socket layer

2 Design description

This section aims to describe the architecture of the MEITC system. It also provides explanations about the choices made for the implementation of different parts of the system. The section is split into 4 parts which explains the design philosophy, the implementation language, the design overview and the system components which are MEITC Web Server (WS), MEITC Mail Server (MS), MEITC Database (DB) server, MEITC Log Server (LS), MEITC Certificate Service Provider (CSP) and user interfaces and also

other software modules defined for implementation purposes. The descriptions of modules and components are made by the structure of the components and also paying attention to the virtualization concepts.

2.1 Design philosophy

The preliminary architecture of the MEITC system is explained in the report D06b.2 MEITC Specification and Test Plan. The approach explained in this Detailed Design and Test Document is based on these specifications. It is always possible that some minor changes can be made in different components during the implementation phases of the project but the fixed general and detailed concerns of the MEITC system design is explained in the following subsections of this document.

2.2 Implementation Language

In the MEITC system, due to the structure of the already developed open source packages taken for some components, different programming and scripting languages will be used. For the system installation and configuration, bash shell scripts will be used. For some applications that interacts directly with the TPM chip, C programming language will be used. For user interface JSP scripting language will be used. For modules interacting the core components of the system (for example: one of the server components) with the user interface the Java programming language will be used. The components developed will be implemented in a manner that will not cause major problems for portability.

2.3 Design Overview

The system developed will be a fully secure message exchange infrastructure for Linux Operating System by using the Trusted Platform Module (TPM) and the Trusted Software Stack (TSS), built over a virtualization layer. This infrastructure will ensure confidentiality, authentication, non-repudiation and data integrity on the installed base.

MEITC system is composed of 6 components: MEITC WS, MEITC MS, MEITC DB Server, MEITC LS, MEITC CSP and User interface as described in the MEITC Specification and Test Plan document. As the OpenTC infrastructure uses the virtualization concepts, the architecture of the system is designed as follows:

The previous architecture was based upon previous discussions and such, they do not refer to the latest mutual agreement of what MEITC will look like. I'm amending the following items so they reflect the final MEITC framework.

- Dom0 contains no software applications other than Trusted Grub (tGrub), measurement services and helper utilities to start DomU.
- First prototype only includes MEITC WS, MEITC MS and MEITC DB Server on two compartments, and 3rd year prototype will include MEITC CSP and MEITC LS, , where it's assumed that all of these compartments isolated from each other using Xen virtualization layer technology. MEITC prototype will be based on Xen.
- For suitability and performance considerations, there can be two virtual compartments, one holding web services and the other holding database, mail server, log server and CSP where certificates, keypairs, passwords for users are hold and is assumed to be more secure than the first compartment. The first

prototype will be built on this model.

The general model of these compartments is in the following figure, according to the first prototype:

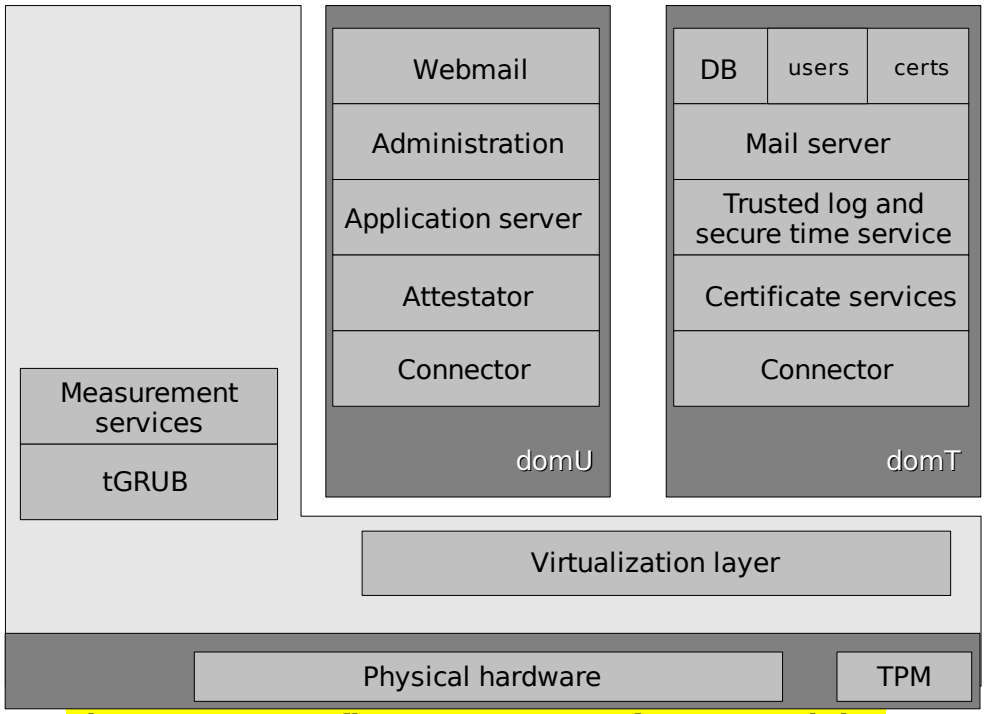


Figure 1: MEITC first prototype software modules

Below is a short explanation about each of the MEITC components:

MEITC Database Server: A database server will host users' mailboxes. User information and quota information will be kept in this database.

MEITC Mail Server: This component will handle all the e-mail traffic and it will use the Trusted Log and the Certificate Service Provider (CSP) to implement the security services for the messages, namely, integrity checking and non-repudiation. E-mails will be stored in a directory structure.

MEITC Web Server: This component will be the front-end for users and the system administrators. Users and system administrators of MEITC will connect to this web server via their web based browsers to compose or read e-mail messages.

MEITC Trusted Log Server: This component guarantees the integrity checking of e-mails and also the non-repudiation: it holds a record for each e-mail that includes data about the message (i.e. the sender and the recipient addresses, etc.), the digest calculated over the message and optionally the details of the remote attestation of the various components. Trusted log server needs a secure time component from WP05a in domU which could be used to count ticks after synchronizing from a trusted time

server.

MEITC Certificate Service Provider: This component will hold users' digital certificates and keys for signing and encrypting e-mails. User and CSP keys will be sealed to the state of the CSP in order to be released only if the system integrity is effective.

Dom0 is responsible to provide measurement values of the specific files in compartments, and check this value before corresponding servers are up and running.

The default communication protocol for data transmission between servers and the client is Transmission Control Protocol / Internet Protocol (TCP/IP). At the upper level Hyper Text Transfer Protocol (HTTP, default port=80) and Secure Socket Layer (SSL, default port = 443) will be used for communication between the web server and client. Proxies will be used for remote attestation in the MEITC system.

Access to the web server will be done through a Javascript compliant web based browser: in order to guarantee the trustworthiness of the whole system, the browser and the web server will communicate on a trusted channel using HTTP on top of the conventional TLS/SSL protocols enabled for the mutual platform authentication.

MEITC system components are named as follows. Each compartment holds one or more of these components.

- MEITC WebServer component (in 1st compartment)
- MEITC MailServer component (in 2nd compartment)
- MEITC ApplicationServer component (in 1st compartment)
- MEITC Administrator component (in 1st compartment)
- MEITC Attestator component (in 1st compartment)
- MEITC Connector component (in 1st and 2nd compartments)
- MEITC CertificateServices component (in 2nd compartment)
- MEITC DBServer component (in 2nd compartment)
- MEITC LogServer component (in 2nd comparment)
- MEITC Sealing component (dom0 and 2nd compartment)
- MEITC Measurement component (dom0)
- MEITC TimeServices component (in 2nd component)
- MEITC TrustedBoot component (dom0)
- MEITC Webmail component (in 1st compartment)

The items in red above show the corrent placement of compartments (i.e sealing, trusted boot and measurement).

As mentioned above, these system components do not necessarily need to be on their own compartments. Components can logically be distributed in two or more compartments for better scalability and preserving machine power. Detailed explanations are made for these components in the next section.

Detailed explainations are made in each of the subsections of the following paragraph which explains each system components respectively.

2.4 MEITC Components and Software Modules

2.4.1 MEITC WebServer Component

Purpose: This component is the front-end for users and the system administrators. Users and system administrators of MEITC will connect to this web server via their web based browsers to compose or read e-mail messages.

Responsibilities:

1. Web Server ensures the connection between the MEITC MS and Administrator component in many cases.
2. When adding a new user to the system, WS takes user information from the Administrator component. It checks whether the user is already defined in the system or not. If the user is already defined, it send an error message to be displayed on the Administrator component.
3. When a user wants to access his inbox in order to send or receive a message, MEITC WS component takes inbox data from MEITC MS component.
4. Web server shows the administration interface to request, revoke or delete a certificate.

Collaboration: MEITC MailServer Component, MEITC DBServer Component, MEITC CertificateServices Component, MEITC Administrator Component

SRS References: 1.2, 2.1.1, 2.1.4, 2.1.5, 2.1.6, 2.2, 3.2-UC10, 3.2-UC12, 3.2-UC40, 3.2-UC50, 3.2-UC60, 3.2-UC100, 3.2-UC110, 3.2-UC120, 3.6.2, 3.6.3

2.4.2 MEITC MailServer Component

Purpose: This component handles all the e-mail traffic and it will use the Trusted Log and the Certificate Service Provider (CSP) to implement the security services for messages.

Responsibilities:

1. MEITC MailServer component handles all e-mail traffic using security services ensured by MEITC infrastructure.
2. When user wants to access to his inbox in order to read/write/delete an e-mail, MEITC MailServer component gets/sends inbox request from the MEITC WS and requests this data from MEITC DBServer component.

Collaboration: MEITC LogServer component, MEITC CertificateServices component, MEITC DBServer component, MEITC WebServer component.

SRS References: 1.2, 2.1.4, 2.1.5, 2.2, 3.2-UC10, 3.2-UC12, 3.2-UC60, 3.2-UC100, 3.2-UC110, 3.2-UC120, 3.6.2, 3.6.3, 3.6.5

2.4.3 MEITC DBServer Component

Purpose: This component will host users' mailbox authentication information together with keys, certificate status information and certificates.

Responsibilities:

1. MEITC DBServer component keeps user e-mail preferences including passwords.
2. MEITC DBServer component is responsible for storing user certificates and keys.
3. MEITC DBServer component gets DB requests from webmail and sends required information back.
4. MEITC DBServer component can get/set certificate status information. Database is directly informed from certificate requests by using request and certificate fields in mailbox table which corresponds to each user. There are 3 use cases for certificates, mentioned below:

    1. User has just been created: Both the request field and certificate field is empty.
    2. User requests a certificate, but not yet approved by the administrator: Request field is populated with certificate request file. Certificate field is still empty.
    3. User requests a certificate and this is approved by the administrator: Request field still includes the certificate request file. Certificate field is populated with assigned certificate.

Collaboration: MEITC MailServer component, MEITC WebServer Component, MEITC Administrator component

SRS References: 1.2, 2.1.4, 2.1.5, 2.1.8, 2.2, 3.2-UC10, 3.2-UC12, 3.2-UC40, 3.2-UC50, 3.2-UC60, 3.2-UC100, 3.2-UC110, 3.2-UC120, 3.4, 3.6.2, 3.6.3, 3.6.4, 3.6.5

2.4.4 MEITC LogServer Component

Purpose: This component guarantees the integrity checking of e-mails and also the non-repudiation by storing a record for each incoming and outgoing e-mail.

Responsibilities:

1. MEITC LogServer component holds records for each e-mail sent includingthe hash of the message.
2. When a user wants to send an e-mail, MEITC MS sends e-mail data to MEITC LS and it generates a secure record for these e-mail data and stores this record. This will be done by referring to the ticks of TPM together with the referenced time frame taken from a trusted time server.

Collaboration: MEITC MailServer component, MEITC TimeServices component

SRS References: 1.2, 2.1.4, 2.1.8, 2.2, 3.2-UC110, 3.4, 3.6.4

2.4.5 MEITC CertificateServices Component

Purpose: This component stores users' digital certificates and keys for signing and encrypting e-mails.

Responsibilities:

1. MEITC CertificateServices component stores user certificates for further information retrieval.

2. MEITC CertificateServices component stores user keypairs for encryption and signing of e-mails.
3. When MEITC Administrator component sends a certificate generation or revoke request, MEITC CertificateServices component generates the certificate together with user keys.

Collaboration: MEITC Administrator component, MEITC ApplicationServer component, MEITC DBcomponent

SRS References: 1.2, 2.1.4, 2.2, 2.4, 3.1, 3.2-UC60, 3.2-UC110, 3.6.2, 3.6.3, 3.6.4

2.4.6 MEITC Administrator Component

Purpose: This component provides interaction between the users (including system administrator) and other components.

Responsibilities:

1. This component is a java application which receives user/administrator requests and acts accordingly. Administrator component ensures all the interaction between the users and the MEITC components
2. Administrator component receives the “adduser”, “delete user” and “modify user” requests from administrator and component.User management is done via administrator component, which in turn passes the requests to web server.
3. Administrator component receives the signature generation, revoke or view request from user or administrator and passes this to MEITC WebServer component. (again, same applies here).

Collaboration: MEITC Webmail component, MEITC ApplicationServer component, MEITC WebServer component

SRS References: 1.2, 2.1, 2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.5, 2.1.6, 2.1.8, 2.2, 2.3, 3.2-UC40, 3.2-UC50, 3.2-UC60, 3.2-UC110, 3.2-UC120, 3.6.3, 3.6.4, 3.6.5

2.4.7 MEITC Webmail Component

Purpose: This component provides web based mail services.

Responsibilities:

1. Sending, receiving and displaying the e-mails passed by MailServer component.
2. Deleting an e-mail on behalf of a user's request.
3. Authenticating via Mailserver component.

Collaboration: MEITC ApplicationServer component, MEITC WebServer component

2.4.8 Meitc System Component

Purpose: It is the core module of the system which constitutes the main function for MEITC system components. It is point of entry for the application. It can be thought as the engine ensuring the working status of different components.

Responsibilities:

1. This the core module for the MEITC system. As it is the point of entry for the system, it has little direct responsibilities. All other responsibilities are shared between the components and software classes.
2. In the TPM take ownership operation, the SystemAdministrator chooses the operation within it.
3. System wants owner and SRK password from the SystemAdministrator.
4. When he gives the password, it calls the TPMTakeOwnership method via the TPMHelper class.

Collaboration: MEITC LogServer component, MEITC WebServer component, MEITC DatabaseServer component, MEITC MailServer component

SRS References: 3.2-UC11

2.4.9 MEITC Measurement component

Purpose: Dom0 includes the Xen kernel which includes all the trusted computing and virtualization functionalities provided by the OpenTC infrastructure. Dom0 includes all the Trusted Computing Base and virtualization functionalities for our implementation.

Responsibilities:

1. During the MEITC system startup process the dom0 checks the compartments in which the MEITC DB Server, MEITC WS and MEITC MS stay and if the integrity checks are successful, dom0 starts the MEITC DB Server, MEITC WS and MEITC MS.
2. MEITC WS, MEITC MS and MEITC DB Server sends messages to dom0 that they have been properly started.

Collaboration: MEITC CertificateServices component, MEITC LogServer component, MEITC WebServer component, MEITC MailServer component, MEITC DatabaseServer component

SRS References: 3.2-UC10

2.4.10 MEITC TrustedBoot Component

Purpose: This component contains TrustedGrub boot loader which checks the measurement values of different compartments in the system and ensures the system integrity.

Responsibilities:

1. During MEITC startup procedure SystemAdministrator boots dom0 and tGrub checks integrity of dom0 and domUs in which the MEITC WS, MEITC MS and MEITC DB Server stay.
2. If the integrity check is successful the compartments are started. If not, the operation is stopped.

Collaboration: None

SRS References: 3.2, 3.2-UC10

2.4.11 MEITC Attestator component

Purpose: This component forms the source proxy application during the remote attestation procedure.

Responsibilities:

1. In the remote attestation procedure, server opens connection with client.
2. Server gets its own TPMQuote and sends it to client.
3. Client verifies the server's quote and if the verification is successful, it sends a message to the server.
4. Client takes its own TPMQuote and sends it to server.
5. Server verifies client's quote and if the verification is successful, it sends message to client and attestation is succeeded.

Collaboration: domT and domU

SRS References: 3.2-UC12

2.4.12 MEITC Connector Component

Purpose: This component connects domT and domU.

Responsibilities:

1. Connector in domT receives MySQL commands from Administrator component and sends these commands to domU via an encyrpted channel.
2. Connector in domU recevies these commands and interprets them, sending directly to MEITC Database component.
3. Connector in domU sends back possible errors and outputs.

Collaboration: domT and domU

SRS References: None
(read until here)

2.4.13 MEITC ApplicationServer Component

Purpose: This component runs MEITC Administrator component and MEITC Webmail component

Responsibilities:

1. MEITC Administrator component is responsible from starting MEITC Administrator component and MEITC Webmail component and ensures that they run properly.

Collaboration: MEITC Administrator component, MEITC Webmail component

SRS References: None

2.4.14 MEITC Sealing Component

Purpose: This component seals MEITC CertificateManager component database against TPM

Responsibilities:

1. MEITC Sealing component seals the keys which reside in MEITC CertificateManager component database and unseals them when requested by domT.

Collaboration: MEITC CertificateManager component, domT

SRS References: None

2.4.15 MEITC TimeServices Component

Purpose: This component provides log files utilizing secure time service from TPM.

Responsibilities:

1. MEITC TimeServices component provides trusted time services to for MEITC LogServer component.

Collaboration: MEITC LogServer component

SRS References: None

3 Project standards, conventions and procedures

3.1 Design standards

For the Java and JSP parts of the project, the design method used is object-oriented in design. The standard UML diagrams is used for representing MEITC components and software classes.

For the parts of the system which are connected directly to the TPM chip, a procedural programming language, C, is used so for these parts some special attention has to be paid. This means:

- We don't use jumps (goto)
- We use hierarchical decomposition: if nesting becomes too deep, define a new lower-level module.

For installation and system startup procedures some bash shell scripts will be written. Since various Linux distributions use different local startup scripts, the installer will

detect and identify which Linux distribution is used and install the bash scripts accordingly. The system will use basic bash 2.0 functions to guarantee a foolproof system startup by checking for exceptions, control and eliminate possible errors.

3.2 Documentation standards

The tools used for the detailed design are:

- Visual Paradigm UML modelling software for UML diagrams
- OpenOffice for documentation
- Eclipse for Java/JSP programming
- Vi and Kate text editors for writing shell scripts

3.3 Naming conventions

For Java/JSP part:

- Variables will be documented in the following style:
  `int maxSpeed; // the maximum speed of the upper arm`
- Exceptions are defined as Java-style. For example: `TcException`
- Class names will be documented in the same way as variable names, except that the first letter must be a capital letter, for instance: `Robot`.
- Interface names will be begin with a big 'I', for instance: `IcompartmentManager`
- Method names will be in the following style:
  `void thisIsAFunction (int a, int b);`

Table 1: Naming Convention for Java/JSP

File Type	***Extension***	**Comment**
documentation	html	These are javadoc generated from Java files
source code	.java	Should be same name as the class name

For C part:

- Variables will be like Java style and will be documented in the same line:
  `int anyVariable; // this is a comment`
- Function names will be in the following sytle and will be document with a single line:
  ```
  // this is a comment for this function
  void this_is_a_function (int a, int b);
  ```

Table 2: Naming Convention for C

File Type	***Extension***	**Comment**
documentation	.odt	If needed some specific documentation will be made in OpenOffice text format. In C files there will be comments for code fragments.
source code	.java	Should be same name as the class name

For shell scripts:

- Shell variables are named using the same general rules as C programming language variables. Capital letters will be used for readability, i.e
  `TCSD_STARTUP=/usr/sbin/tcsd`
- Function names are all lower letter.
  `function startvnc () { exit 0; }`

Table 3 : Naming Convention for shell scripts

File Type	***Extension***	**Comment**
documentation	.odt	If needed some specific documentation will be made in OpenOffice text format. Necessary comments will be placed in shell scripts
source code	.sh	The script name consists of lowercase letters

3.4 Programming standards

During the MEITC application development the following standards on behalf of coding and commentary constructs and layouts are used. This means that source code written in the project must comply to these standards.

- Standard file headings
  - Header file headings (For .h files):

```
// MEITC header file: -file name-, version: -version number-
// Author: -Author(s) of document-
// Created on: -creation date-
// Description: -description of contents-
```

  - Source file headings (For c-code (.c) files):

```
// MEITC source file: -file name-, version: -version number-
// Author: -Author(s) of document-
// Created on: -creation date-
// Description: -description of contents-
```

  - Source file headings (For java (.java) files):

    Standard javadoc style headings will be used. For details, see:
    http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

  - Source file standard(For(.jsp) files):

    .jsp source files will be coded in the following standard

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1-strict.dtd">

<!-- JSP that take does something  -->

<%-- page settings --%>
<%@ page import = "java.util.*" %>

<html xmlns = "http://www.w3.org/1999/xhtml">

      <!-- head section of the document -->
      <head>
            <title>Some heading</title>
      </head>

      <!-- body section of the document -->
      <body>
      <% some jsp statement %>
      some html statement

      <% some jsp statement

      %>
      </body>
</html> <!-- end XHTML document -->
```

- Standard class definitions (Java classes)

```
class -class name- ( (extends, implements) -parentclass-)
   {
   -tab- -variable declarations-
   -tab- -constructor declarations-
   -tab- -method declarations-
   }
```

- Standard method declarations (Java and C files)

```
-return type- -method name- ( -parameters- )
```

- Standard methods definitions (Java and C files)

```
-return type- -method name- ( -parameters- )
// pre-condition: -pre-condition description-
// post-condition: -post-condition description-
// returns: -return description-
{
```

```
- implementation -
}
```

- Standard variable names

  Variable names in British English. If a variable name is a combination of several words then these words are separated by capital letters. For instance: maxInt.

- Comment language

  All comments are written in British English. Shell comments are written clear and short. No comments are written after the command, and they will be placed in a new line.

3.5 Database Design

In this documentation a main title (level 1) exists for each table in the database. For each table; “Fields”, “Referential Integrity Constraints” sub titles (level 2) exists. If necessary “Indexes” sub titles could be exists in this level (level 2). Before “Fields” sub titles descriptions about table exists. This descriptions explains table fields and their usage.

A matrix exists under the “Fields” title. The rows of this matrix contains field informations and the column of the matrix contains field name, fields type, default value and descriptions. All field names is defined in the database as given below. (small/big letter etc. with properties).

MEITC administration screen uses postfixadmin database tables. Postfixadmin management applicaton works with Postfix mail server with no issues.

3.5.1 Admin table

This table contains administrator user information. This information is used to login MEITC administration module.

Fields of admin table is defined below.

Name	Type	Default Value	Description
username	varchar(255)		Administrator user name
password	varchar(255)		Administrator password
created	datetime	0000-00-00 00:00:00	Creation time of the record
modified	datetime	0000-00-00 00:00:00	Last update time of the record
active	tinyint(1)		Status of the record

Index of admin table is defined below.

Name	Fields	Description
username	username	To disallow dublicate usernames and allows quick

		access to records

3.5.2 Alias table

Postfix alias information exists in this table. Postfix mail server uses address ve goto fields. This tables is similar to /etc/aliases file.

Fields of alias table is defined below.

Name	Type	Default Value	Description
address	varchar(255)		Source email address
goto	text		This field is destination email address. Multiple destination email addresses need to be separated by a "," (comma).
domain	varchar(255)		Domain name
created	datetime	0000-00-00 00:00:00	Creation time of the record
modified	datetime	0000-00-00 00:00:00	last update time of the record
active	tinyint(1)	1	status of the record

Index of alias table is defined below.

Name	Fields	Description
address	address	To disallow dublicate usernames and allows quick access to records

3.5.3 domain table

Domain information exists in this table. In this table domain and description fields is used by postfix mail server.

Fields of domain table is defined below.

Name	Type	Default Value	Description
domain	varchar(255)		domain name
description	varchar(255)		description of the domain name
aliases	int(10)	0	alias quantity
mailboxes	int(10)	0	user account quantity
maxquota	int(10)	0	maximum quota
transport	varchar(255)		
backupmx	tinyint(1)	0	
created	datetime	0000-00-00 00:00:00	creation time of the record
modified	datetime	0000-00-00 00:00:00	last update time of the record
active	tinyint(1)	1	status of the record

Index table is defined below.

Name	Fields	Description
domain	domain	To disallow dublicate usernames and allows quick access to records

3.5.4 domain_admins table

This table contains domain administration informations. Field table is defined below.

Name	Type	Default Value	Description
username	varchar(255)		domain administration user name
domain	varchar(255)		domain name
created	datetime	0000-00-00 00:00:00	creation time of the record
active	tinyint(1)		status of the record

Index table is defined below.

Name	Fields	Description
username	username	allows quick access to records

3.5.5 Log table

Log information exists in this table. Field table is defined below.

Name	Type	Default Value	Description
timestamp	datetime	0000-00-00 00:00:00	process time
username	varchar(255)		username of the user which made the process
domain	varchar(255)		domain name
action	varchar(255)		process type
data	varchar(255)		information about process

Index table is defined below.

Name	Fields	Description
timestamp	timestamp	allows quick access to records

3.5.6 mailbox table

Postfix account information exists in this table, also including certificates and keypairs for users. The following table shows the fields of mailbox table.

Name	Type	Default Value	Description
username	varchar(255)		postfix user name

password	varchar(255)		postfix user password
name	varchar(255)		real name of the user
maildir	varchar(255)		folder name of the email information which is saved
quota	int(10)	0	quota information
domain	varchar(255)		domain name
created	datetime	0000-00-00 00:00:00	creation time of the record
modified	datetime	0000-00-00 00:00:00	last update time of the record
active	tinyint(1)	1	status of the record
certreq	text		certificate request
cert	text		certificate of user
publickey	text		public key of user
privatekey	text		private key of user

Following table gives index field.

Name	Fields	Description
username	username	To disallow dublicate usernames and allows quick access to records

3.5.7 Vacation table

This table keeps a history of days taken for each vacation by the postfix user.

Name	Type	Default Value	Description
email	varchar(255)		e-mail account
subject	varchar(255)		subject of the message
body	text		message content
cache	text		
domain	varchar(255)		domain name
created	datetime	0000-00-00 00:00:00	creation time of the record
active	tinyint(1)	1	status of the record

Following table gives index field.

Name	Fields	Description
email	email	To disallow dublicate usernames and allows quick access to records

3.6 Software development tools

We used Visual Paradigm UML modelling tool for the object-oriented design. In particular for the drawing of the sequence and class diagrams. For the development of

the software Eclipse is used. This tool is the best known and widely used tool for Java style programming. It consists of a source code editor, java compiler, javadoc generator, jar builder and with some plugins it is possible to make programming in widely different programming languages including JSP. For C and bash scripts standard Linux editors Vi and kate are used.

For our documentation javadoc documents which are generated from the Java source codes are used. OpenOffice is used for writing requirements and design documents.

4 Design issues

Design issues of MEITC regarding portability, maintainability, extendibility and reliability has been explained in “WP06b.2 MEITC specification and test plan”. Most important software system attribute requirements for the MEITC system which will be developed is localization, defined below.

4.1 Localization

The webmail and the administration pages will have multilingual framework. The language determination can be done in two ways.

- Either using the language and locale preferences that are transmitted from the browser to the server using the HTTP request header field “Accept-Language”. Since this header is intended primarily for language and cultural preferences.
- Using the locale determination and localization framework in JSTL.

In our system, we'll go with the second approach. JSTL supports both ways of determining the user's locale preferences. MEITC webmail and MEITC Admin panel will specify a fixed locale (usually one that the user has explicitly selected from the list of supported languages), using JSTL's <fmt:setLocale> action. Once this action is used, the specified locale is used for all locale-sensitive operations. If the <fmt:setLocale> action is been used, locale-sensitive operations will search for the first supported locale from the list of preferred locales provided by the Accept-Language header.

MEITC system is able to show all GUI in English and Turkish, however since a localization framework is introduced, it's possible to localize the system (user and admin screens) to other languages to ease communication.

5 Test plan

MEITC test plan will include the following items during the project development phase. “MEITC Specification and Test Plan” includes a primary test document which is used as a basis for this plan. It's planned that all tests will start by M30, the time 2nd prototype is ready.

1. **Unit tests:** This will be accomplished by the person who wrote the MEITC administration screen. In order to find logical design errors, white box testing will be used.

2. **Module tests:** This testing will be accomplished by all programmers working in MEITC project. All the modules mentioned in use cases and later in MEITC detailed design document will be tested. The expected outputs are those which

are considered valid during the system's normal operation.

3. **Integration test:** This test will be done after all the MEITC components and modules are fully tested using unit tests and module tests. All the upper components (WS, MS, LS, DB) will be combined one by one (i.e incrementally) to the dom0 and integration tests will be made accordingly.

4. **Usability test:** Since MEITC will be used by humans, functional testing should be accompanied by usability tests to measure the layout, efficiency and effectiveness of the user interface. The set of tasks will be identified after the second prototype in M30 is finished. At the same time the analysis method of collected data and representative sample of the real user population will be discussed and finalized.

6 Sequence diagrams

6.1 MEITC system startup sequence diagram

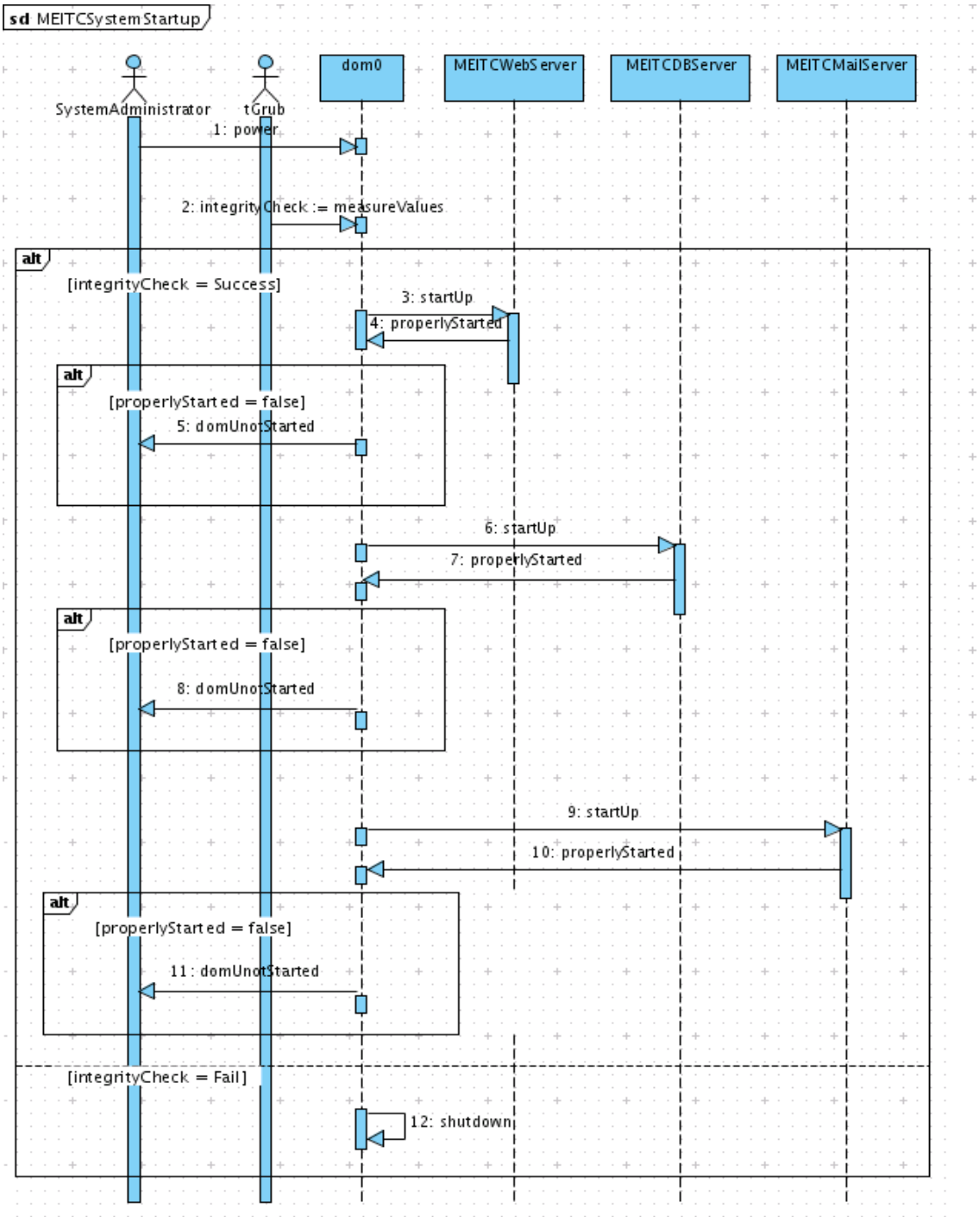


Figure 2:MEITC system startup sequence diagram

6.2 Adding a new user sequence diagram

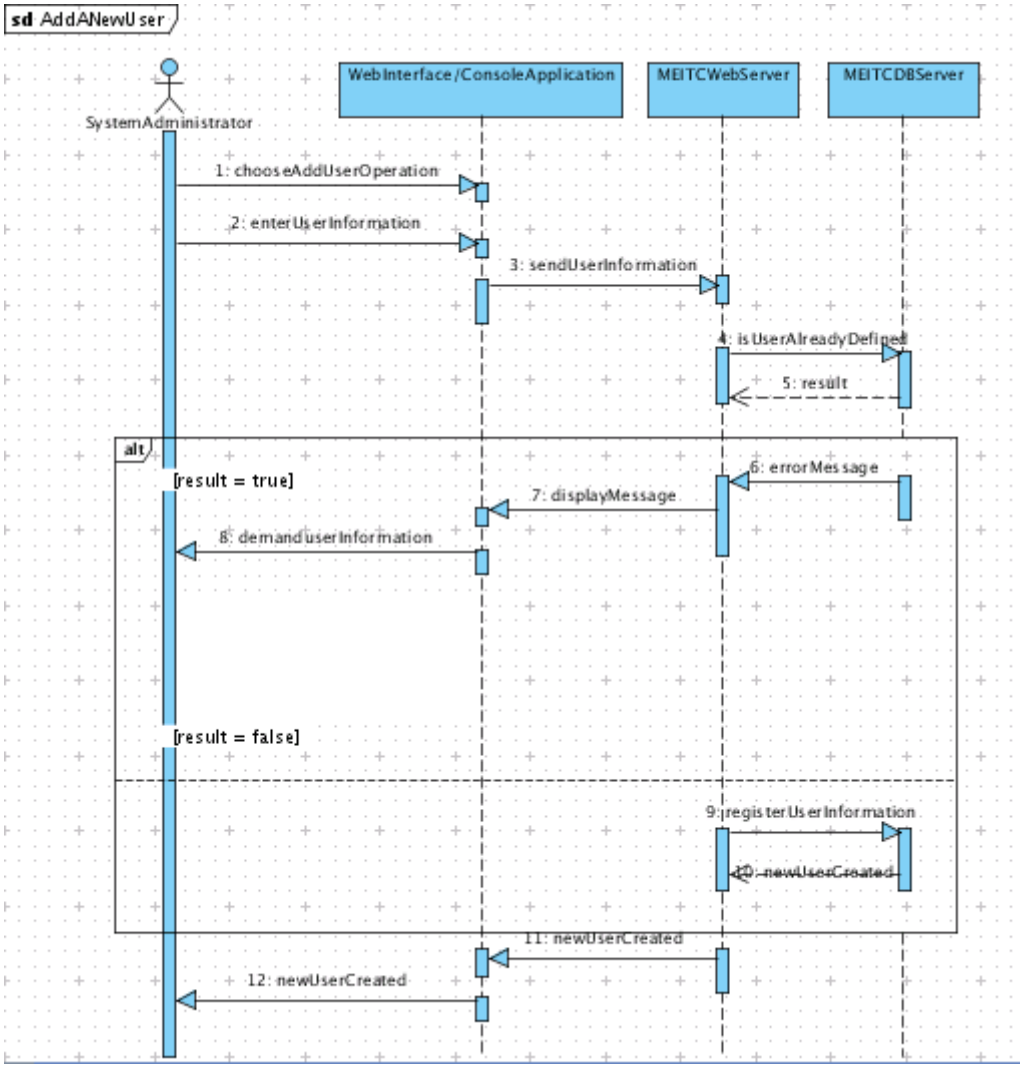


Figure 3:Adding a new user sequence diagram

6.3 Deleting an existing user sequence diagram

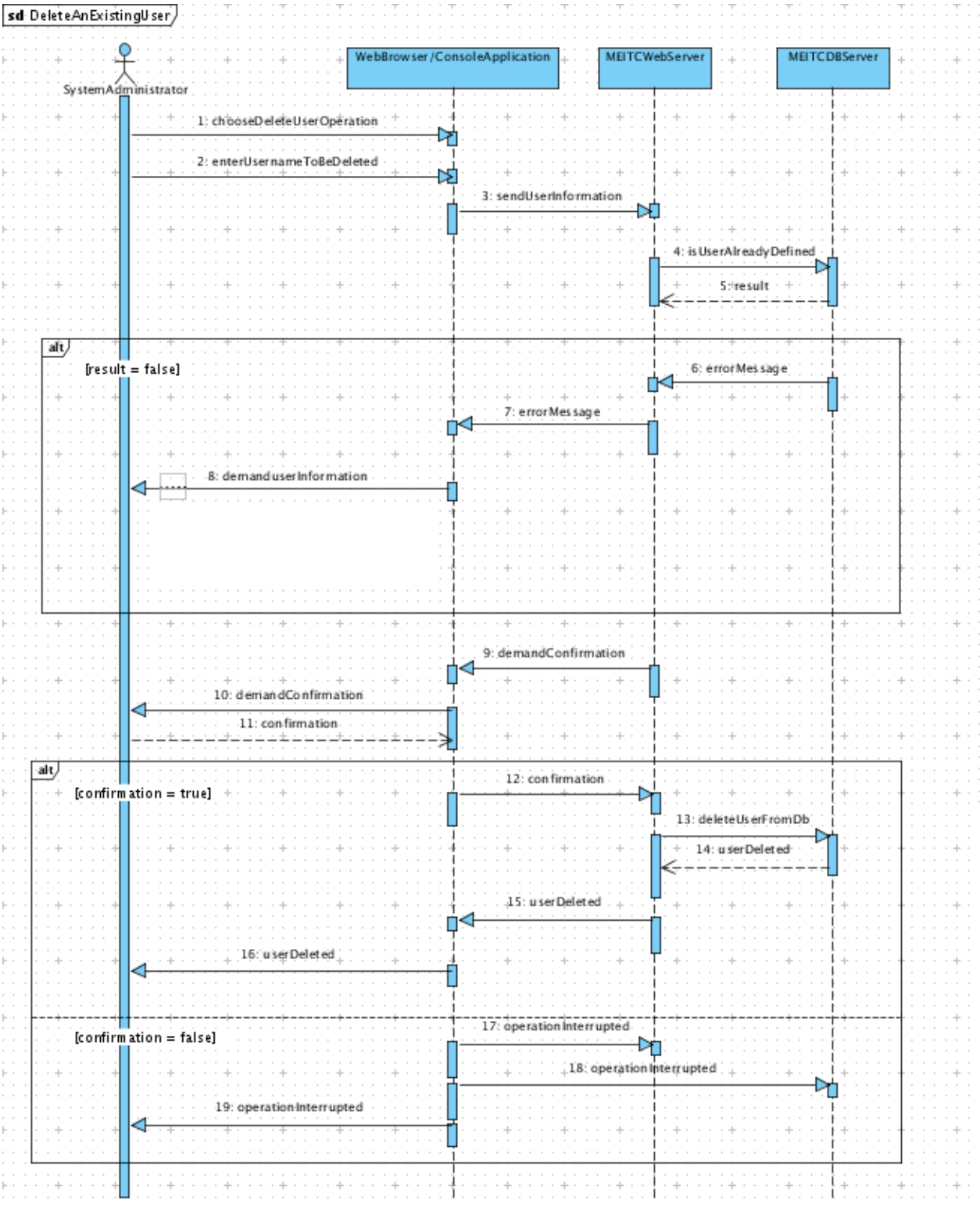


Figure 4:Deleting an existing user sequence diagram

6.4 Sending an e-mail sequence diagram

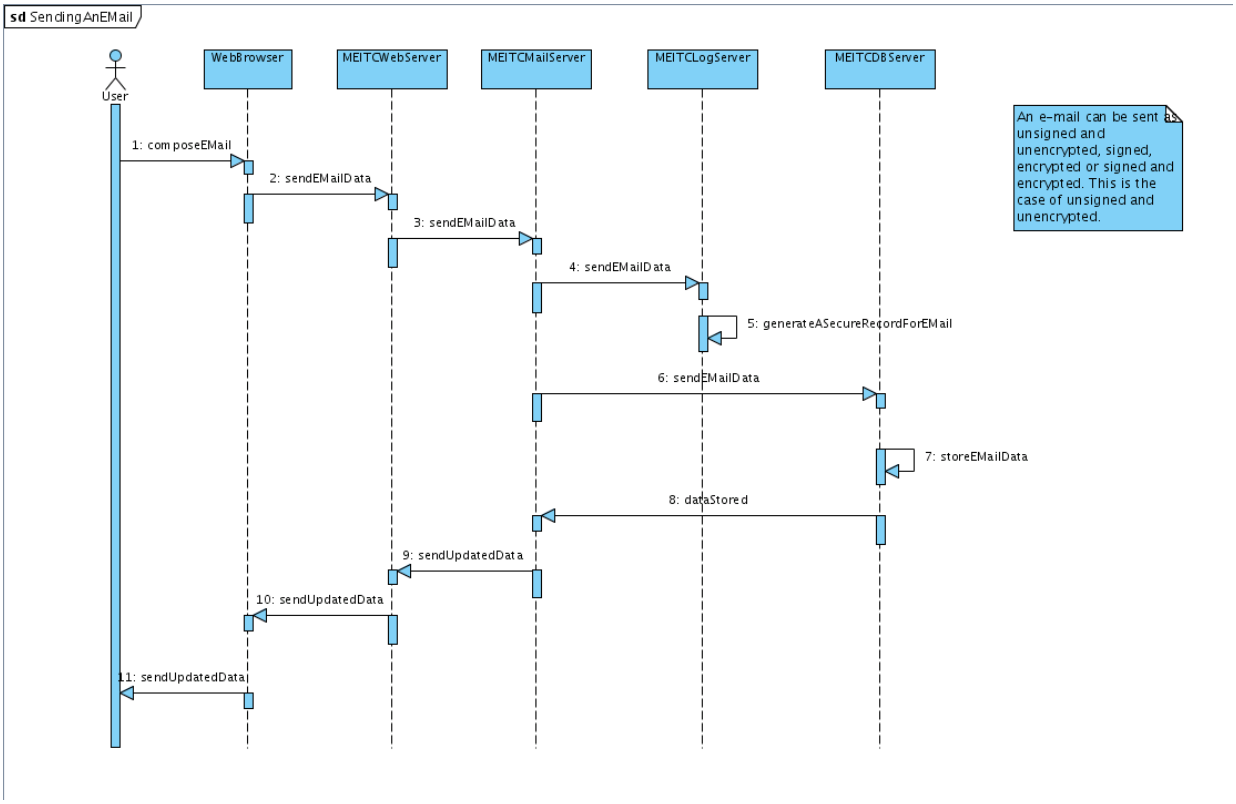


Figure 5:Sending an e-mail sequence diagram

6.5 Sending a signed e-mail sequence diagram

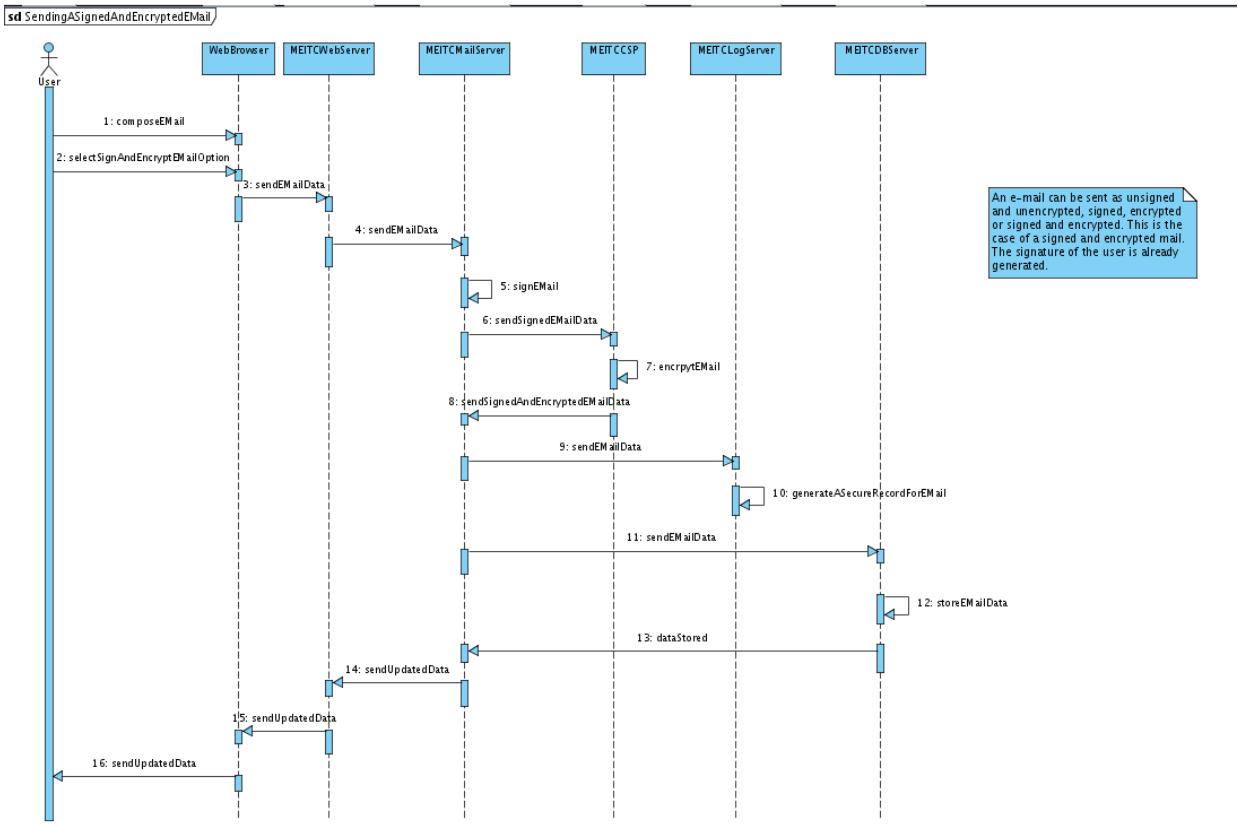


Figure 6:Sending a signed e-mail sequence diagram

6.6 Sending an encrypted e-mail sequence diagram

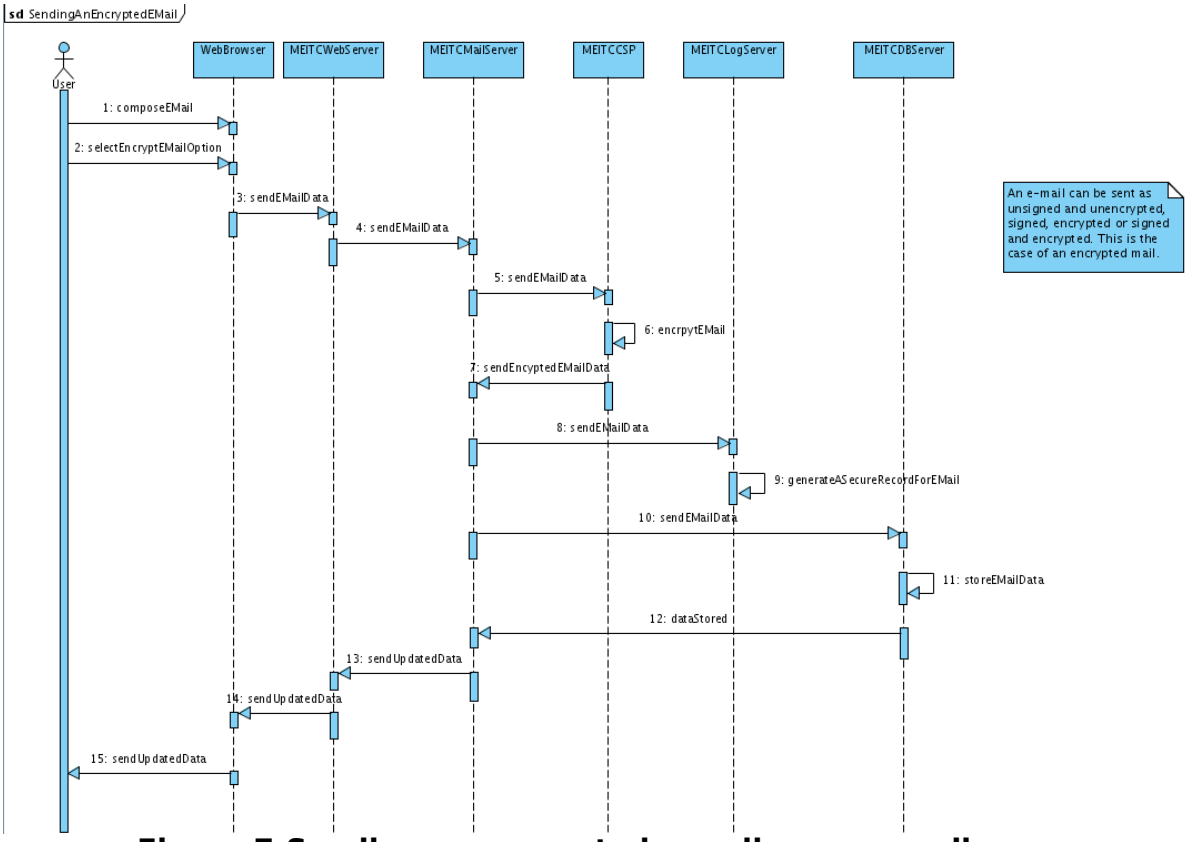


Figure 7:Sending an encyrpted e-mail sequence diagram

6.7 Sending a signed and encrypted e-mail sequence diagram

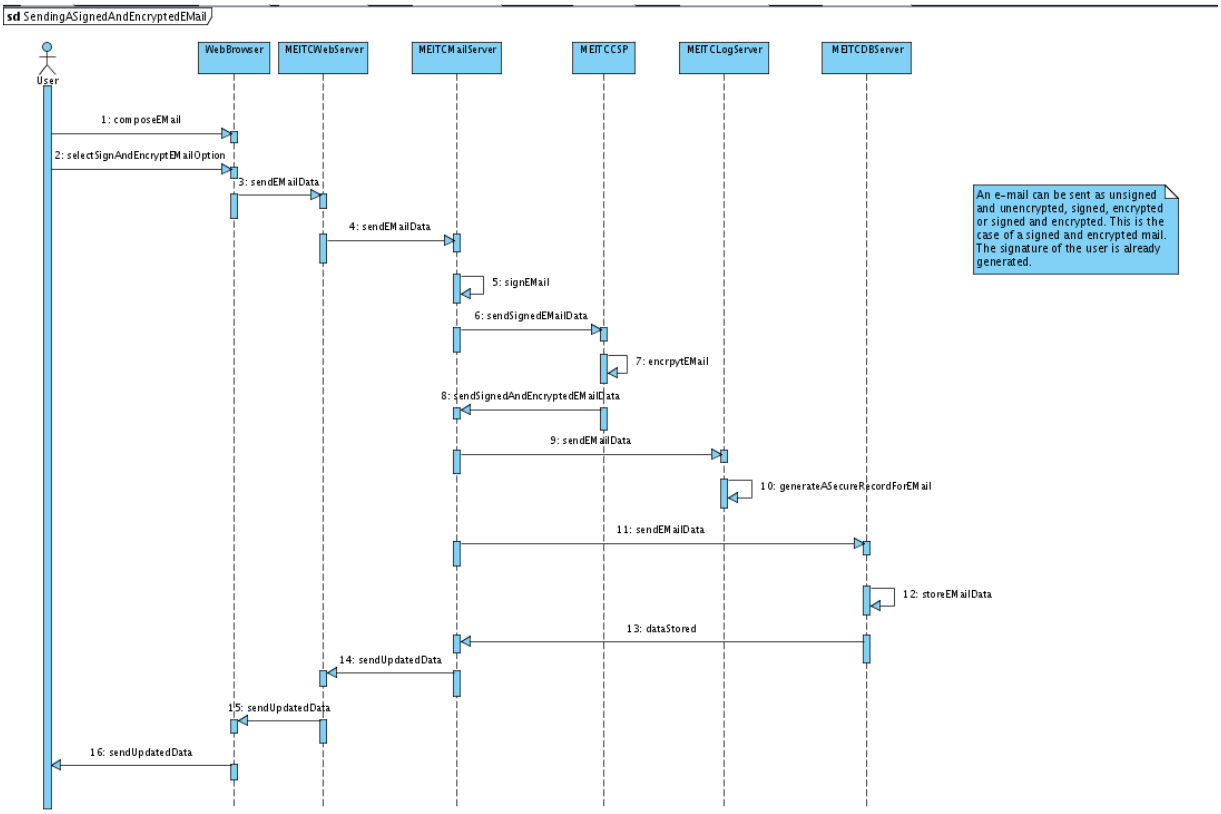


Figure 8:Sending a signed and encrypted e-mail sequence diagram

6.8 Deleting an e-mail sequence diagram

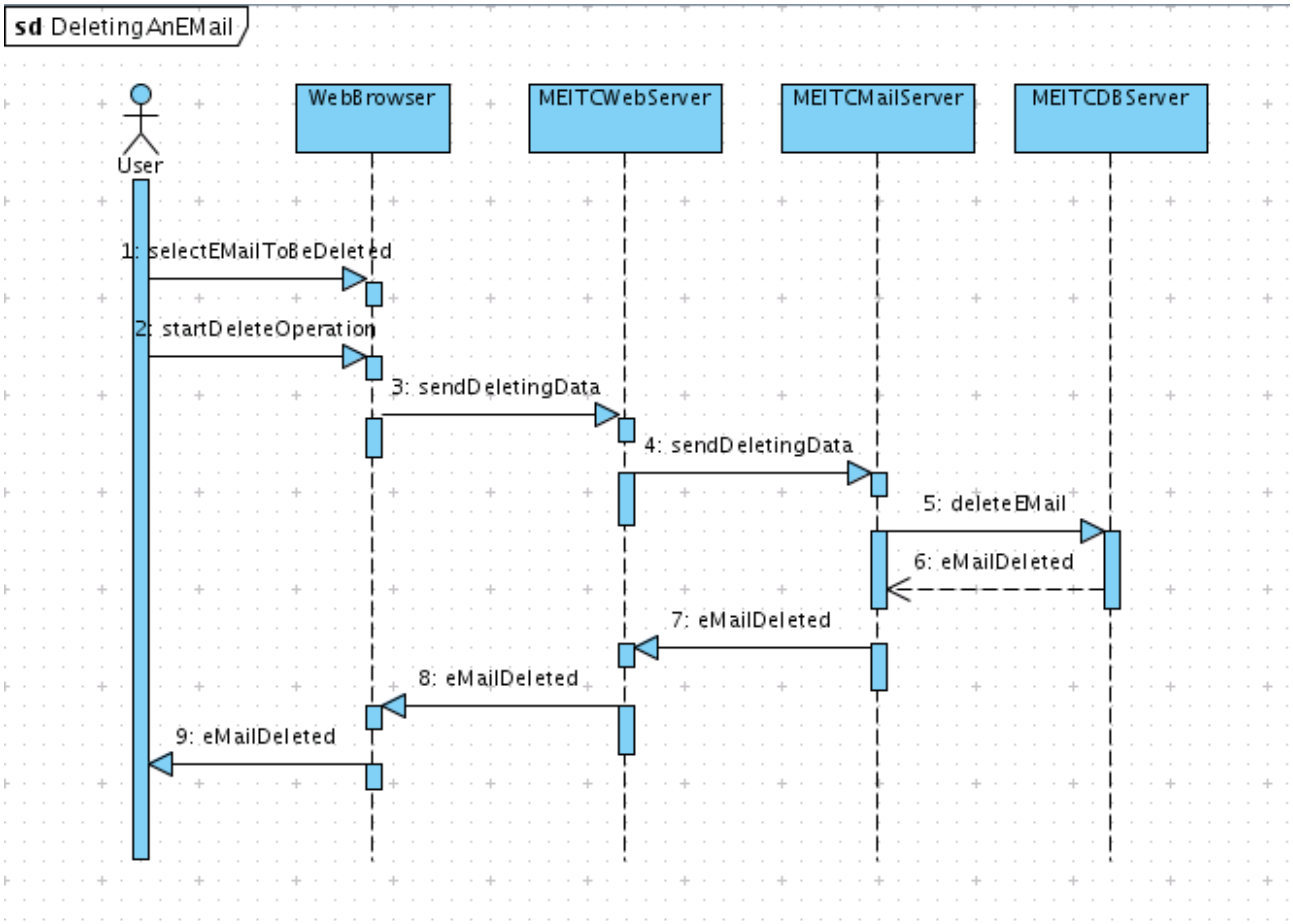


Figure 9:Deleting an e-mail sequence diagram

6.9 Accessing user's inbox sequence diagram

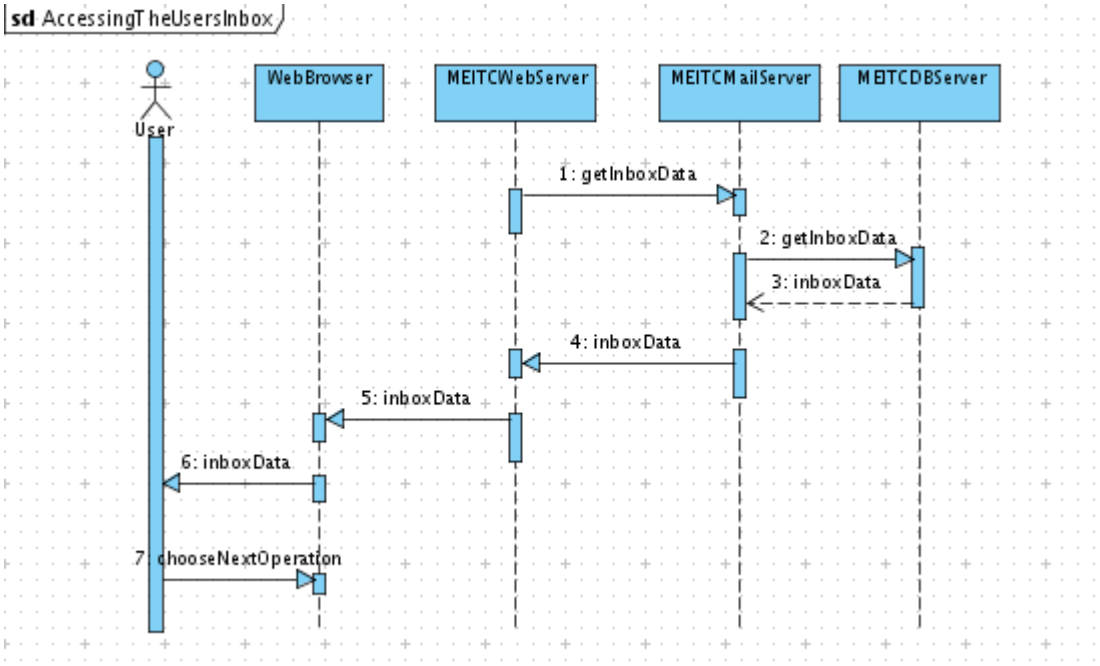


Figure 10:Accessing user's inbox sequence diagram

6.10 User authenticates via MEITC sequence diagram

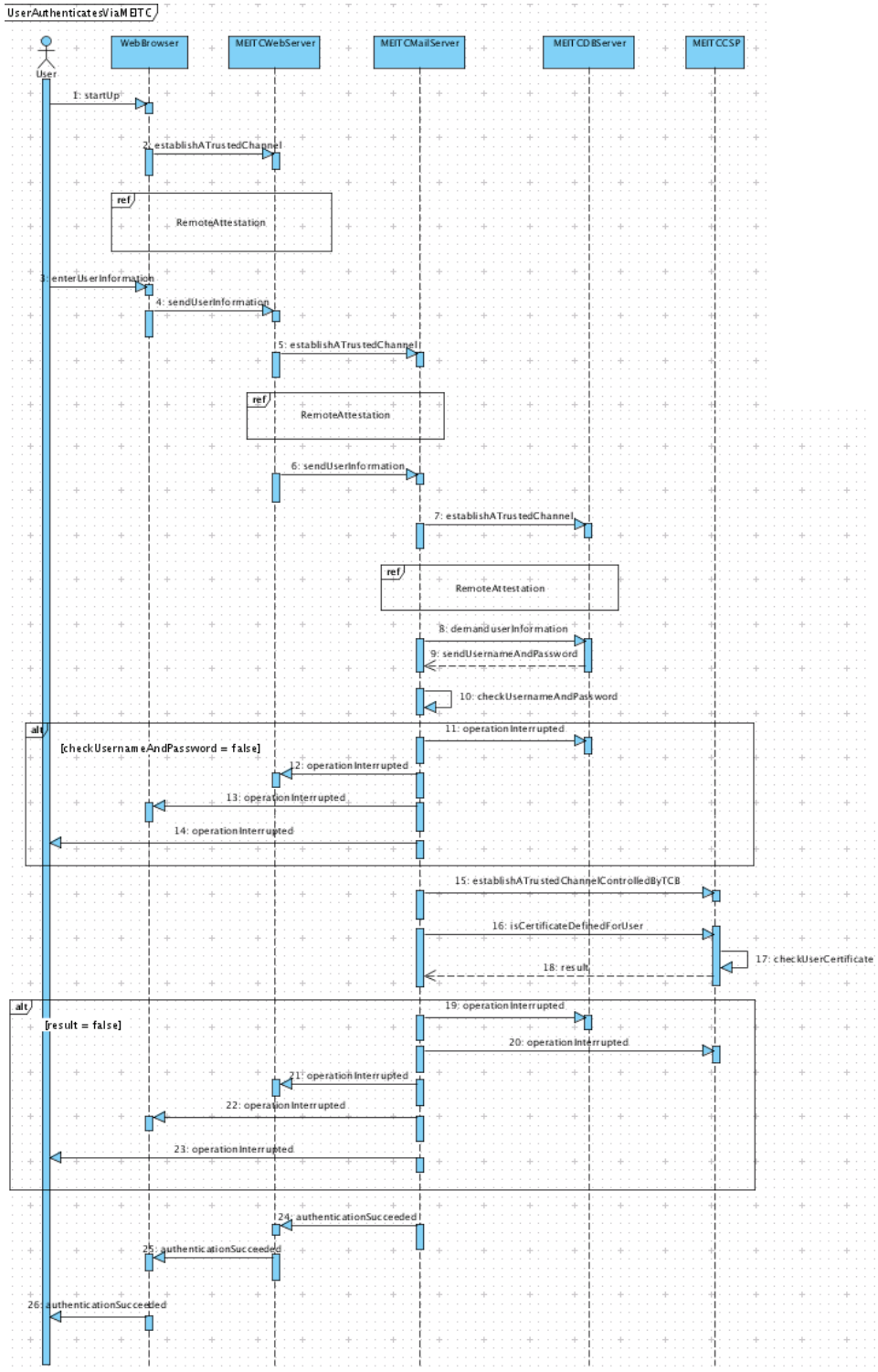


Figure 11:User authenticates via MEITC sequence diagram

6.11 Taking TPM ownership sequence diagram

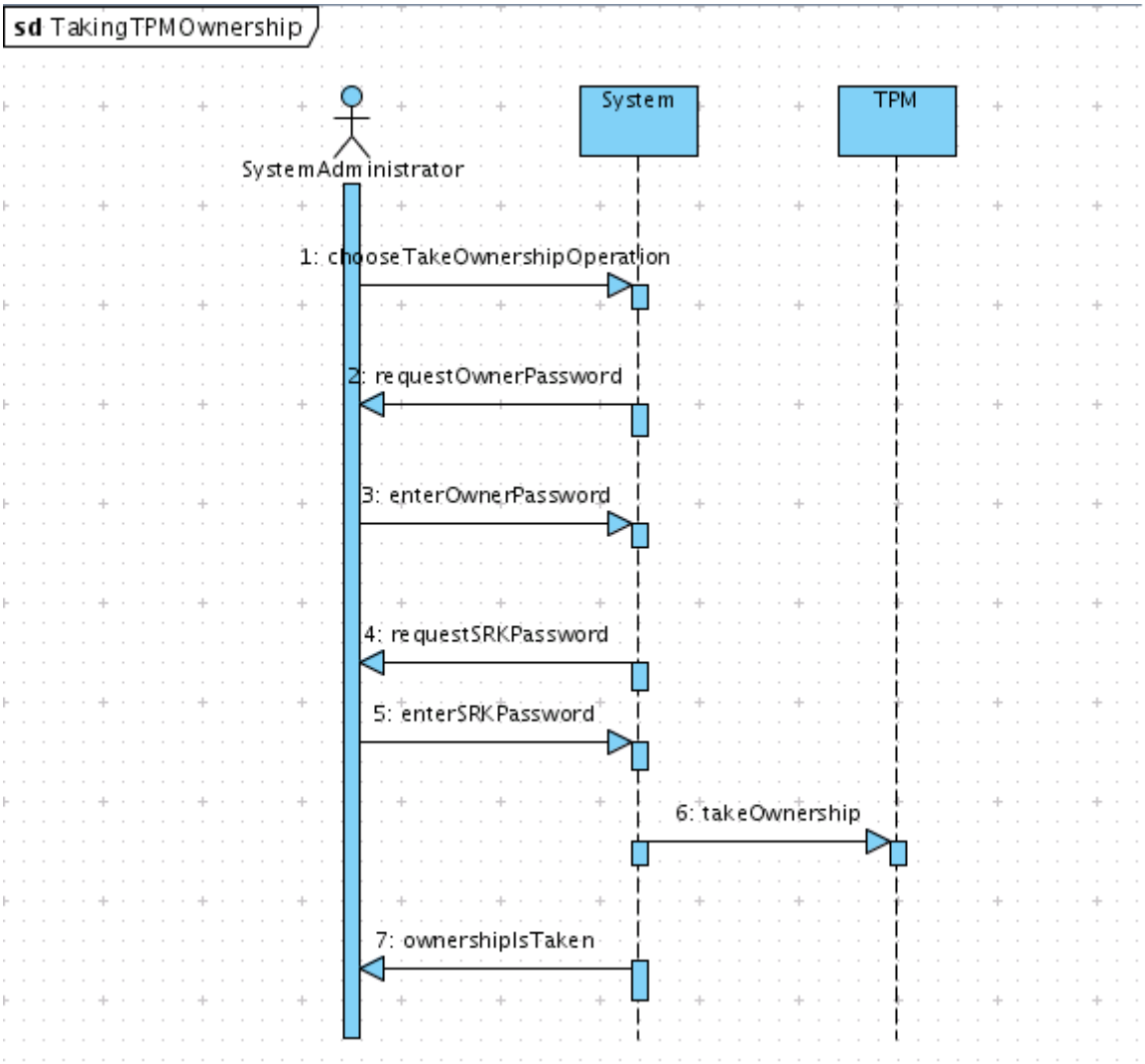


Figure 12:Taking TPM ownership sequence diagram

6.12 Establishing a trusted channel sequence diagram

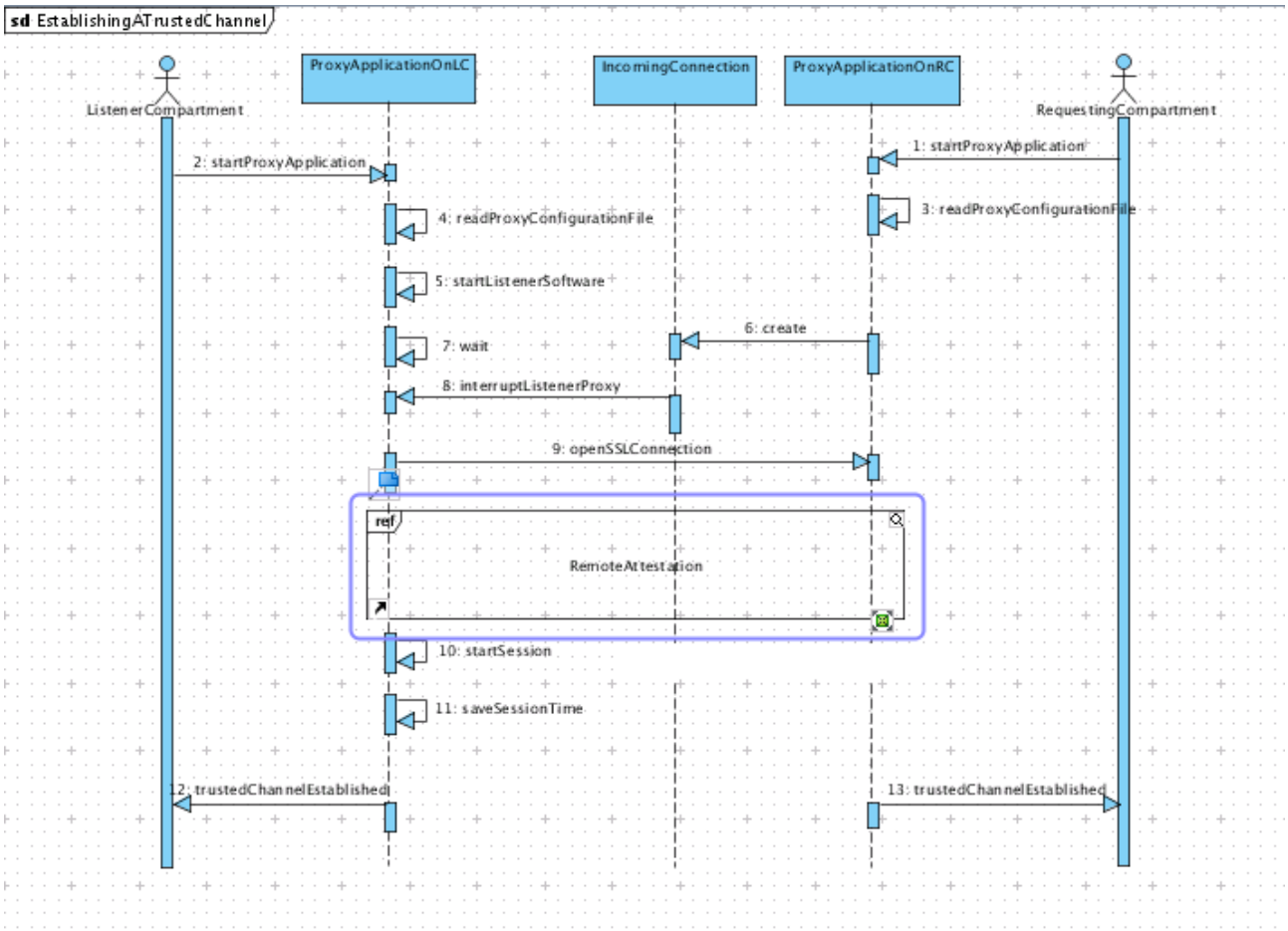


Figure 13:Establishing a trusted channel sequence diagram

6.13 Remote attestation sequence diagram

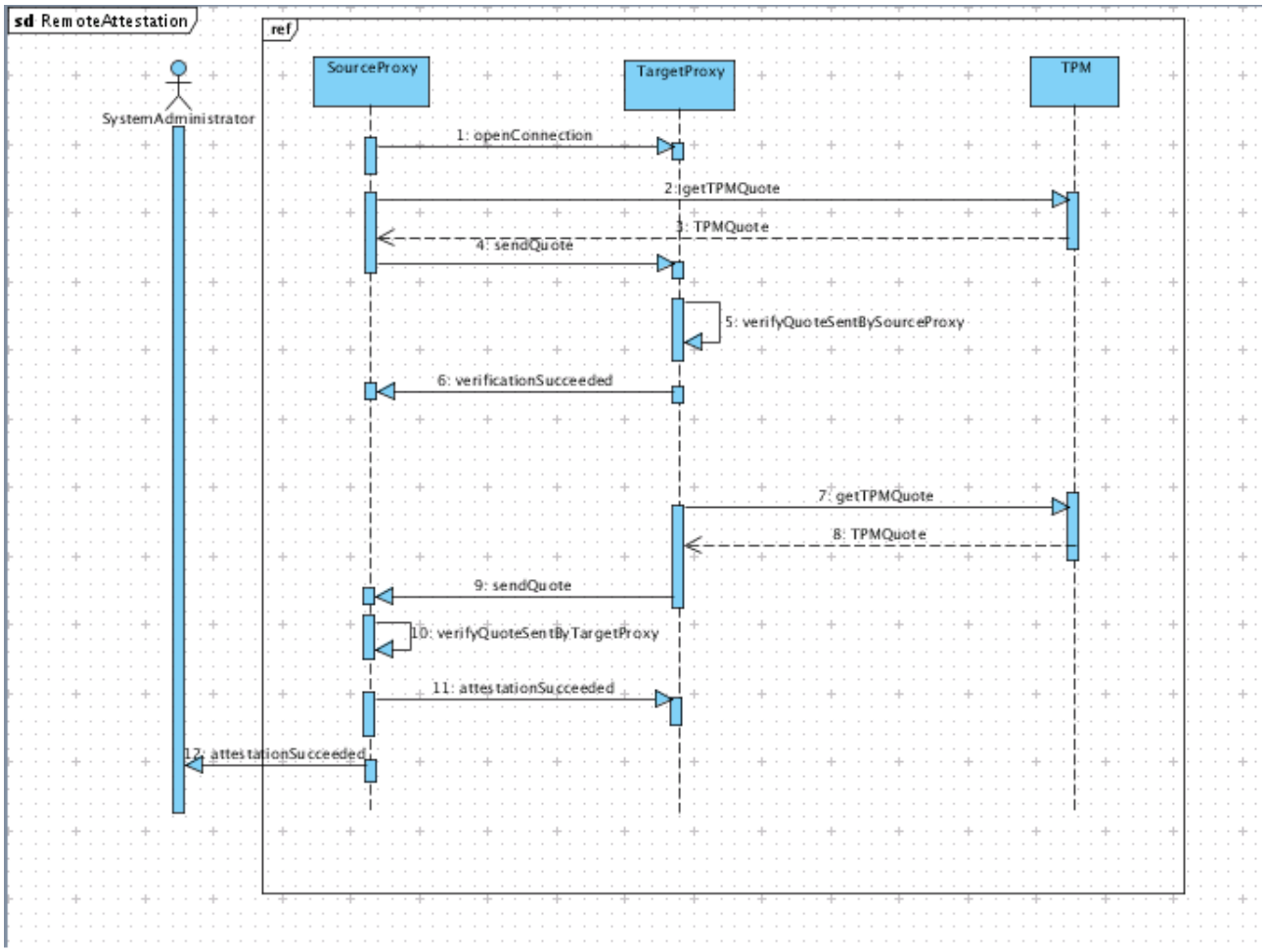


Figure 14:Remote attestation sequence diagram

6.14 Creating a certificate sequence diagram

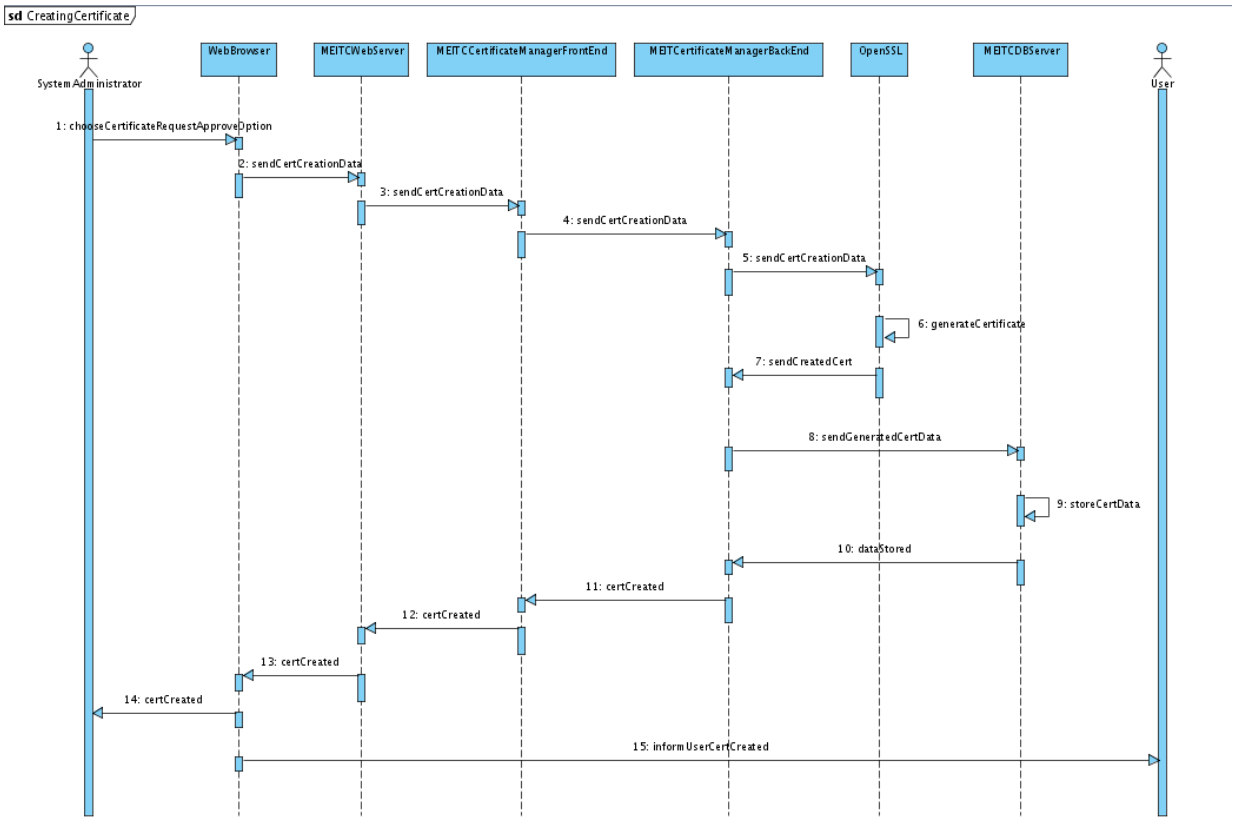


Figure 15:Creating a certificate sequence diagram

6.15 Creating a certificate request sequence diagram

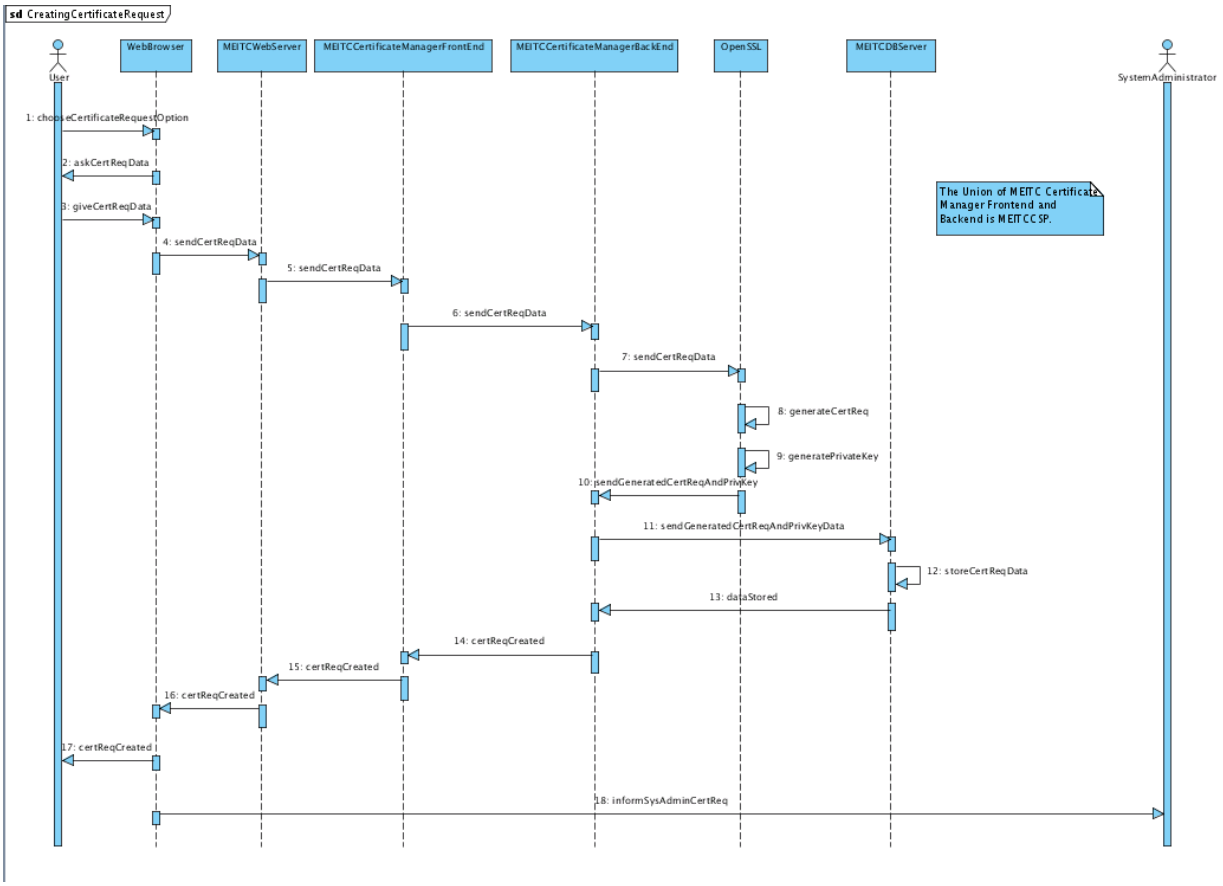


Figure 16:Creating a certificate request sequence diagram

6.16 Creating a certificate revocation request sequence diagram

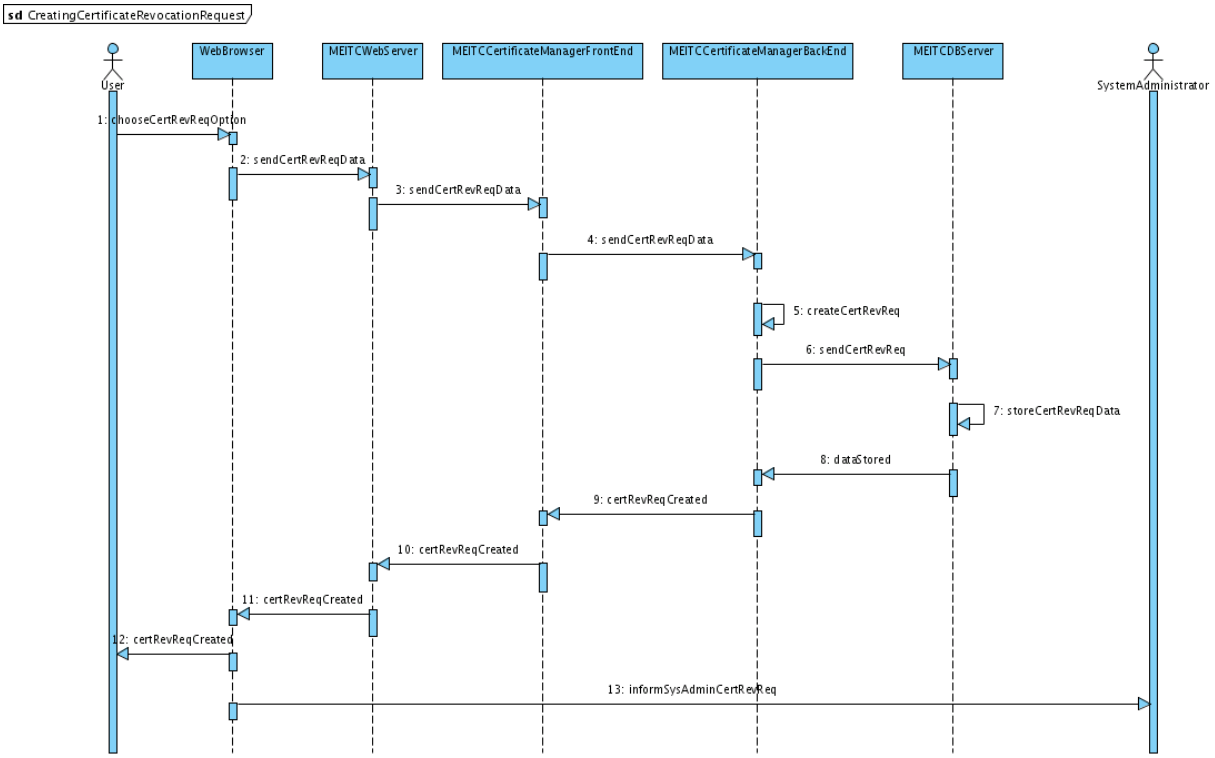


Figure 17:Creating a certificate revocation request sequence diagram

6.17 Revoking a certificate sequence diagram

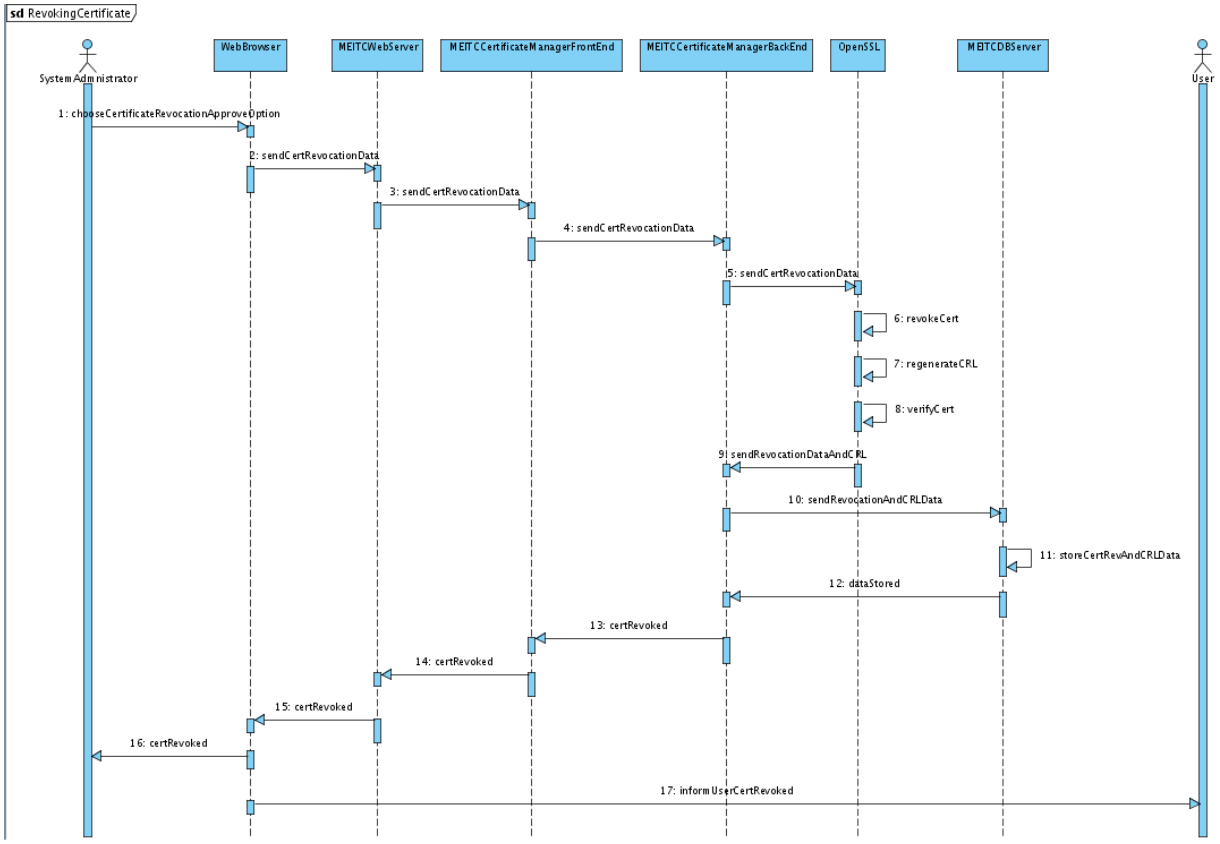


Figure 18:Revoking a certificate sequence diagram

7 MEITC All Classes diagram

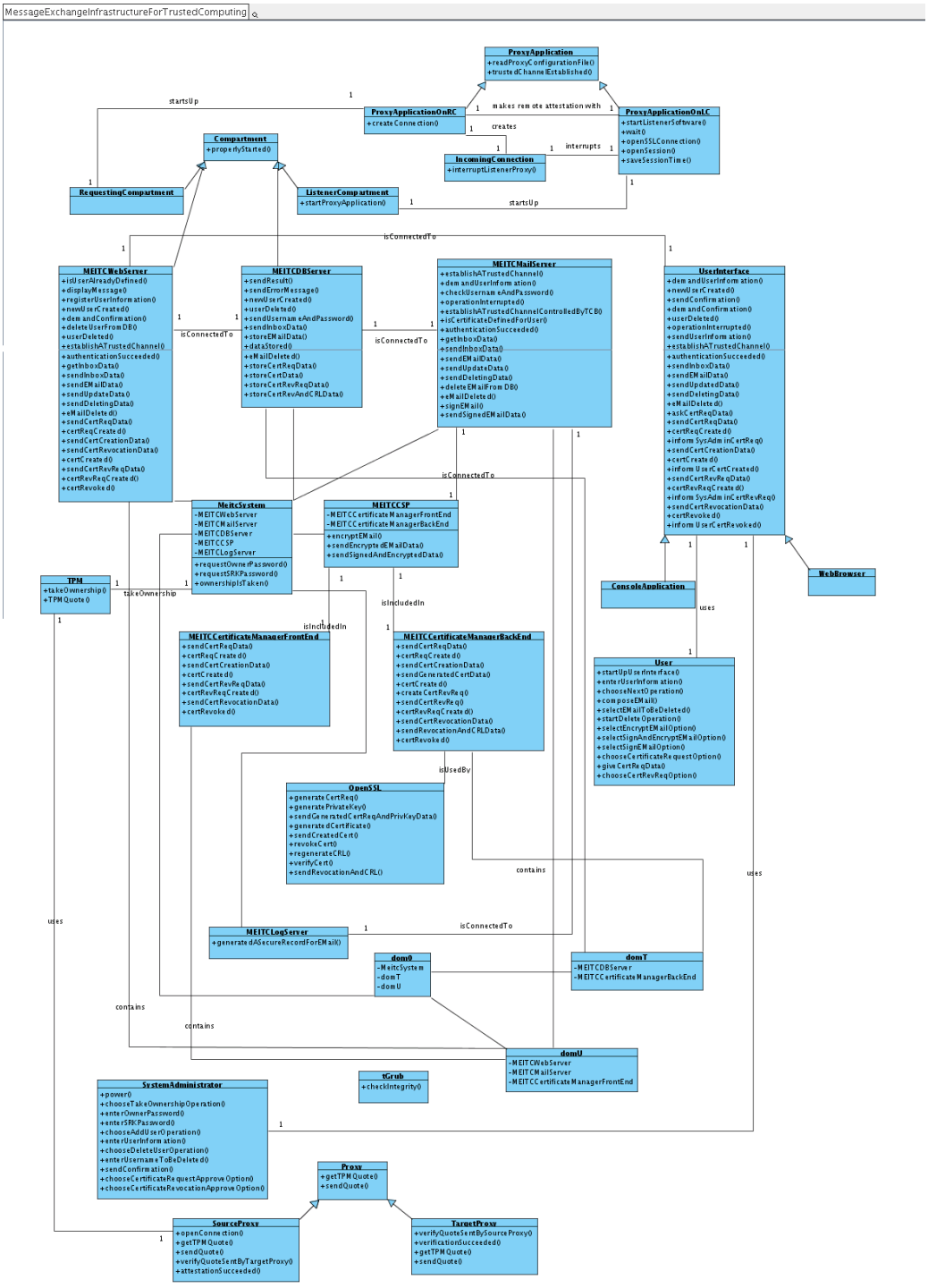


Figure 19:MEITC all classes diagram

8 Installation

This section contains information resolving issues to do with the installation of all parts of Message Exchange Infrastructure for Trusted Computing (MEITC) system.

8.1 Package structure

In MEITC the following software package structures will be used.

- RPM (Red Hat Package Manager)
- PISI (Packages Installed Successfully as Intended)
- TAR (Tape Archive)

PISI is the package management system of Pardus Linux distribution. It stores and handles the dependencies for the other packages and libraries in a database in XML format. Pardus repositories contain Apache, Dovecot, Cyrus-sasl, MySQL, Java Development Kit and Postfix PISI packages, which are necessary for proper running of MEITC system.

RPM is a powerful command line driven package management system capable of installing, uninstalling, verifying, querying, and updating computer software packages. RPM is a core component of many Linux distributions, such as Red Hat, Fedora, SUSE, Mandriva and many others. All MEITC software components will be available in RPM file format.

Tar archive is the most common means of distributing bundles of files. This format is traditionally produced by the Unix tar command. Necessary software packages for MEITC framework will also be prepared in tar file format. After installing a tar package, some modifications may be needed to run software packages properly.

8.2 File and directory structure

The following MEITC components file and directory structures are explained in this section:

- Web Server
- Mail Server
- Database Server

The symbolic name "$PREFIX" is used to refer to the full pathname of the release directory of installed software.

8.2.1 Web Server

For the message exchange infrastructure on the web server part, the following softwares should be up and running

- Apache Web Server: Version 2.0.3 or more
- Java 2 Standard Edition Runtime Environment (JRE)
- Apache Tomcat: Version 5.5
- Claros: Version 1.7

Apache web server is used for the incoming connections to the web server. Apache

will redirect the requests via JSP to the mail server, where a Postfix system is installed. Apache HTML files are located in the following folders:

- /var/www/localhost/htdocs (Pardus)
- /srv/www/htdocs/ (openSUSE)

Apache configuration file is located in the following locations:

- /etc/apache2/httpd.conf (Pardus)
- /usr/local/apache2/conf/httpd.conf (OpenSUSE)

By default, Apache uses port 80, and Tomcat uses 8080. These settings can be changed in the configuration files.

Apache Tomcat 5.5 requires the Java 2 Standard Edition Runtime Environment (JRE) version 5.0 or later. The following directories will be used for JRE.

- /opt/sun-jdk/bin: Executables for all the development tools contained in the JDK. The $PATH environment variable will contain an entry for this directory.
- /opt/sun-jdk/demo: Examples, with source code that shows how to program for the Java platform.
- /opt/sun-jdk/include: C-language header files that support native-code programming using the Java Native Interface and the Java Virtual Machine Debugger Interface.
- /opt/sun-jdk/jre: Root directory of the Java runtime environment used by the JDK development tools.
- /opt/sun-jdk/lib: Files used by the development tools. Includes tools.jar, which contains non-core classes for support of the tools and utilities in the JDK.
- /opt/sun-jdk/man : Contains man pages for the JDK tools.

These are some of the key apache tomcat directories

- $PREFIX/bin – Startup (startup.sh), shutdown (shutdown.sh) and other scripts.
- $PREFIX/conf - Server configuration files (including server.xml).
- $PREFIX/logs - Log and output files
- $PREFIX/shared - For classes and resources that must be shared across all web applications
- $PREFIX/webapps - Automatically loaded web applications
- $PREFIX/work - Temporary working directories for web applications
- /tmp - Directory used by the JVM for temporary files (java.io.tmpdir)

Claros files are located in $PREFIX/webapps/claros. Claros contains the following directories:

- `$PREFIX/webapps/claros/WEB-INF/config/config.xml` file can be used to change main configurations. PREFIX is the real path to the Tomcat installation.

8.2.2 Mail Server

The mail server is used to handle the incoming and outgoing mails. Mail server runs postfix and dovecot to answer POP3/IMAP connections and contains following software:

- Postfix
- Dovecot
- Cyrus-sasl

Postfix files are located in /etc/postfix directory. In this directory, settings for MySQL are defined in order to store users' mailboxes properly in database. /etc/postfix/main.cf file is used for main Postfix configuration.

Dovecot is used for handling POP3/IMAP connections for Postfix. The incoming connections and requests for connecting users' mailboxes are redirected to cyrus-sasl for encyrpted connection. Dovecot's files is located in the following folder:

- /etc/dovecot

/etc/dovecot/dovecot.cf file is used for configuration.

8.2.3 Database Server

Database server holds the users' mailboxes and informations. It contains following software packages.

- Mysql

MySQL’s files is located in the following folders:

- /etc/mysql : my.cf and mysqlaccess.cf is used for configuration.
- /var/log/mysql/ : Log files for mysql which is mysql.err, mysql.log and mysqld.err is located in this path.

9 Graphical user interface

9.1 Webmail GUI

9.1.1 User Login Screen

Figure 20:User login screen

9.1.2 Inbox Screen

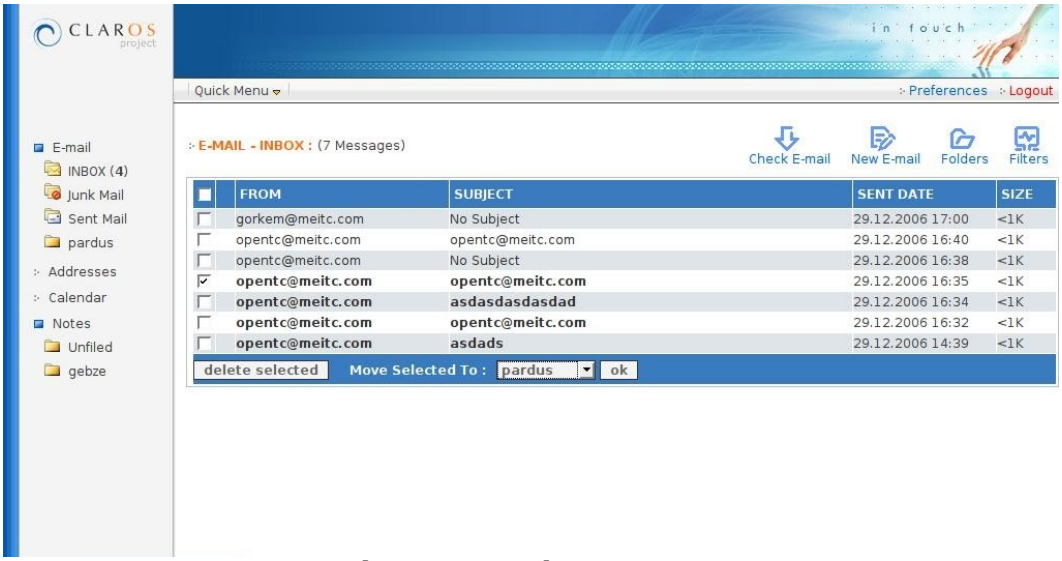


Figure 21:Inbox screen

9.1.3 Compose Email Screen

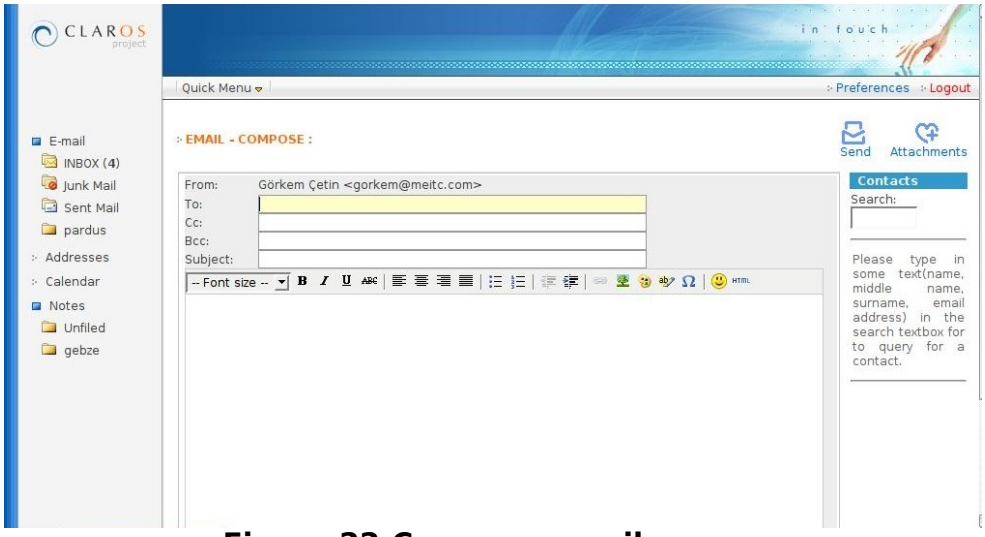


Figure 22:Compose email screen

9.2 Admin GUI

9.2.1 Admin Login screen

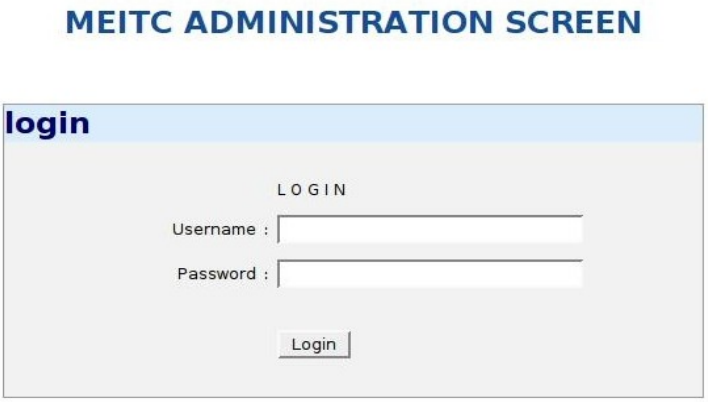


Figure 23:Admin Login screen

9.2.2 Main menu screen

Figure 24:Main menu screen

9.2.3 User list screen

Figure 25:User list screen

9.2.4 Add user screen

Figure 26:Add user screen

9.2.5 Delete user list screen

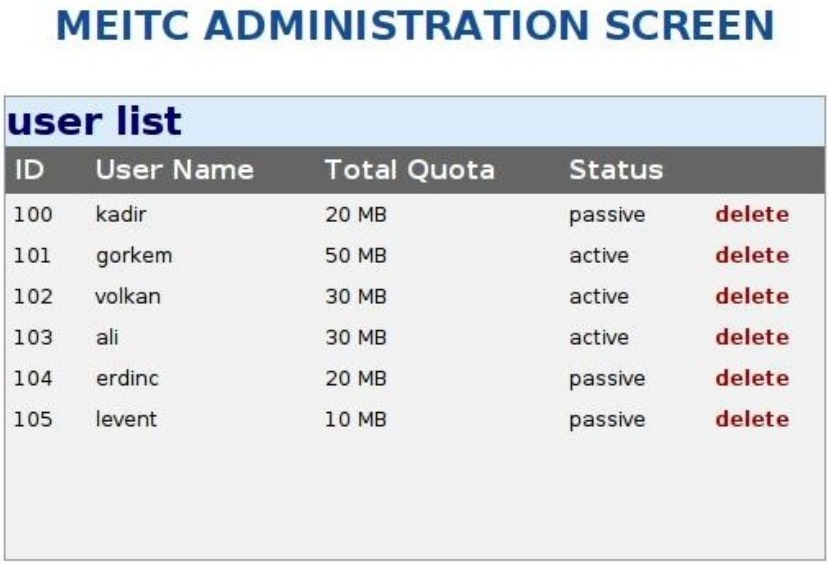


Figure 27:Delete user list screen

9.2.6 Delete user screen

Figure 28:Delete user screen

9.2.7 Reset user list screen

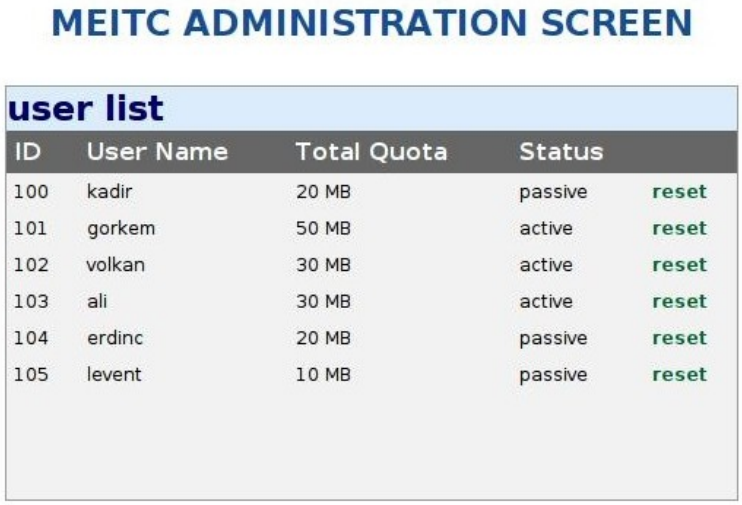


Figure 29:Reset user list screen

9.2.8 Reset user screen

Figure 30:Reset user screen

9.2.9 Services screen

Figure 31:Services screen

SWP06.c: WYSIWYS high level requirements specification

Project number	IST- 027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Internal deliverable

Deliverable reference number	IST-027635/D06c.1/FINAL 1.00
Deliverable title	WYSIWYS high level requirements specification
WP contributing to the deliverable	WP6
Due date	Oct 2007 - M24
Actual submission date	Nov 2007

Responsible Organisation	Politecnico di Torino
Authors	Gianluca Ramunno, Marco Vallini (POL)
Abstract	WYSIWYS is a functional and security requirement for electronic signatures, especially when used in legal contexts. This document consists in a high level requirements specification for a WYSIWYS application designed for OpenTC security architecture.
Keywords	

Dissemination level	Public
Revision	FINAL 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

List of figures

List of Tables

1 Motivation and problem description

“What You See Is What You Sign” (WYSIWYS) is a functional and security requirement for electronic signatures, especially when used in legal contexts (e.g. the European Directive 1999/93/EC on electronic signatures). To guarantee the trustworthiness of the content displayed and being signed, there is the need to guarantee a trusted path from the signing (or verifying) application to the user and in the opposite direction. Many past and present solutions that claim to be WYSIWYS compliant, in reality they are not. In fact they do not protect against the Trojan software or “malware” that can act on either the document image displayed to the user or the user’s input to activate the signing device operations. This is caused by the insecure architecture of the I/O subsystems integrated within the current monolithic Operating Systems.

Therefore the design of a WYSIWYS application must also take into account the underlying architecture in order to guarantee the actual trustworthiness of the application. In particular trusted input/output paths between the application and the user must be must be in place in order to guarantee the correct binding between the document presentation and the data actually signed or verified.

The security properties and services provided by OpenTC architecture can be used as foundation for a WYSIWYS application; enabling features from OpenTC are the trusted GUI, the assurance about the integrity of the security architecture and of the application. Moreover memory isolation through virtualisation and information flow control policies allow designing the WYSIWYS application in a modular fashion with a strong confinement of components with different levels of requirements for strength.

Another relevant aspect is the correctness of the document presentation. Given the complexity of the current document formats, designing and implementing trustworthy viewers solely for the purpose of a secure electronic signature doesn't match the market requirements, making this infeasible in practice. However a pragmatic approach can be used to go in the right direction for achieving this requirement: standard applications used to produce the documents being signed can be used as “trusted viewers” provided that they are properly configured to avoid hidden content, and dynamic content depending on the platform configuration or on the time when the document is presented.

This document includes a high level requirement specification for a reference architecture of an application for signing and verifying electronic documents that satisfies the WYSIWYS requirement.

2 Security Environment

This section describes the security aspects of the environment in which the product is intended to be used and the manner in which it is expected to be employed.

2.1 Assumptions

A description of assumptions shall describe the security aspects of the environment in which the Target of Evaluation (TOE) will be used or is intended to be used. This shall include the following:

- *information about the intended usage of the TOE, including such aspects as the intended application, potential asset value, and possible limitations of use; and*
- *information about the environment of use of the TOE, including physical, personnel, and connectivity aspects.*

/A 10/ Trusted Administrator

The security administrator of the system is non-malicious.

/A 20/ Correct hardware

The underlying hardware (e.g., CPU, devices, TPM, ...) does not contain backdoors, is non-malicious and behaves as specified.

/A 30/ No Physical attacks

Physical attacks against the underlying hardware platform do not happen.

/A 40/ TOE Binding

The IT-environment offers a mechanism that allows the TOE (WYSIWYS application) to store information and data like signing keys such that it cannot be accessed by another TOE configuration. Example mechanisms are the sealing function offered by a TPM as specified by the TCG in combination with an authenticated bootstrap architecture, or a tamper-resistant storage in combination with a secure bootstrap architecture.

/A 50/ No man-in-the-middle attack

A physical attack that relays the whole communication between a local user and the I/O devices to another device does not happen.

/A 60/ Trusted video path

The architecture underlying TOE provides a reliable and secure video output path.

/A 70/ Trusted input paths

The architecture underlying TOE provides reliable and secure paths for input devices

(keyboard, mouse, etc.).

/A 80/ Trusted path to cryptographic devices

The architecture underlying TOE provides a reliable and secure path to signing devices.

/A 90/ CRTM, TPM, boot loader, VMM and basic security services are trustworthy

The architecture underlying TOE, namely Core Root of Trust for Measurement (CRTM), TPM, boot loader, Virtual Machine Monitor (VMM) and services providing security features behave as expected. All of them are referred to as Trusted Computing Base (TCB) hereinafter.

/A 100/ TCB guarantees memory isolation between VMs

The TCB guarantees memory isolation between Virtual Machines (VMs) also called compartments.

/A 110/ TCB is able to enforce security policies for information flow control

The TCB can enforce security policies for information flow control between compartments: it can guarantees authenticity, integrity and confidentiality of communication channels among compartment.

/A 120/ TCB prevents exploits and replay attacks

The TCB is designed to prevent exploits of uncritical applications to gain access to security sensitive information and replay attacks, namely resetting the state of an application by replaying an older state.

/A 130/ TCB provides secure installation services for TOE

TCB provides installation services for all security critical applications like TOE.

/A 140/ Integrity of TOE is guaranteed by TCB

The TCB guarantees the integrity of TOE: either preventing TOE from running if it compromised or allowing TOE to be started but alerting the user about TOE being compromised.

2.2 Threats

A description of threats shall include all threats to the assets against which specific protection within the TOE or its environment is required. Note that not all possible threats that might be encountered in the environment need to be listed, only those which are relevant for secure TOE operation.

A threat shall be described in terms of an identified threat agent, the attack, and the asset that is the subject of the attack. Threat agents should be described by addressing aspects such as expertise, available resources, and motivation. Attacks should be described by addressing aspects such as attack methods, any vulnerabilities exploited, and opportunity.

If security objectives are derived from only organizational security policies and assumptions, then the description of threats may be omitted.

/T 10/ Trojan Horse

An adversary may try to get access to sensitive information by deceiving Administrators or Users such that a application under control of the adversary claims to be the TOE.

/T 20/ Unauthorised User

An unauthorised user may use TOE to read or modify information owned by another user.

/T 30/ Unauthorised Administrator

An unauthorised user may use a management functionality of the TOE to grant itself access to sensitive information.

/T 40/ Unauthorised Data Access

An unauthorised application may read or manipulate user information persistently stored by TOE.

/T 50/ Denial of Service

An adversary may try to prevent that authorised users can use the TOE by denial of service attacks against the TCB or the TOE itself.

/T 60/ Document replacement when displayed

A malicious application may try to replace the document being displayed to fool the user.

/T 70/ Document replacement when being signed

A malicious application may try to replace the document being signed with another one while keeping displayed the document selected by the user.

/T 80/ Incorrect document visualisation by output device

The output device may be not able to correctly represent all document details, e.g. due to screen resolution or output device size not enough for a correct representation or a limited set of available colours.

/T 90/ Misinterpretation of document format

The format of the document to be signed or verified may be wrongly interpreted by the viewer.

/T 100/ Dynamic code embedded in the document

The document may include dynamic code (i.e. macros) which can, without invalidating the signature, modify the document visualisation if different platforms are used or the

document is displayed at different times (e.g. signature or verification time).

/T 110/ Hidden content

The document may include hidden content being signed (e.g. text in the same colour as the background) without the user being able to notice it.

3 Functional Requirements (Use Case Model)

3.1 Goal

The goal is designing an application for signing and verifying the electronic documents that meets the WYSIWYS requirement. To achieve this goal, the design is based on OpenTC, a security architecture built on top of Trusted Computing and virtualisation technologies. The application performs the following operations: displaying the document to be signed and electronically signing the document, displaying an already signed document and verifying the electronic signature.

3.2 Target Groups

Defines the users/other components that wish to use the product.

- Home user (Single-user platform at home)
- Employee (Multi-user platform in enterprise environment)

3.3 Roles and Actors

In this section we define different roles and actors important for the use case model. Actors are parties outside the system that interact with the system; an actor can be a class of users, roles users can play, or other systems. Note that, depending on the use case, some parties or actors may not be involved.

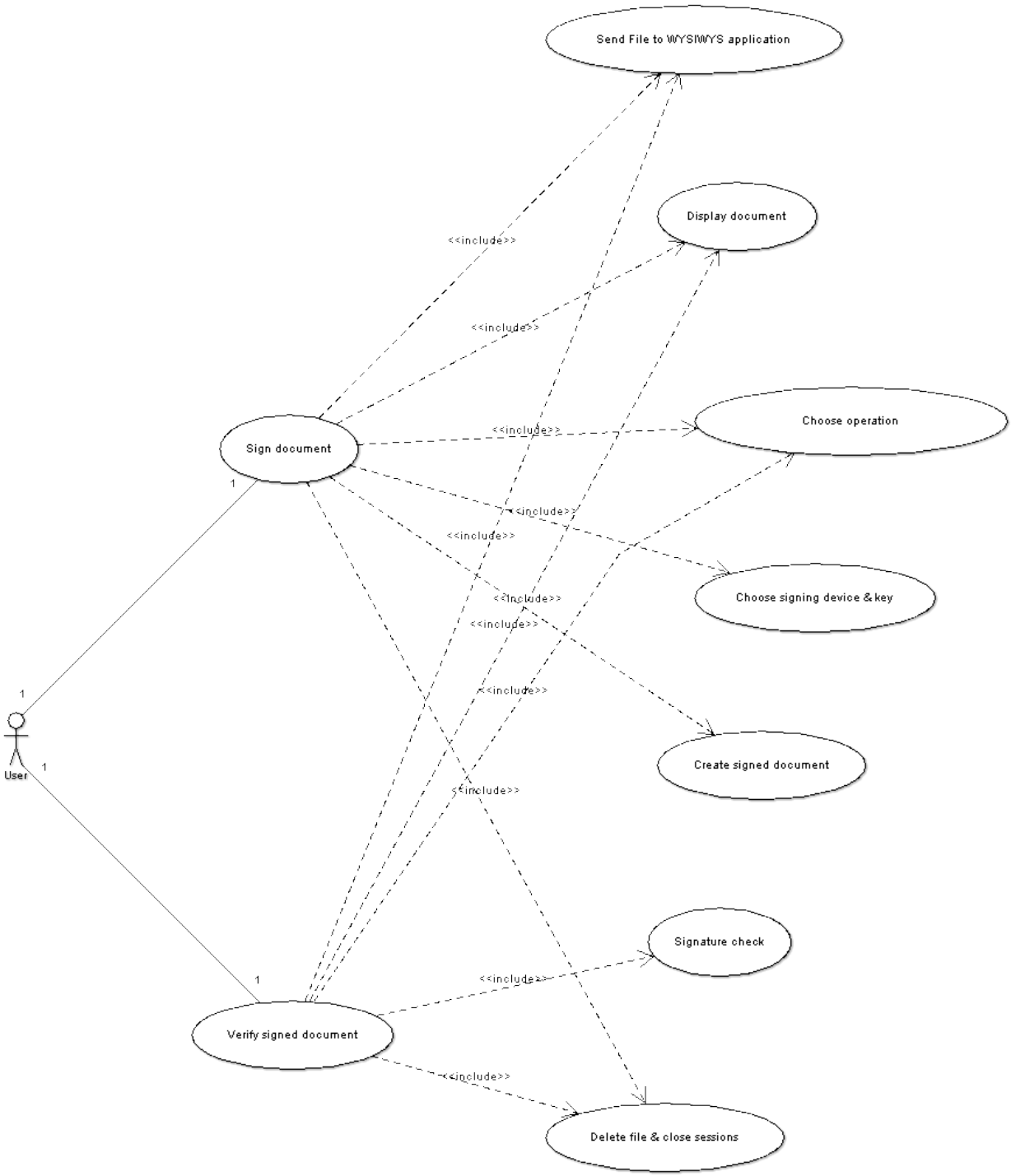
User: The user of a computing platform is an entity interacting with the platform under the platform's security policy. Examples are employees using enterprise-owned hardware.

3.4 Overview

The user can use WYSIWYS application to perform two main operations:

1. signing a document
2. verifying a signed document

Figure 1: Use cases diagram



3.5 Use Cases (Detailed Description)

Each use case focuses on describing how to achieve a single business goal or task. From a traditional software engineering perspective a use case describes just one feature of the system. For most software projects this means that

multiple, perhaps dozens, of use cases are needed to fully specify the new system. The degree of formality of a particular software project and the stage of the project will influence the level of detail required in each use case.

A use case defines the interactions between external actors and the system under consideration to accomplish a business goal.

Use cases treat the system as a "black box", and the interactions with the system, including system responses, are perceived as such from outside the system. This is a deliberate policy, because it simplifies the description of requirements, and avoids the trap of making assumptions about how this functionality will be accomplished.

A use case should:

- *describe a business task to serve a business goal*
- *have no implementation-specific language*
- *be at the appropriate level of detail*
- *be short enough to implement by one software developer in a single release.*

3.5.1 Sign a document

Use case unique ID	/UC 10/
Title	Sign document
Short description/purpose(s)	The user wants to sign a document
Actors	User
Includes	/UC 30/ Send file to WYSIWYS /UC 40/ Display document /UC 50/ Choose operation /UC 60/ Choose signing device & key /UC 70/ Create signed document /UC 90/ Delete files & close sessions
Preconditions	WYSIWYS application is running
Postcondition	The user receives back the signed document
Normal Flow	1. Send file to WYSIWYS application /UC 30/ 2. Display document /UC 40/ 3. Choose operation (sign) /UC 50/ 4. Choose signing device & key /UC 60/ 5. Create signed document /UC 70/ 6. Signature verification /UC 80/ 7. Delete file & close sessions /UC 90/

3.5.2 Verify a signed document

Use case unique ID	/UC 20/
Title	Verify signed document
Short description/purpose(s)	The user wants to verify a signature applied to a document
Actors	User
Includes	/UC 30/ Send file to WYSIWYS application /UC 40/ Display document /UC 50/ Choose operation /UC 80/ Signature verification /UC 90/ Delete file & close sessions
Preconditions	WYSIWYS application is running
Postcondition	The user receives the result of signature verification
Normal Flow	1. Send file to WYSIWYS application /UC 30/ 2. Display document /UC 40/ 3. Choose operation (verify) /UC 50/ 4. Signature verification /UC 80/ 5. Delete file & sessions /UC 90/

3.5.3 Basic operations

Use case unique ID	/UC 30/
Title	Send file to WYSIWYS application
Short description/purpose(s)	The user sends file to WYSIWYS application
Actors	User
Preconditions	WYSIWYS application is running
Postcondition	The document is loaded into WYSIWYS application
Normal Flow	6. The user sends the application the document's file using a proper command 7. The application saves the document internally 8. The application activates a trusted interface for user interaction

Use case unique ID	/UC 40/
Title	Display document
Short description/purpose(s)	The application shows the document and guarantees a trustworthy display
Preconditions	/UC 30/
Postcondition	The document is shown to the user
Normal Flow	1. The application activates the correct viewer for the document format 2. The application loads the document file from an internal storage 3. The document is displayed to the user

Use case unique ID	/UC 50/
Title	Choose operation
Short description/purpose(s)	The user chooses to sign or verify a document
Actors	User
Preconditions	/UC 40/
Postcondition	The user has chosen the operation to be executed
Normal Flow	1. The user is required to choose one operation 2. The user decides to sign or verify the document 3. The application takes charge of user's choice

Use case unique ID	/UC 60/
Title	Choose signing device & key
Short description/purpose(s)	The user selects the signing device and key
Actors	User
Preconditions	/UC 50/
Postcondition	The signing device and the key are chosen
Normal Flow	1. The application shows to the user the list of available signing devices 2. The user chooses the signing device 3. The application shows to the user the list of available keys 4. The user chooses the signing key

Use case unique ID	/UC 70/
Title	Create signed document
Short description/purpose(s)	Create the file containing the signed document
Preconditions	/UC 60/
Postcondition	The user receives the signed document
Normal Flow	1. The application loads the document to be signed from the internal storage 2. The selected signing device generates the electronic signature over document file using the selected key 3. The application creates the file containing the document and the signature 4. The application returns to the user the signed document

Use case unique ID	/UC 80/
Title	Signature verification
Short description/purpose(s)	The application verifies the correctness of the electronic signature over the document
Preconditions	/UC 50/
Postcondition	The user receives the result of verification
Normal Flow	1. The application loads the signed document to be verified 2. The application actually verifies correctness of the signature 3. The application returns the result of the verification to the user

Use case unique ID	/UC 90/
Title	Delete file & close sessions
Short description/purpose(s)	All sessions are destroyed and the file internally saved is deleted
Preconditions	/UC 70/ or /UC 80/
Postcondition	The application returned to its initial state
Normal Flow	5. The application deletes the file from the internal storage 6. The application closes all sessions

4 Security Objectives & Security Requirements

4.1 Security Objectives

The security objectives shall address all of the security environment aspects identified. The security objectives shall reflect the stated intent and shall be suitable to counter all identified threats and cover all identified organizational security policies and assumptions. A threat may be countered by one or more objectives for the product, one or more objectives for the environment, or a combination of these.

/SO 10/ Separability

The use of different security-critical TOE components based on the OpenTC security architecture has to be at least as secure as the execution of the same applications on physically separated computing platforms connected via network.

/SO 20/ No unauthorized use of TOE components

Unauthorized entities must not be able to arbitrarily execute TOE components.

/SO 30/ Visual identification of TOE User Interface

The user must be able to reliably identify the User Interface of TOE.

/SO 40/ Correct visualisation of the document

TOE must correctly visualise the document being signed or verified.

/SO 50/ Binding between visualisation and signature/verification operations

TOE must actually sign or verify the document being displayed to the user.

4.2 Security Requirements

This part of the requirement specification defines the security requirements that have to be satisfied by the product. The statements shall define the functional and assurance security requirements that the product and the supporting evidence for its evaluation need to satisfy in order to meet the security objectives.

/SR 10/ No communication among TOE components and external parties

Security policies should be enforced to guarantee that TOE components cannot interact with external parties.

/SR 20/ Information flow

Security policies should be enforced to guarantee that information flow is only possible among TOE components. Primarily, eves dropping on another, non-cooperating compartment must be foiled.

/SR 30/ Integrity of document to be signed or verified

TOE should guarantee that the displayed document cannot be corrupted while being signed or verified.

/SR 40/ Trusted WORM Storage

The TOE should use a trusted storage Write Once Read Many (WORM) for storing documents to be signed or verified and used by TOE components during all intermediate operations.

/SR 50/ Trusted RW Storage

The TOE should use a trusted storage Read/Write for temporary files during operations.

5 Supplementary Requirements

Obligatory criteria, mandatory for successful completion.

5.1 Preconditions

Requirements that have to be fulfilled already, because they were needed for the development process.

/PR 100/ Trusted Computing Base

The TOE is build upon OpenTC, a security architecture for Trusted Computing Base.

/PR 200/ Reliable document viewer

The TOE should use at least one application that is considered reliable as viewer for one specific document format (e.g. OpenDocument).

5.2 Required Criteria

Mandatory criteria, that are obligatory for successful completion.

/RC 10/ Xen support

The realization of the use cases should be based on a Xen-based architecture.

/RC 20/ Single-user support

The TOE should support at least one user.

/RC 30/ Cryptographic devices

The TOE should support all common cryptographic devices - hardware and software - through standard interfaces (particularly PKCS#11).

/RC 40/ Document formats

The TOE should support virtually any type of document format via plug-in based architecture for document viewers.

5.3 Desired Criteria

Optional criteria, that are not mandatory for successful completion.

/DC 10/ Multi-user support

The security architecture should be able to handle multiple users.

/DC 20/ L4 support

The realization of the use cases should be based on an L4-based architecture.

5.4 Distinguishing Criteria

What our product does not provide.

5.5 Execution Environment

This section specifies software and hardware the user requires at least to run our product successfully.

5.5.1 Software

- Standard Linux 2.6.x distribution
- Xenolinux 3.1.x (Linux 2.6.x running on top of Xen 3.1.x hypervisor)
- OpenOffice 2.3 or higher
- (optional) L4-Linux (Linux 2.6.x running on top of Fiasco, L4V2 µ-kernel)

5.5.2 Hardware

- Intel LT/VT or AMD-V Platform
- TPM 1.2 Platform

5.6 Development Environment

This section specifies hard- and software that developers need at least to implement the product successfully.

5.6.1 Software

- Linux 2.6.x
- gcc 4.2.x
- eclipse-3.1
- OpenOffice 2.3 or higher

5.6.2 Hardware

- Intel LT/VT or AMD-V Platform
- TPM 1.2 Platform

6 High-Level Software Architecture

6.1 Introduction

This section contains some views of a high-level software architecture for a WYSIWYS application. In particular the granularity of the views is at package level; each package includes a group of components that share the same level of strength for security requirements. Such groups can be actually compartmented using a different virtual machines. To show the interactions among those virtual machine sequence diagrams are used, thus overloading their semantic, since they are normally used to show interactions among objects.

6.2 Logical views

6.2.1 Packages

In figure 2 the package diagram shows the 'use' relationships among different packages.

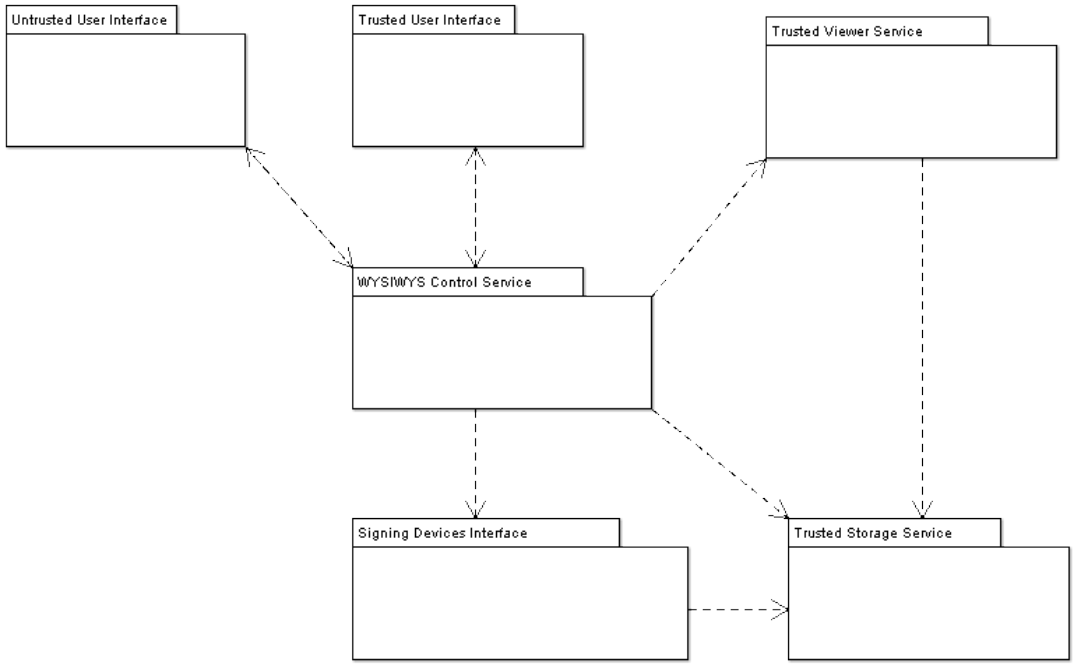


Figure 2: Package diagram

Untrusted User Interface

It is the standard interface provided to the user by the environment for daily operations (like browsing the Internet, reading e-mails and writing documents); it allows the user to start WYSIWYS application and to choose the document file to sign or verify.

Trusted User Interface

It is part of WYSIWYS application and it allows the user to interact with *WYSIWYS*

Control Service to choose the operation (sign or verify) to be executed and related options.

WYSIWYS Control Service

It implements the application logic and controls all packages. It receives the document file to be signed or verified from *Untrusted User Interface* and it manages the interactions between all packages.

Signing Devices Interface

It exposes a simple API to give access to the signing devices. Different types of devices can be supported: software and hardware (commonly used smart-cards or TPM). Each user can use a (sub)set of all devices the platform makes available. Such devices hold the users' keys.

Trusted Viewer Service

It shows the document to be signed or verified. It guarantees a trustworthy visualisation using the correct viewer with regards to the document format.

Trusted Storage Service

It allows a trusted storage of the document file for all WYSIWYS operations. It implements a WORM storage (Write Once Read Many). Only who write a File can delete it while every package can read it.

6.2.2 Use case realisation

Table 1 lists required packages for the realisation of each use case for the basic operations. Then a possible implementation of such use cases is described through the interaction of components grouped in packages via sequence diagrams.

Use Case	**Required packages**
/UC 30/ Send file to WYSIWYS application	Untrusted User Interface, Trusted User Interface, WYSIWYS Control Service, Trusted Storage Service
/UC 40/ Display document	WYSIWYS Control Service, Trusted Storage Service, Trusted Viewer Service
/UC 50/ Choose operation	Trusted User Interface, WYSIWYS Control Service
/UC 60/ Choose signing device & key	WYSIWYS Control Service, Signing Devices Interface, Trusted User Interface
/UC 70/ Create signed document	Untrusted User Interface, WYSIWYS Control Service, Trusted Storage Service, Signing Devices Interface
/UC 80/ Signature verification	WYSIWYS Control Service, Trusted User Interface, Signing Devices Interface, Trusted Storage Service
/UC 90/ Delete file & close sessions	Trusted User Interface, WYSIWYS Control Service, Trusted Storage Service, Trusted Viewer Service

Table 1: Packages required by use cases

/UC 30/ Send File to WYSIWYS application

The *User* selects the file to be signed or verified through the *Untrusted User Interface* that sends that file to *WYSIWYS Control Service.* The file is then sent to *Trusted Storage Service* that saves it in the secure Write Only Read Many storage. *Trusted Storage Service* returns a result about the correctness of the saving operation. Then *WYSIWYS Control Service* immediately activates the *Trusted User Interface* to interact with the *User*.

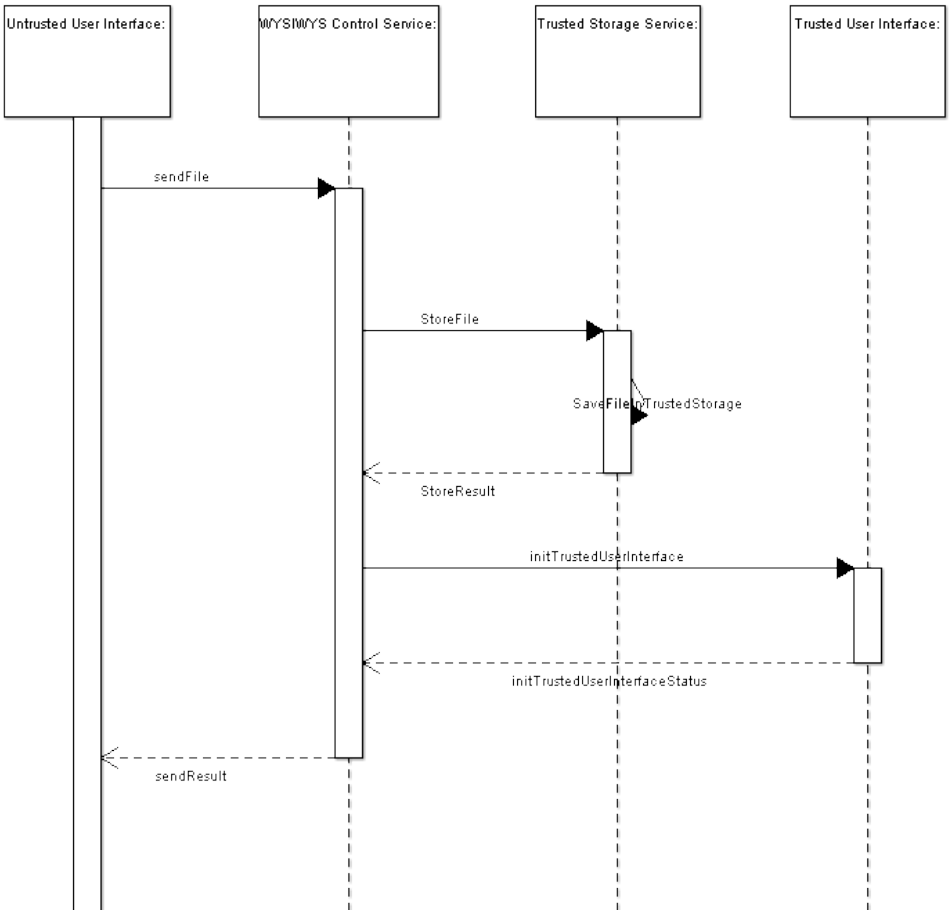


Figure 3: UC 30 sequence diagram

/UC 40/ Display document

WYSIWYS Control Service requests *Trusted Viewer Service* to show the document. *Trusted Viewer Service* loads the document file directly from *Trusted Storage Service*, then activates the proper viewer with regards to the file format and shows document. The *Trusted Viewer Service* returns a result about the correctness of the display operation.

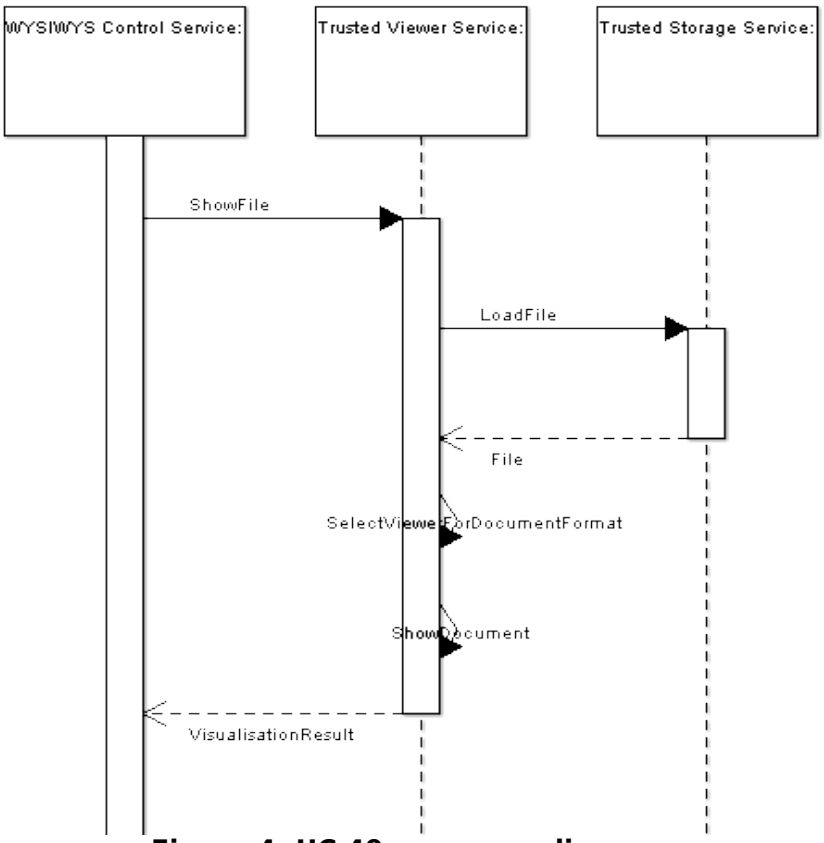


Figure 4: UC 40 sequence diagram

/UC 50/ Choose operation

WYSIWYS Control Service requests *Trusted User Interface* to show the list of allowed operations (sign or verify) to *User*. He/she selects the wanted operation and the choice is then taken in charge of by *WYSIWYS Control Service*.

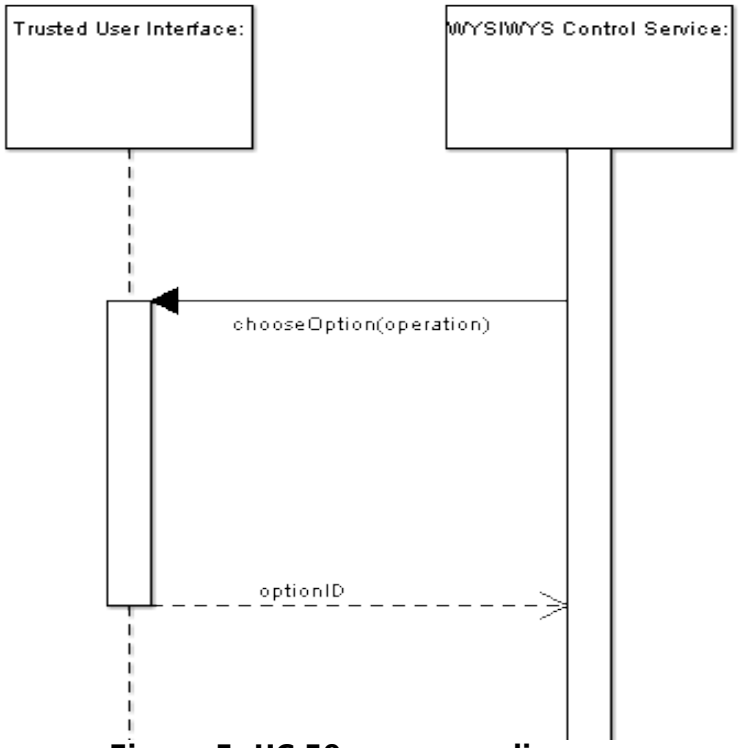


Figure 5: UC 50 sequence diagram

/UC 60/ Choose signing device & key

WYSIWYS Control Service has received the command to sign the document, so it requests *Signing Devices Interface* for the list of available signing devices for the *User*. Through *Trusted User Interface* the *User* chooses the signing device to be used. A similar sequence of operations is performed to allow the *User* to select the wanted signing key for the chosen device.

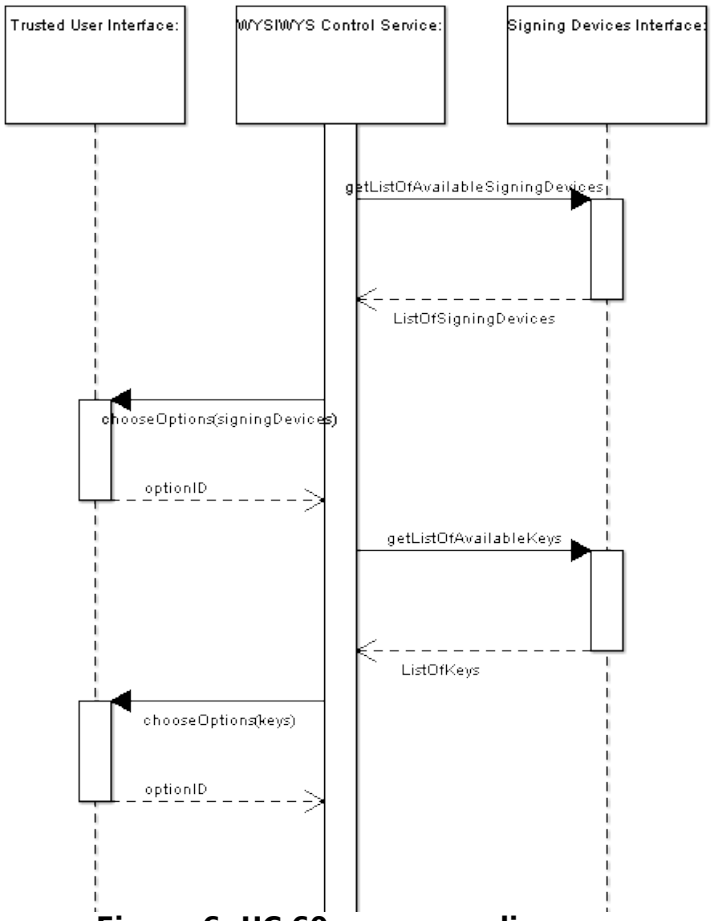


Figure 6: UC 60 sequence diagram

/UC 70/ Create signed document

WYSIWYS Control Service requests *Signing Devices Interface* to sign the document file. *Signing Devices Interface* directly loads the file from *Trusted Storage Service* and signs the document using the chosen device and key. Then the file just created is sent back to *WYSIWYS Control Service* that in turns returns the signed document to *User* through the *Untrusted User Interface*.

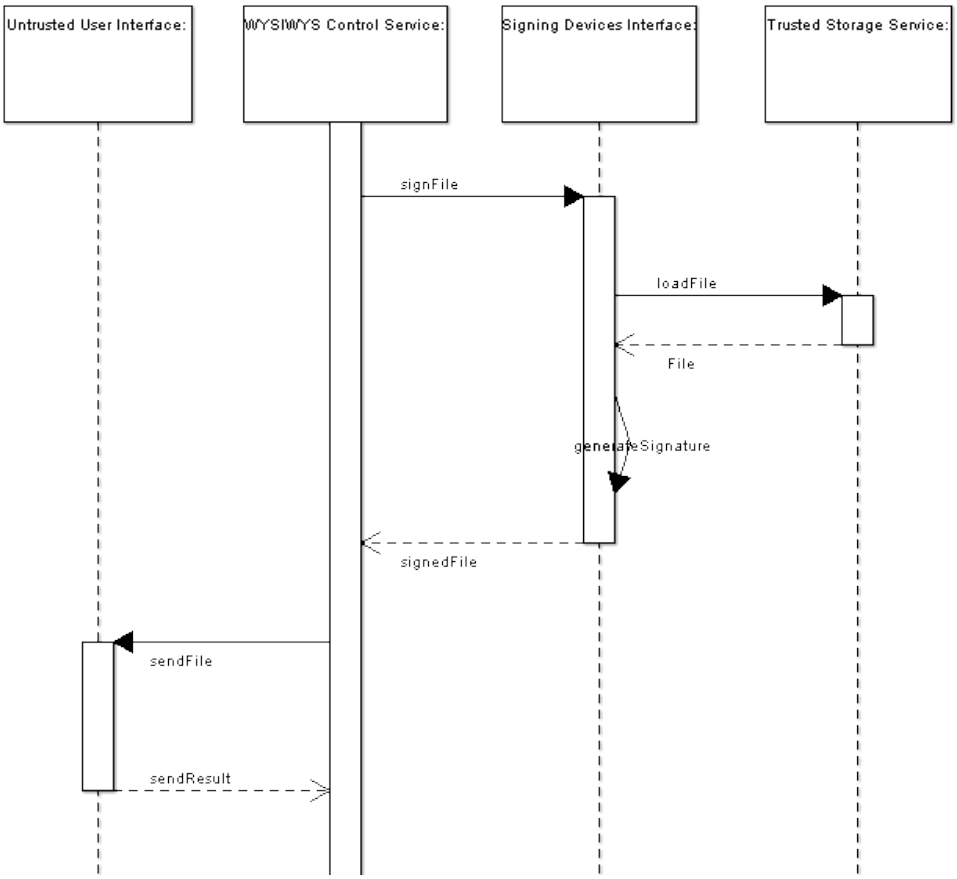


Figure 7: UC 70 sequence diagram

/UC 80/ Signature verification

WYSIWYS Control Service requests *Signing Devices Interface* to verify a signed document. *Signing Devices Interface* loads the file directly from *Trusted Storage Service* and verifies the signature. Then it returns the result of the verification to *WYSIWYS Control Service* that in turns returns the result to *User* through *Trusted User Interface*.

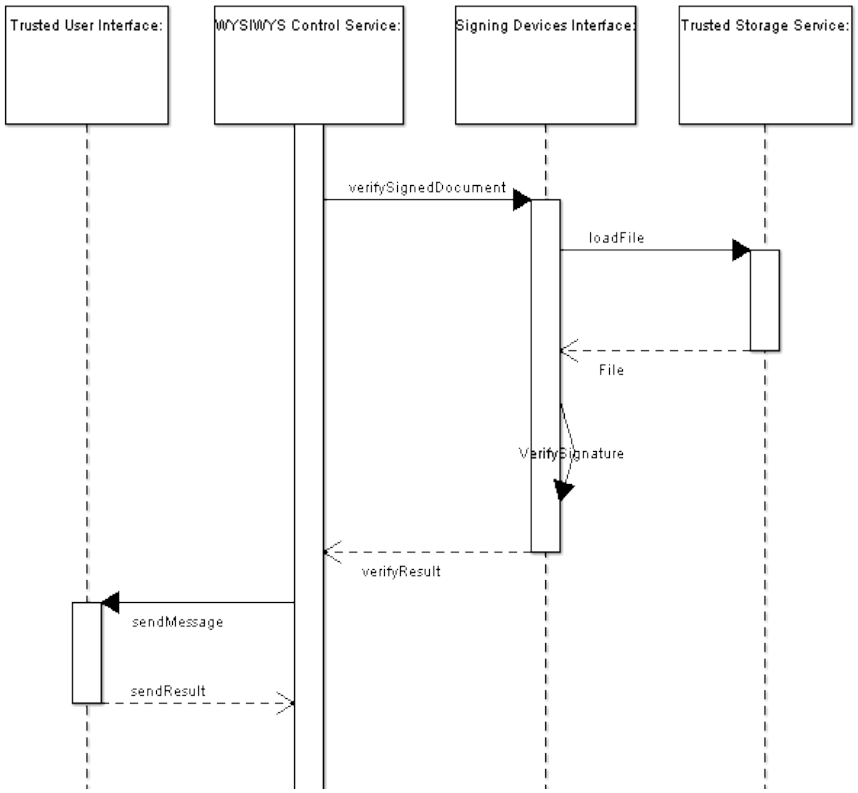


Figure 8: UC 80 sequence diagram

/UC 90/ Delete file & close sessions

WYSIWYS Control Service deletes from *Trusted Storage Service* the file previously loaded upon user's choice, then it requests *Trusted User Interface* and *Trusted Viewer Service* to close the relative session opened for the operation requested.

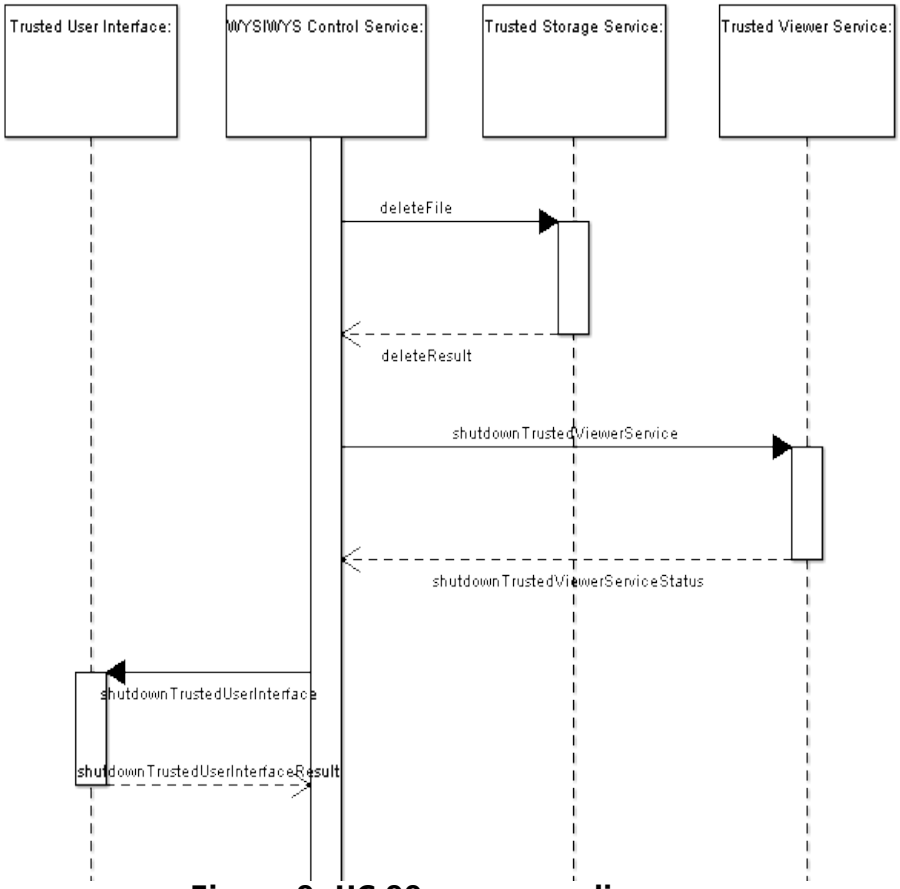


Figure 9: UC 90 sequence diagram

7 List of Abbreviations

Listing of term definitions and abbreviations used in the overview documents and architectural design specification (IT expressions and terms from the application domain).

Abbreviation	Explanation
CPU	Central Processing Unit
IT	Information Technology
RW	Read/Write
TC	Trusted Computing
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TOE	Target of Evaluation
TPM	Trusted Platform Module
VM	Virtual Machine
VMM	Virtual Machine Monitor
WORM	Write Once Read Many
WYSIWYS	What You See Is What You Sign

8 Acknowledgements

The authors want to thank Ahmad-Reza Sadeghi from Ruhr University Bochum and Chris Stüble from Sirrix AG for providing the template used for this document and ans example on how to use it. The authors want also to thank Francesco Sartorio for the substantial contribution to the requirements analysis.

D6e.4: Final MFA System Specification

D6e.5: Complete MFA System Prototype

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Internal Report

Deliverable reference number	IST-027635/D6e.4, D6e5/PUBLIC | 1.00
Deliverable title	Final MFA System Specification,MFA Concept Prototype
WP contributing to the deliverable	WP6
Due date	April 2007 M18, October 2007 M24
Actual submission date	November 13 2007

Responsible Organisation	INTEK
Authors	Irina Beliakova, Andrej Sokolov
Abstract	This document specifies MFA System which use trusted platform features for a secure, multifactor authentication to remote computers. The system allows to make remote logon by using TPM credentials and/or password. The document contains system architecture, components, their interactions, data structures ,s function calls and implementation of the MFA Concept prototype.
Keywords	PAM, TPM, Credential Manager, MFA

Dissemination level	Public
Revision	PUBLIC | 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

Illustration Index

1 Introduction

The TPM Multifactor Authentication (MFA) system is an application of the Trusted Computing technology and shows the benefits of such technology for ensuring that only a user who owns a registered platform equipped with a TPM may have access to the remote computer resources.

Multifactor authentication to remote server involves components executed at server and client computers. Client components register User and Client TPM Platform with the Remote server. Client components use the local TPM through TSS. The user can login to a remote server (services/application) when the User and Client Platform registration are completed.

The parameters of multifactor authentication system are controlled by authentication policies.

This document gives high level design specification of a MFA system architecture, components description, protocol of the components interaction, installation instructions and high level API.

2 Requirements breakdown

As long as an authorized system is used to access corporate resources, the entire infrastructure can be thought of as protected. Even if somebody's credentials have been stolen, the intruder will have to operate from a trusted corporate platform to gain the access to the resources. On the other hand even the authorized user may mistakenly try to gain the access from improperly configured system, such as home computer, untrusted device, and etc. - the platforms that by definition are outside the corporate control.

The multifactor authentication, including both user and client platform authentication covers up most of these threats. The access to the network is granted only if both elements - user and platform - are successfully authenticated.

For this to work, an installed and running TSS on the client platform is required. Implementation of Multifactor Authentication application expects the presence of an underlying trusted framework and requires the following services from it:

- Trusted Software Stack (TSS) for Linux according to TCG specification. TSS stack must support basic TCG functionality including Attestation Identities Keys (AIK) generation, TPM Platform Configuration Registers (PCRs) calculation, storing, and retrieval.
- Security services such as OpenSSL to provide secure communication protocol.
- PAM Authentication Framework

3 High level design specification

A MFA system could consist of several server and client computers connected with each other. Each of clients has a TPM module (Server may have a TPM, too). Servers provides to remote clients computer, services (SSH, FTP and so on) that could be accessed remotely after successful user credential authentication. During user authentication MFA system can check:

- remote client platform attestation identities key (AIK) or user key (UK),
- remote operating system (OS) and software set by platform configuration registers (PCRs).

MFA System is based on the PAM Framework. The Pluggable Authentication Modules (PAM) library is a generalized API for authentication-related services which allows a system administrator to add new authentication methods simply by installing new PAM modules and modifying authentication policies by editing special user policy configuration files.

One of the goals of MFA implementation is easy integration with different services that use pluggable authentication modules (PAM) for authentication without any changes in applications/service source code on the servers and clients.

3.1 MFA system architecture

The MFA architecture includes two component types: MFA Server and MFA Client.

The client and server components are installed by the system administrator. After the installation is completed the user has access to the client system and the system administrator has access to server computer.

These components can run on on a single Linux box with TSS and TPM as well as on top of the Open_TC framework. The MFA System Architecture is presented in the Figure 1.

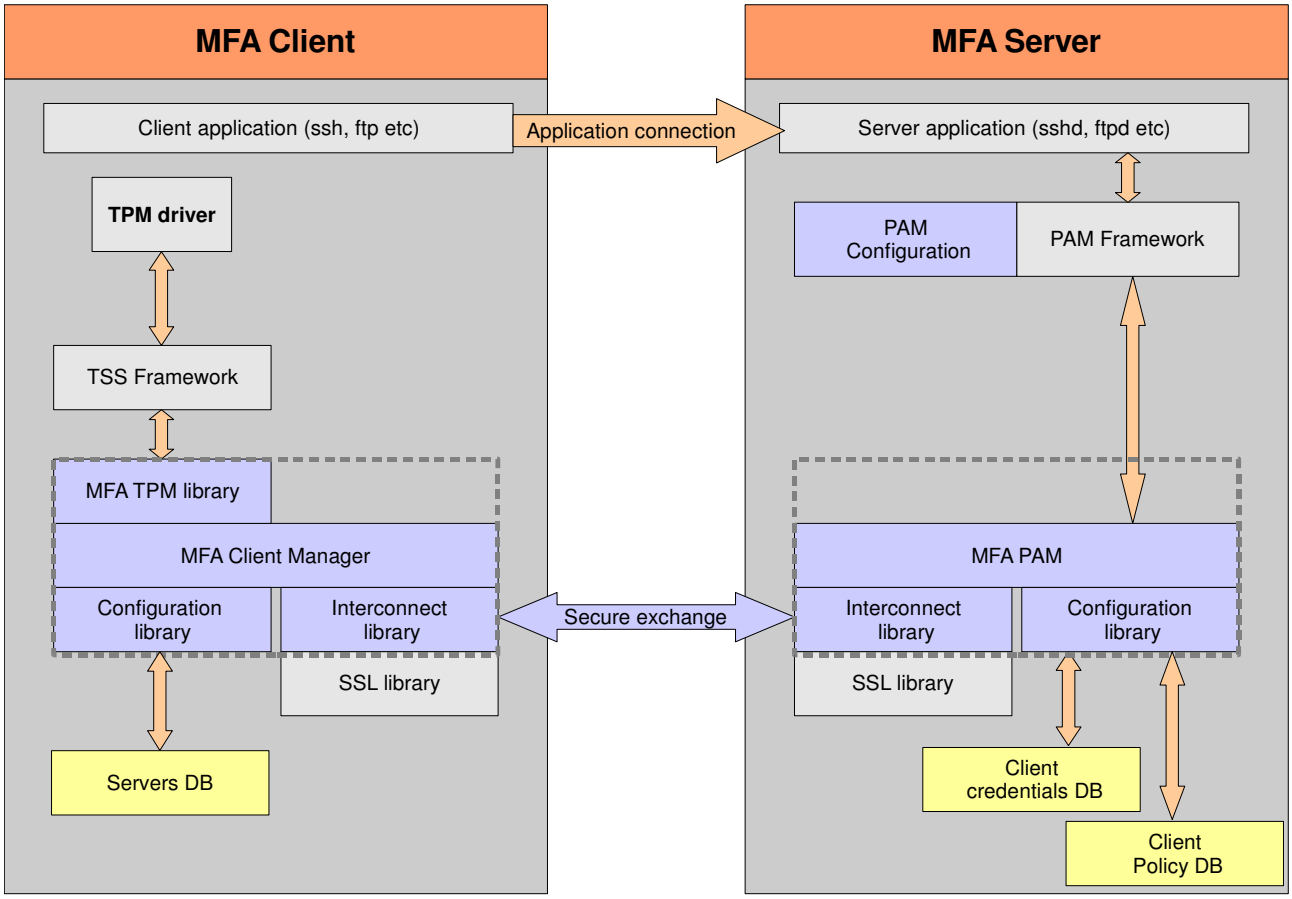


Figure 1: MFA System Architecture

MFA Server includes the following components:

- **Server application** – standard sshd, ftpd or other that realized required service and support PAM framework;
- **MFA PAM** – is a PAM standard module. It is called directly by service/application to authenticate users. The system uses the PAM possibilities to add multifactor authentication/registration to the services. This module communicates with MFA Client Manager;
- **PAM Configuration** – system file (one for each service) that describes the list and sequences of the actions that called from PAM Framework;
- **Configuration library** – library that include functions to control User and Platform Credentials, Policy Data Base;
- **Clients Credential and Policy DB** – database, that keep securely User and Platform credentials and policy. It is a set of files and directories;
- **Interconnect library** – library of functions for interconnection within PAM MFA module and MFA Client Manager;
- **External modules**:
    - TPM TSS
    - SSL library for work with keys and certificates.

MFA Client includes the following components:

- **Clients Application** – standard applications (ssh, ftp or etc.);
- **MFA Client manager** – a service that processes any request from the server, to provide the information about client credentials, certificates and any user platform data;
- **Configuration library** – library that include functions to control User Platform Credentials and Policies;
- **Servers DB** – database, that keep registered Servers key (certificates). It is a set of files;
- **Interconnect library** – library of functions for interconnection within PAM MFA module and MFA Client Manager;
- **External modules**:
    - TPM TSS to access to the TPM;
    - SSL library for work with keys and certificates.

Client components perform the following tasks:

- Process Server Requested actions to gather User TPM Credentials;
- Interact with TPM through TSS stack;
- Send data to the MFA PAM module on servers;
- Log MFA System activity.

Server components perform the following tasks:

- Authenticate and Register Client Platform & User;
- Keep Client Platform/User database;
- Control Credentials & Policy settings;
- Interact with MFA Client Manager on client machine.

3.2 Client Authentication Protocol

Client/server message flow of the client authentication process is presented on the Figure 2.

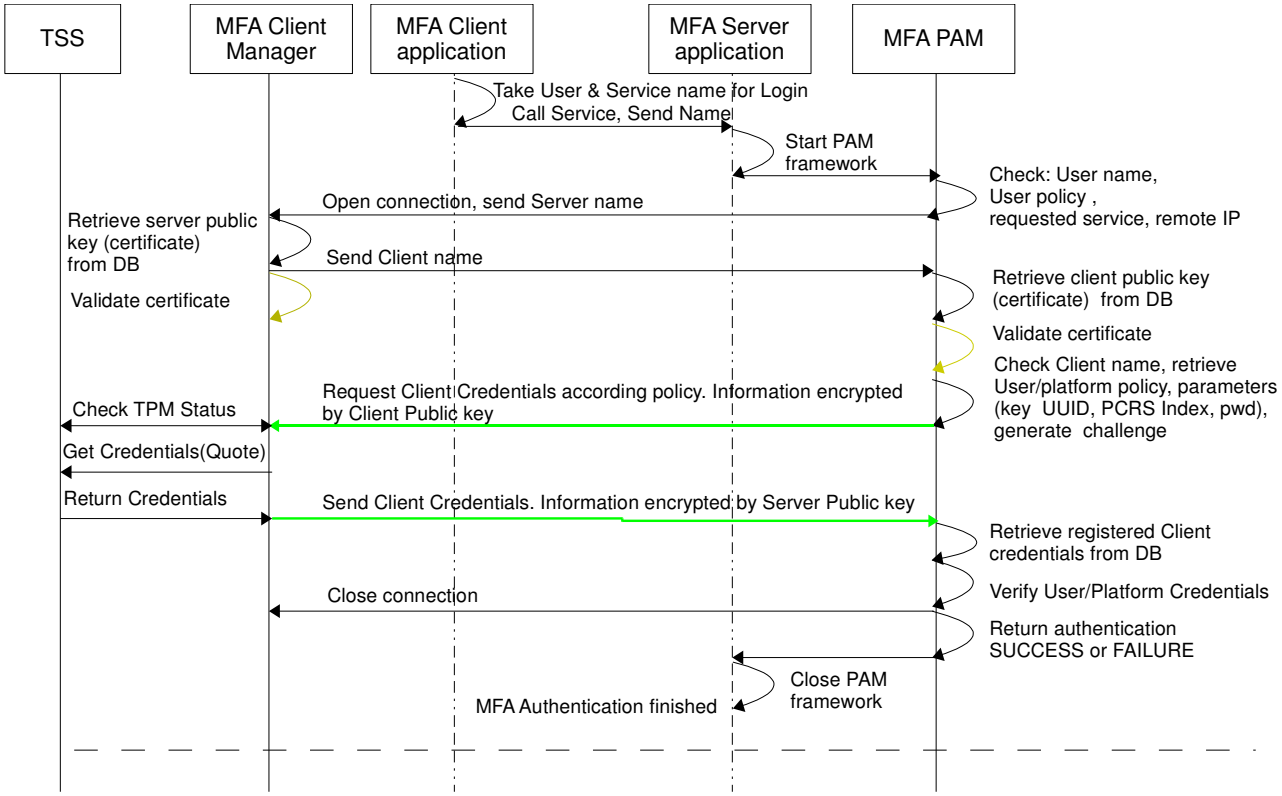


Figure 2: Client Authentication Protocol

MFA server acts as one of pluggable authentication module and usage of it by different applications can be configured in standard PAM configuration file.

When MFA PAM is started by application/service it does the next steps:

1. MFA PAM (Server side):
   - get call parameters: login name, application/service name, connection properties;
   - check that user has configured policy to access to the service;
   - connect to MFA Client Manager on remote client;
   - exchange by public keys (certificates) with Client;
   - check that access to the requested service for the user from this Client is allowed according configured policy;
   - send request for the client authentication platform and user credentials to

MFA Client Manager.

- MFA Client Manager (Client side):
- check that the remote Server is registered and enabled in the Client configuration (during registration process);
- create the client platform credentials according Server request policy;
- send encrypted credential information to the Server.

2. MFA PAM (Server side):
   - retrieve from Configuration registered reference platform credentials and compare with credentials received from client;
   - return PAM_SUCCESS or PAM_AUTH_ERR;
   - in case PAM_SUCCESS the user logon to the service/application is completed and service/application can be used by user on the Server;
   - close connection to MFA Client Manager.

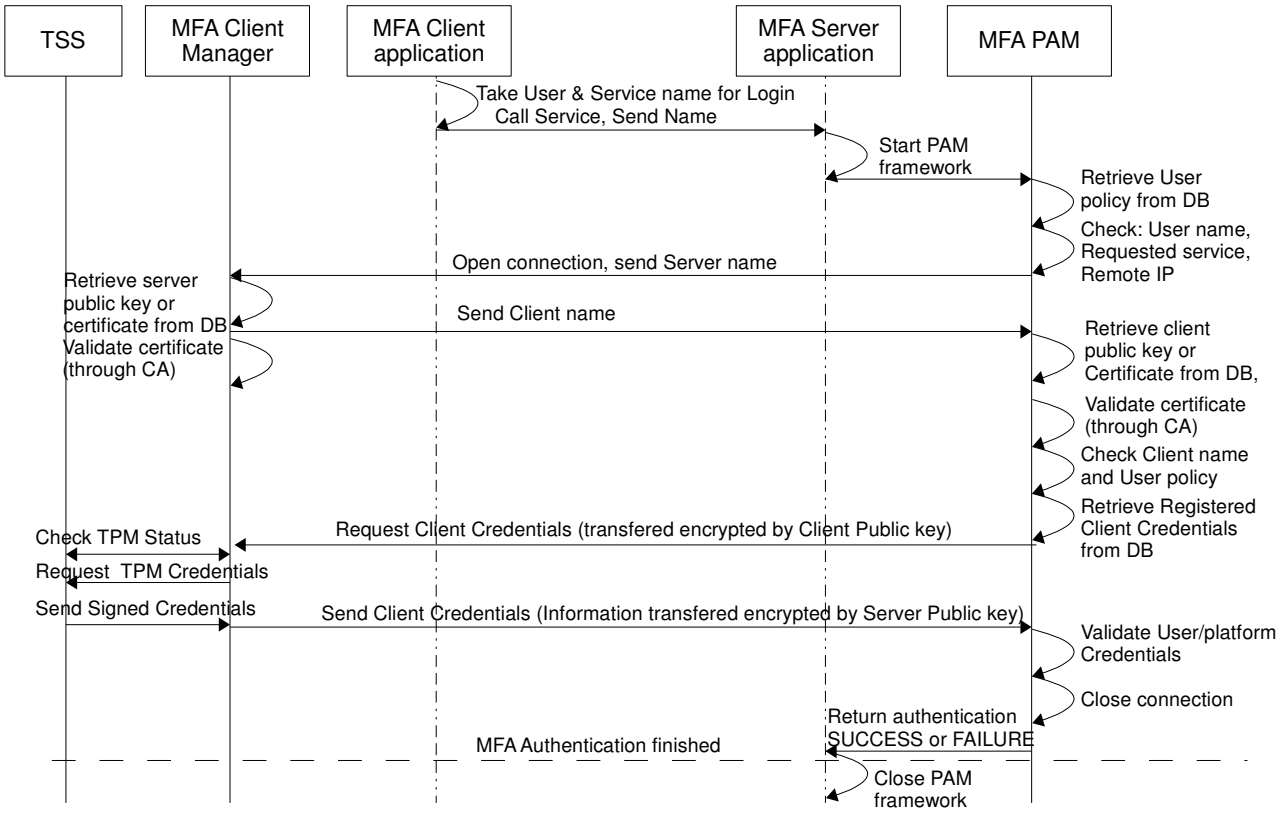


Figure 3: Client Authentication Protocol

Client Registration Protocol

Client/server message flow of the client authentication process is presented on the Figure 3.

Main task of registration procedure is to collect user and platform client credentials for future logon to the remote service and be sure that we got information exactly from computer about we think.

Before starting the registration:

- server has to be installed server part of MFA PAM system;

- client computer has installed TSS;
- client computer has installed client part of MFA PAM system;
- client and server have installed their own public key (certificates);
- in the case of usaging AIKs, the client has to have own AIK (AIK certificate).

As the first step of registration user (U) need to connect with server administrator (SA) (by defined for server or organization procedure) and describe his computer, installed software, required services, scheduler of services usage and etc. If server administrator approve user's request registration procedure could be started.

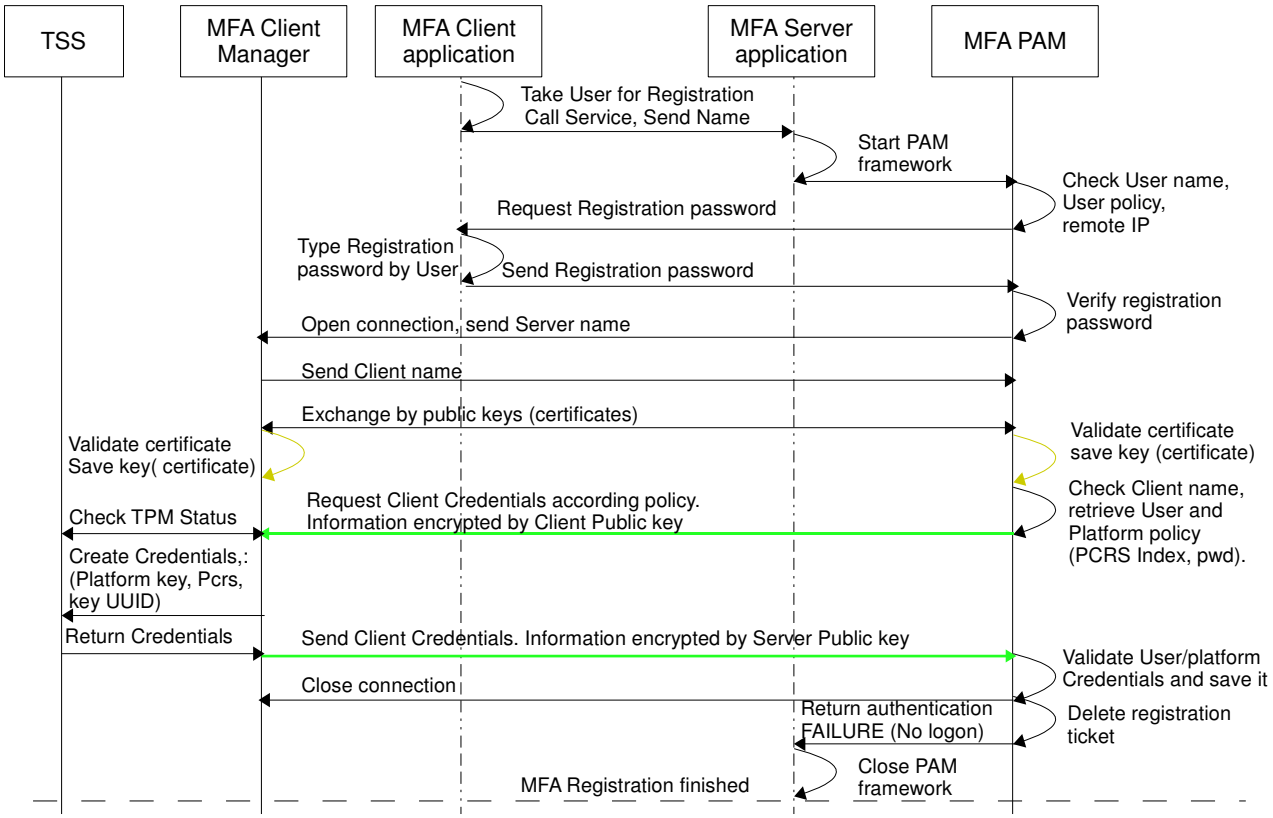


Figure 4: Client Registration Protocol

Registration steps:

1. SA: add new login name (if necessary), add registration ticket and policy for new client computer for MFA server configuration: set available services, usage policy and registration password;
2. SA: inform user about user login name, standard and registration passwords;
3. U (or user computer administrator – based on organization policy): check condition of client computer, OS and other software;
4. U: start one of possible console-based client application (for example: ssh) with registration user name (reg_<user name>) and server name or IP as parameters;
5. MFA PAM (server side):
   - get call parameters: login name, application/service name, connection

properties;
- check that connection with call parameters is possible according to configured policy;
- exchange public keys (certificates) with Clients;
- send request to MFA Client Manager for new user client and platform credentials, according to policy.

6. MFA Client Manager (client side):
   - exchange public keys (certificates) with Server;
   - collect user platform credentials according server request and policy,
   - send collected credentials to the server.
7. MFA PAM (server side):
   - validate user and client platform credentials and store it in the Configuration DB;
   - return PAM_AUTH_ERR;,
   - close connection to MFA client manager.
8. Registration procedure finished.

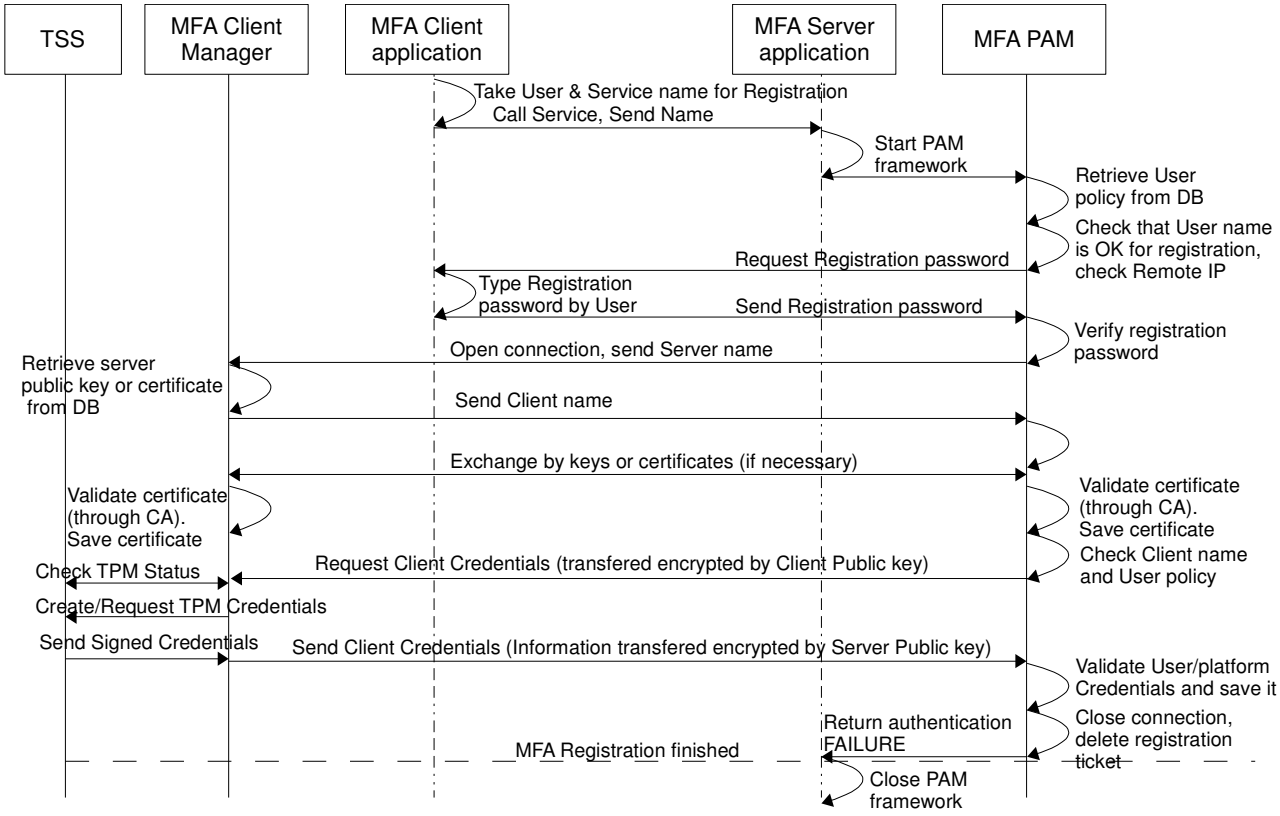


Figure 5: Client Registration Protocol

Policies

The policy determines what type authentication is required to access the remote service. Remote platform credential based on:

- client platform attestation identities key (AIK) or user key (UK);
- operating system (OS) and software set by platform configuration registers

(PCRs).

3.2.1 Registration tickets

Registration tickets is used for registration process and define registration policy. It contains following main parts:

- count of possible client platform could be registered for the user;
- user registration password;
- configuration of Client platform credential-policy;
- list of available services for user from the client platform.

4 MFA System API

MFA Prototype is implemented as a set of libraries that could be used for developing other MFA applications. This chapter describes developed API from low level up to high level functions.

4.1 Data Types

This section describes the basic data types defined by this API.

Pointer Size:
Pointer size becomes 32 bits on 32-bit systems and 64 bits with 64-bit system.

Basic Types:
There are some new types for 64-bit systems that were derived from the basic C language integer and long types, so they work in existing code. These are the expected values and definitions.

Type	Definition
UINT16	Unsigned INT16
UINT32	Unsigned INT32
BYTE	Unsigned character
MFA_UNICODE	MFA_UNICODE character. MFA_UNICODE characters are to be treated as an array of 16 bits.
MFA_PVOID	void Pointer (32 or 64 bit depending on architecture)

Derived Types:

Type	Definition	Usage
INTERCONNECT_DATA	UINT32	Interconnect Object attributes
SERVER_RESULT	UINT32	result of a MFA interface command
CONFIG_RESULT	UINT32	result of a MFA interface command
INTERCONNECT_RESULT	UINT32	result of a MFA interface command
CLIENT_RESULT	UINT32	result of a MFA interface command

4.2 Data Structure Types

This section describes the structures defined by this API.

4.2.1 mfa_c_typeServiceList

This structure provides list of Service that works with MFA.

Definition:

```
typedef struct mfa_c_tdServicesList {
      UINT16            Count;
      char**            List;
      } mfa_c_typeServicesList;
```

Parameters:

Count
Count of the Servces in the List
List
Pointer to List of MFA Services

mfa_c_typePCR

Definition:

```
typedef struct mfa_c_tdPCR {
      UINT8       Number;
      BYTE        Value[SHA_DIGEST_LENGTH];
      } mfa_c_typePCR;
```

Parameters:

Number
Number of PCR
Value
Value of PCR

4.2.2 mfa_c_typePCRList

This structure provides list of MFA Client Platform PCRs.

Definition:

```
typedef struct mfa_c_tdPCRList {
      UINT16      Count;
      mfa_c_typePCR*    List;
      } mfa_c_typePCRList;
```

Parameters:

Count
Count of the Platform configuration registers in PCR List
List
Pointer to List of Platform Configuration Register (PCR) numbers

4.2.3 mfa_c_typeRegTicket

This structure provides MFA authentication Policy String.

Definition:

```
typedef struct mfa_c_tdRegTicket {
      int   TicketsCount;
      char  RegLogin[CONFIG_MAX_USERNAME_LENGTH + 1];
```

```
        char  RegPassword[CONFIG_MAX_REGPASS_LENGTH + 1];
        mfa_c_typePCRList PCRs;
        int   KeyType;
        mfa_c_typeServicesList ServicesList;
        int TicketFile;
    } mfa_c_typeRegTicket;
```

Parameters:

TicketsCount
Count of registration tickets
RegLogin
Registration user name
RegPassword
Plain registration password
PCRs
List of PCRs to save/check
KeyType
Type of key for signature: CONFIG_KEY_NONE | CONFIG_KEY_USER | CONFIG_KEY_AIK
ServicesList
List of services available for user, service "ALL" - for all services
TicketFile
store of the open ticket file descriptor

4.2.4 mfa_c_typeUserPolicy

This structure provides information about MFA Client Platform and User data, including credentials, services, state, policy and so on. It used with the functions: ***mfa_server_GetUserCredential*** and ***mfa_server_SetUserCredentials***
Structure contains policy and credential information for user and peer pair

Definition:

```
    typedef struct mfa_c_tdUserPolicy {
        CONFIG_TRUE_OR_FALSE    EnablePolicy;
        int                     PeerStatus;
        char                    Login[CONFIG_MAX_USERNAME_LENGTH + 1];
        mfa_c_typePCRList       PCRs;
        int                     KeyType;
        TSS_UUID                KeyUUID;
        char*                   KeyPasswordHEX;
        mfa_c_typeServicesList  ServicesList;
        RSA*                    RSAPublicKey;
    } mfa_c_typeUserPolicy;
```

Parameters:

EnablePolicy
Enable or disable policy flag (TRUE or FALSE)
Peer status
CONFIG_PEER_(ENABLE|DISABLE|NOT_REGISTERED)
filled in by mfa_c_GetUserPolicy function
Login
login name
PCRs
list of PCRs numbers and values

KeyType
type of key for signature: CONFIG_KEY_NONE | CONFIG_KEY_USER | CONFIG_KEY_AIK
Key UUID
KeyUUID;
KeyPasswordHEX
TPM key password HEX string
ServicesList
List of services available for user, service "ALL" - for all services
RSAPublicKey
RSA public key

4.2.5 mfa_ic_TypeContainnerHeader

Definition:

```
typedef struct mfa_ic_tdContainerHeader {
      UINT16      Data_length;
      BYTE        Flags;
      BYTE        Data_SHA1[SHA_DIGEST_LENGTH];
} mfa_ic_typeContainerHeader;
```

Parameters:

Data_length
Size of data packet
Flags
INTERCONNECT_DATA_PLAIN
INTERCONNECT_DATA_RSA_ENCRYPT
Data_SHA1
Value of SHA1 of data packet

4.2.6 mfa_ic_TypeSocket

Main structure for interconnect communication functions.

Definition:

```
typedef struct mfa_ic_tdSocket {
      int               Socket;
      BYTE        Local_IP4[4];
      in_port_t   Local_Port;
      BYTE        Remote_IP4[4];
      in_port_t   Remote_Port;
      int         Timeout;
      char*       Remote_host_name;
      EVP_PKEY*   Remote_public_key;
      char*       Local_host_name;
      EVP_PKEY*   Local_private_key;
      size_t      Shared_key_size;
      BYTE*       Shared_key;
} mfa_ic_typeSocket;
```

Parameters:

Socket

Socket handler
Local_IP4[4]
Local IP address for the socket
Local_Port
Local port for the socket
Remote_IP4
Remore IP address for the socket
Remote_Port
Remote port for the socket
Timeout
Connection timeout in seconds (default is INTERCONNECT_TIMEOUT)
Remote_host_name
Name of remote host
Remote_public_key
Public key of remote host
Local_host_name
Name of local host
Local_private_key
My private key
Shared_key_size
Size of shred key for symmetric encryption
Shared_key
Shared key for symmetric encryption

4.3 Interface definition

The syntax used in describing the MFA application is based on the common procedural language constructs. Data types are described in terms of ANSI C.

4.3.1 MFA Server Functions

mfa_server_reg_InitializeConnection
Initialize a connection from server to the client during registration.

Definition:

```
SERVER_RESULT mfa_server_reg_InitializeConnection(
mfa_ic_typeSocket* socket,
char* client_host,
char** client_name);
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
client_host
DNS client name
client_name
Generated Client name

Return Values:

SERVER_RESULT

mfa_server_auth_InitializeConnection

Initialize a connection from server to the client during authentication.

Definition:

```
SERVER_RESULT mfa_server_auth_InitializeConnection(
mfa_ic_typeSocket* socket,
char* client_host,
char** client_name);
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

client_host
DNS client name

client_name
Generated Client name

Return Values:

SERVER_RESULT

mfa_server_ValidateUserCredential

This method sends request to client for credentials according to policy,
verifies credentials based on the matched and reference credentails, return Success, or Failure.

Definition:

```
SERVER_RESULT mfa_server_ValidateUserCredential(
mfa_ic_typeSocket* socket,
mfa_c_typeUserPolicy* policy,
char* client_name
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

policy
Pointer to structure `mfa_c_typeUserPolicy`

client_name
Client name

Return Values:

SERVER_RESULT

mfa_server_ObtainUserCredential

This method sends request to client for registration credentials according to policy.

Definition:

```
SERVER_RESULT mfa_server_ObtainUserCredential(
mfa_ic_typeSocket* socket,
char* client_name,
mfa_c_typeRegTicket* reg_ticket,
mfa_c_typeUserPolicy* policy
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
client_name
Client name
reg_ticket
Pointer to `mfa_c_typeRegTicket` structure
policy
Pointer to structure `mfa_c_typeUserPolicy`

Return Values:
SERVER_RESULT

mfa_server_ClientAuthentication
This method performs the full cycle of remote User And Platform authentication.

Definition:
```
SERVER_RESULT mfa_server_ClientAuthentication(
char* client_name, //in
char* user_name, //in
char* service //in
);
```
Parameters:
client_name
Name of the client host platform
user_name
Client User Name, who wanted to be logged on
service
Service name
Return Values:
SERVER_RESULT

mfa_server_ClientRegistration
This method performs the full cycle of remote Platform User And Platform registration.

Definition:
```
SERVER_RESULT mfa_server_ClientRegistration(
char* client_host,
char* user_name,
char* reg_password
)
```
Parameters:
client_host
DNS name of the client host platform, that should be registered
user_name
Client User Name, who wanted to be registered
reg_password
Password for registration
Return Values:
SERVER_RESULT

4.3.2 MFA Client Functions

mfa_ichl_NewTPMPublicKeyReply

Client Response to mfa_ichl_NewTPMPublicKeyRequest

Definition:

```
INTERCONNECT_RESULT mfa_ichl_NewTPMPublicKeyReply(
mfa_ic_typeSocket* socket,
char*       key_password
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

key_password
String with key password

Return Values:

INTERCONNECT_RESULT

mfa_ichl_QuoteReply

Retrieve information about user platform Credentials from TPM and send this information to the server.
This function is called after server have sent mfa_ichl_QuoteRequest.

Definition:

```
INTERCONNECT_RESULT mfa_ichl_QuoteReply(
mfa_ic_typeSocket* socket,
char* quota_param_str
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

quota_param_str
String with requested parameters for quote action

Return Values:

INTERCONNECT_RESULT

mfa_ichl_QuoteRequest

This function has two mode:

1. request PCRs values from Client and put them to response_pcrs for registration,
2. request Signature of PCRs values for authentication and verify it (this mode is used if response_pcrs == NULL).

Definition:

```
INTERCONNECT_RESULT mfa_ichl_QuoteRequest(
mfa_ic_typeSocket* ic_socket,
char* key_password_hex,
TSS_UUID key_uuid,
RSA* rsa_public_key,
mfa_c_typePCRList request_pcrs,
mfa_c_typePCRList* response_pcrs)
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
key_password_hex
Pointer to Key Password
key_uuid
Key identificater
rsa_public_key
Pointer to Public key for signature verification
request_pcrs
mfa_c_typePCRList structure with Pcrs numbers that should be quote
response_pcrs
Pointer to the structure mfa_c_typePCRList with the value of the Pcrs

Return Values:
INTERCONNECT_RESULT

ClientSideAuthentication

This method performs the actions to get Platform User And Platform authentication data.

Definition:

```
CONFIG_RESULT ClientSideAuthentication(
mfa_ic_typeSocket* socket,
char* server_name)
```

Parameters:
socket
Pointer to `mfa_ic_typeSocket` structure
server_name
Server Name

Return Values:
CONFIG_RESULT

ClientSideRegistration

This method performs the actions to get Platform User And Platform registration data.

Definition:

```
CONFIG_RESULT ClientSideRegistration(
mfa_ic_typeSocket* socket,
char* server_name)
```

Parameters:
socket
Pointer to `mfa_ic_typeSocket` structure
server_name
Server Name

Return Values:
CONFIG_RESULT

4.3.3 MFA Channel Module Functions

mfa_ic_OpenConnection

This method open connection to remote host.

Definition:

```
INTERCONNECT_RESULT mfa_ic_OpenConnection
(
char*             remote_host_name, // in
in_port_t         remote_port,      // in
mfa_ic_type_Socket* socket    // out
);
```

Parameters:

remote_host_name
Remote host name or IP
remote_port
Port on remote host we connect to
socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT
ERROR

mfa_ic_ListenConnection

Bind and listen for incoming connections.

Definition:

```
INTERCONNECT_RESULT mfa_ic_ListenConnection(
in_port_t local_port,   // in
mfa_ic_typeSocket* socket     // out
);
```

Parameters:

local_port
Local Port for listen incoming connections
socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ic_AcceptConnection

Accept incoming connection.

Definition:

```
INTERCONNECT_RESULT mfa_ic_AcceptConnection
(
mfa_ic_typeSocket* socket          // in/out
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ic_CloseSocket

Close socket.

Definition:

```
INTERCONNECT_RESULT mfa_ic_CloseSocket(
mfa_ic_typeSocket* Socket          // in/out
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ic_ShutdownSocket

Shutdown socket

Definition:

```
INTERCONNECT_RESULT mfa_ic_ShutdownSocket (
mfa_ic_typeSocket* socket          // in/out
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ic_SetPeerPublicKey

Associate Peer public key with the socket.

Definition:

```
INTERCONNECT_RESULT mfa_ic_SetPeerPublicKey
(
mfa_ic_typeSocket* socket,         // in/out
char* peer_name
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
peer_name
Name of remote host

Return Values:

INTERCONNECT_RESULT

mfa_ic_SetLocalPrivateKey

Associate Local Private key with the socket.

Definition:

```
INTERCONNECT_RESULT mfa_ic_SetLocalPrivateKey
(
mfa_ic_typeSocket* socket,         // in/out
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure

Return Values:
INTERCONNECT_RESULT

mfa_ic_WriteToSocket

Write buffer to socket with encryption (if encryption flag set).

Definition:

```
INTERCONNECT_RESULT mfa_ic_WriteToSocket(
mfa_ic_typeSocket* socket,    // in
UINT16 buffer_size,// in
BYTE* buffer,     // in
BYTE flags        // in
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
buffer_size
Size of buffer for transfer
buffer
Data for transfer
flags
Set of write operation flags, possible values:
INTERCONNECT_DATA_PLAIN
INTERCONNECT_DATA_ENCRYPT_BY_PUBLIC_KEY
INTERCONNECT_DATA_ENCRYPT_BY_SHARED_KEY
INTERCONNECT_DATA_SIGN_BY_PRIVATE_KEY
);

Return Values:
INTERCONNECT_RESULT

mfa_ic_ReadFromSocket

Read into buffer from socket with encryption (if encryption flag set).
Buffer is allocated inside function.

Definition:

```
INTERCONNECT_RESULT mfa_ic_ReadFromSocket(
mfa_ic_typeSocket* socket,    // in
UINT16* buffer_size,// in
BYTE** buffer,    // in
BYTE* flags       // in
);
```

Parameters:

socket
Pointer to `mfa_ic_typeSocket` structure
buffer_size
Size of received data
buffer
Received data
flags
Set of received transfer flags, possible values:

INTERCONNECT_DATA_PLAIN
INTERCONNECT_DATA_ENCRYPT_BY_PUBLIC_KEY
INTERCONNECT_DATA_ENCRYPT_BY_SHARED_KEY
INTERCONNECT_DATA_SIGN_BY_PRIVATE_KEY

Return Values:

INTERCONNECT_RESULT

mfa_ichl_SendMyKey

Take local public key (or certificate or AIK) in PEM format from Configuration and send it to peer.

Definition:

```
INTERCONNECT_RESULT mfa_ichl_SendMyKey(
mfa_ic_typeSocket* socket
);
```

Parameters:

socket

Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ichl_RecvAndSavePeerKey

Receive peer key (or certificate or AIK) in PEM format, compare with local one (if exists) and store it.

Definition:

```
INTERCONNECT_RESULT mfa_ichl_RecvAndSavePeerKey(
mfa_ic_typeSocket* socket,
char* peer_name
);
```

Parameters:

socket

Pointer to `mfa_ic_typeSocket` structure

Return Values:

INTERCONNECT_RESULT

mfa_ichl_NewTPMPublicKeyRequest

This method sends request to client for creating Pubic key, that will represent platform

Definition:

```
SERVER_RESULT mfa_ichl_NewTPMPublicKeyRequest(
mfa_ic_typeSocket* socket, //in
char** new_key_password_hex,//out
TSS_UUID* new_key_uuid,//out
RSA** rsa_public_key //out
);
```

Parameters:

socket

Pointer to `mfa_ic_typeSocket` structure

key_password_hex

Pointer to Key Password

key_uuid
Key unique identificater
rsa_public_key
Pointer to Public key

Return Values:
INTERCONNECT_RESULT

mfa_ic_ErrorString
Error code to string

Definition:

```
void mfa_ic_ErrorString(
INTERCONNECT_RESULT error_code,
char* error,
int err_string_size
);
```

Parameters:
error_code
Error code
error
Error string
err_string_size
Error string size

4.3.4 MFA Credentials and Policy Configuration Functions

mfa_db_GetUserCredential
This Method retrieves user reference credentials and policy from DB on the server for the authentication.

Definition:

```
SERVER_RESULT mfa_db_GetUserCredential(
char* client_name,
char* user_name,
char* service,
mfa_c_typeUserPolicy** policy
);
```

Parameters:
client_name
Client name
user_name
User name
service
Requested service
policy
Pointer to structure `mfa_c_typeUserPolicy`

Return Values:
SERVER_RESULT

mfa_db_GetUserCredentialParam
The Method retrieves the user registration ticket from DB on the server.

Definition:

```
SERVER_RESULT mfa_server_GetUserCredentialParam(
char* client_host,
char* user_name,
char* reg_password,
mfa_c_typeRegTicket** ticket
);
```

Parameters:

client_host
DNS client name
user_name
User name
reg_password
Registration password
ticket
Pointer to structure `mfa_c_typeRegTicket`

Return Values:

SERVER_RESULT

mfa_db_SaveUserCredentials

This method save User registration credentials in DB

Definition:

```
SERVER_RESULT mfa_db_SaveUserCredential(
char* user_name,
char* client_name,
mfa_c_typeUserPolicy policy
);
```

Parameters:

user_name
Client User Name, who wanted to be logged on
client_name
Name of the client host platform
policy
Structure `mfa_c_typeUserPolicy`

Return Values:

SERVER_RESULT

*mfa_c_*CheckClientSideConfiguration

This method checks a client side configuration.

Definition:

```
CONFIG_RESULT mfa_c_CheckClientSideConfiguration();
```

Return Values:

CONFIG_RESULT

*mfa_c_*CheckServerSideConfiguration

This method checks a server side configuration

Definition:

```
CONFIG_RESULT mfa_c_CheckServerSideConfiguration();
```

Return Values:
CONFIG_RESULT

mfa_c_IsPeerEnabled
This method checks that peer is enabled. On success return CONFIG_SUCCESS
else return peer status (CONFIG_PEER_DISABLED or CONFIG_PEER_NOT_REGISTERED).

Definition:
```
CONFIG_RESULT mfa_c_IsPeerEnabled(
char* peer_host_name)
```

Parameters:
peer_host_name
Pointer to String with host name

Return Values:
CONFIG_RESULT

mfa_c_EnablePeer
This method enables peer. By default registered peer is enabled.

Definition:
```
CONFIG_RESULT mfa_c_EnablePeer(
char* peer_host_name)
;
```
Parameters:
peer_host_name
Pointer to String with host name
Return Values:
CONFIG_RESULT

mfa_c_DisablePeer
This method disable peer.

Definition:
```
CONFIG_RESULT mfa_c_DisablePeer(
char* peer_host_name,
char* disable_reason
);
```
Parameters:
peer_host_name
Pointer to String with host name
disable_reason
Pointer to String with the reason

Return Values:
CONFIG_RESULT

mfa_c_GetRegTicket
Get registration ticket for user_name.

after success return ticket the file opened and locked and must be closed by mfa_c_CloseRegTicketFile

Definition:

```
CONFIG_RESULT mfa_c_GetRegTicket(
char* user_name,
mfa_c_typeRegTicket** ticket
);
```

Parameters:

user_name
User name
ticket
Pointer to the mfa_c_typeRegTicket structure

Return Values:

CONFIG_RESULT

mfa_c_SetRegTicket

Set registration ticket for user_name.
After success return ticket the file opened and locked and must be closed by mfa_c_CloseRegTicketFile.

Definition:

```
CONFIG_RESULT mfa_c_SetRegTicket(
char* user_name,
mfa_c_typeRegTicket* ticket
);
```

Parameters:

user_name
User name
ticket
mfa_c_typeRegTicket structure

Return Values:

CONFIG_RESULT

mfa_c_SetUserPolicy

This method set user policy and save credentials for peer. Before set policy it checks the peer is registered and enabled. If policy == NULL delete policy file.

Definition:

```
CONFIG_RESULT mfa_c_SetUserPolicy(
char* user_name,
char* peer_host_name,
mfa_c_typeUserPolicy policy
);
```

Parameters:

user_name
String with user name
peer_host_name
String with host name
Policy
Structure `mfa_c_typeUserPolicy`

Return Values:
CONFIG_RESULT

mfa_c_GetUserPolicy

This method gets user policy and credentials for peer. If policy == NULL returns new objects otherwise reuse existing object.

Definition:

```
CONFIG_RESULT mfa_c_GetUserPolicy(
char* user_name,
char* peer_host_name,
mfa_c_typeUserPolicy ** policy
);
```

Parameters:

user_name
String with user name
peer_host_name
String with host name
Policy
Pointer to structure `mfa_c_typeUserPolicy`

Return Values:
CONFIG_RESULT

5 MFA System Prototype Component Description

The following components will be described in the document :

- PAM Configuration (eg. for **sshd, ftpd**);
- MFA PAM - Authentication Service-**pam_mfa.so**;
  pam_mfa.so include the following modules:
  mfa_pam.c - main PAM code;
  server_side.c - registration and authentication module;
  interconnect_lib.c - low level interconnect functions library;
  interconnect_hl_lib.c - high level interconnect functions library
  config_llc_lib.c - low level functions library to work with configuration;
  config_lib.c - high level function library to work with configuration;
  mfa_errors.c - error module;
  log_lib.c - logging functions module;
- Utility for work with registration tickets -**mfa_reg_ticket**:
  mfa_reg_ticket.c - main executable file;
  config_lib.c - high level function library to work with configuration;
  config_llc_lib.c - low level function library to work with configuration;
  mfa_errors.c - error module;
  log_lib.c - logging functions module;

- MFA Client Manager – **mfa_client_manager**
  mfa_client_manager includes the following modules:

mfa_client_manager.c- client side of the authentication and regstration module;
interconnect_lib.c - low level interconnect functions library;
interconnect_hl_lib.c - high level interconnect functions library;
config_llc_lib.c - low level functions library to work with configuration;
config_lib.c - high level function library to work with configuration;
mfa_errors.c - error module;
log_lib.c - logging functions module.

5.1 PAM Configuration

The PAM Configuration is a system file, which is part of the PAM Framework. It provides possibility to use multifactor user authentication and registration with ability to authenticate entities based on arbitrary combination of credentials. Authentication providers could create new method authentication compliant with PAM requirements that will be available for authenticate a user.
For usage MFA PAM authentication the module mfa_pam.so must be added to PAM configuration file for required service in to *auth* section. See PAM documentation for more information.

Examples of PAM configuration files for sshd and ftpd services:
/etc/pam.d/ftpd:

```
#
# $FreeBSD: src/etc/pam.d/ftpd,v 1.18 2003/04/30 21:57:54 markm Exp $
# PAM configuration for the "ftpd" service
# auth

auth       required        pam_nologin.so        no_warn
auth       sufficient      pam_opie.so           no_warn no_fake_prompts
auth       requisite       pam_opieaccess.so     no_warn allow_local
#auth      sufficient      pam_krb5.so           no_warn
#auth       sufficient      pam_ssh.so            no_warn try_first_pass
auth       required        pam_unix.so           no_warn try_first_pass
auth       required        pam_mfa.so

# account
#account   required        pam_krb5.so
account    required        pam_unix.so

# session
session    required        pam_permit.so
```

/etc/pam.d/sshd:

```
#
# $FreeBSD: src/etc/pam.d/sshd,v 1.15 2003/04/30 21:57:54 markm Exp $
# PAM configuration for the "sshd" service
#
# auth
```

```
auth       required   pam_nologin.so        no_warn
auth       sufficient pam_opie.so           no_warn no_fake_prompts
auth       requisite  pam_opieaccess.so     no_warn allow_local
#auth      sufficient pam_krb5.so           no_warn try_first_pass
#auth      sufficient pam_ssh.so            no_warn try_first_pass
#auth      required   pam_unix.so           no_warn try_first_pass
auth       required  pam_mfa.so

# account
#account         required   pam_krb5.so
account          required   pam_login_access.so
account          required   pam_unix.so

# session
#session         optional   pam_ssh.so
session          required   pam_permit.so
```

5.2 MFA PAM Module

MFA PAM - Authentication/Registration Service is **pam_mfa.so**, configured locally on the server with a PAM Configuration System files /etc/pam.d/* . **pam_mfa.so** - PAM single shared library file that can be loaded by the PAM framework. The module registers and authenticates user with TPM credentials depends on the user policy.

This module performs its actions by opening a connection with TPM on client computer that requested PAM Service through interconnect library and sending the command to MFA Client Manager according to authentication or registration protocols. It requests MFA TPM credentials from client to register or authenticate user. It save requested credentials in configuration data base using Configuration library.

Standard PAM authentication function format is:

pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char *argv[])

5.3 Client Credentials and Policy DB

Client Credentials and Policy DB is located in directories and subdirectories.

Two directories used to keep these data:

One for user/platform registration policy - /usr/local/etc/mfa_pam/regtickets.

Another for User/Platform policy and credentials - /usr/local/etc/mfa_pam/peers.

/usr/local/etc/mfa_pam/regtickets – directory with registration tickets, that describes the policy for new users. Name of file with ticket is same as user name.

/usr/local/etc/mfa_pam/peers/<client name> - main configuration directory, that contains the list of the clients, that registered to have access to the server services, subdirectories contains registered clients and user credentials in files:

main files: peer.disable – if file exists peer disabled, files contain reason to be disabled

peer.certificate - peer certificate in PEM format

peer.public_key - peer public key in PEM format

<user name>.policy	- user policy and credentials
<Key UUID>.pub	- user public key in PEM format
<Key UUID>.cert	- user certificate in PEM format

5.4 MFA Client Manager

mfa_client_manager – application that executes commands that the server sends to the client to perform authentication or registration. It communicates with server through the interconnect library. It communicates with TPM service through the TSS library.

It works according the registration or authentication protocol.

5.5 Low level interaction protocol

Description of interconnect protocol

5.5.1 Container

Container structure used for transfer data block (blob)

```
Offset      Content
00-01       Data length
02          INTERCONNECT_DATA flags
03-23       SHA1 160 of data
24-END      data
```

5.5.2 Commands

The command system used to interact server and client through the channel:

REGISTRATION <peer name> - start registration process from peer
AUTHENTICATION <peer name> - start authentication process from peer

MY_NAME <host name> - send host name to peer
NEED_NEW_NAME - ask Server to generate a new name for Client
YOUR_NAME <new name> - response on NEED_NEW_NAME request

MY_PUBKEY
<public key in PEM format> - send public key in PEM format
MY_CERT
<certificate in PEM format> - send certificate in PEM format>

MAKE_NEW_KEY <new key password>
- request to make new TPM key with <password>
PUBLIC_KEY <UUID>
<public key in PEM format> - MAKE_NEW_KEY response

TPM_REQUEST QUOTE <parameters> - request for quote.
Parameters:
SIGN_KEY_UUID: <key uuid>
SIGN_KEY_PASSWD: <hex string with password>
CHALLENGE: <hex string with challenge>
PCRs: <pcrs list>
SEND_PCRs: YES

TPM_RESPONSE QUOTE <parameters> - quote response
Parameters:
TPM_PCR_COUNT: <TPM pcrs count>\n
QUOTE_HEADER: <first 8 bytes or quote resoponse HEX\n
PCRs: <pcrs values>\n (in necessary)\n
QUOTE_SIGNATURE: <quote signature HEX>\n

TPM_REQUEST SIGN_CHALLENGE - request for challenge signed by TPM
TPM_RESPONSE SIGN_CHALLENGE - response with signed challenge

PING <hex string with challenge> - PING request
PONG <hex string with challenge> - Response on PING request

ERROR - ERROR
QUIT - close socket

5.5.3 Registration process

```
Server: Connect to Client
Server: REGISTRATION <server name>
Client: Check that Server isn't disabled by configuration

If Client does not have own name,
     Client: REQ_NEW_NAME request
     Server: NEW_NAME response
endif

Client: MY_NAME <client name>
Server: Check that Client is not disabled by configuration
Server: MY_PUBKEY\n<server public key in PEM format>
Client: MY_PUBKEY\n<client public key in PEM format>
Server: MAKE_NEW_KEY <key password> (encrypted)
Client: PUBLIC_KEY <UUID>\n<public key in PEM format> (encrypted)
Server: TPM_REQUEST QUOTE <parameters> (encrypted)
Client: TPM_RESPONSE QUOTE <parameters> (encrypted)
Server: QUIT
```

The sample of the registration protocol can be found in the source code:
doc\examples\protocol\registration_quote.txt

5.5.4 Authentication process

```
Server: Connect to Client
Server: AUTHENTICATION <server name>
Client: Check that Server isn't disabled by configuration
Client: MY_NAME <client name>
Server: Check that Client isn't disabled by configuration
Server: TPM_REQUEST QUOTE <parameters> (encrypted)
Client: TPM_RESPONSE QUOTE <parameters> (encrypted)
Server: QUIT
```

The sample of the authentication protocol can be found in the source code:
doc\examples\protocol\authentication_quote.txt

5.6 Utility for work with registration tickets

```
mfa_reg_ticket <command> <command option>

Possible commands:
   add <user>       - Add ticket for user
   change <user>    - Change ticket for user
     Options for add or change:
         [--count=<tickets count>]
         [--password=<registration passwors>]
         [--tpm-key-type=(none|user|aik)]
         [--pcrs=<common separated PCRs numbers list>]
         [--services=<common list of services available for user>]
   show <user>      - Show ticket for user
   delete <user>    - Delete ticket for user
   list             - List of available tickets
```

6 MFA prototype Installation instructions

To compile and install software you must have:

1. GCC
2. installed TSS (trousers 0.2.6) includes and libraries
3. PAM includes and libraries (only on the server)
4. GNU make

To install MFA prototype components on the clients:

1. copy tpm_mfa.tbz (bzip2 tar archive) client machines with TPM
2. unpack the archive file to some directory (archive_root_dir) by command
   `tar -xjf  mfa_pam.tbz -C <source directory>`
3. change current directory to <source directory>/mfa
4. edit Makefile.client and make necessary changes (C compile, path to TSS libraries and includes)
5. compile mfa_client_manager by start command
   `gmake -f Makefile.client mfa_client_manager`
6. after succeseful compitation executable file mfa_client_manager will be in MFA/bin directory.
7. make directory /usr/local/etc/mfa_pam:
   ```
   mkdir /usr/local/etc/mfa_pam
   chmod 0700 /usr/local/etc/mfa_pam
   cd /usr/local/etc/mfa_pam
   ```
8. make private key
   `openssl genrsa -out my.key 2048`
9. write public part of key
   `openssl rsa -in my.key -pubout -out my.pub`
10. set SRK password (if exists) to srk.passwd file,
    `echo "<SRK password>" > srk.passwd`
11. copy mfa_client_manager execute file to place like /usr/local/sbin/,

```
cp <source dir>/MFA/bin/mfa_client_manager /usr/local/sbin/
```

To install MFA prototype components on the servers:

1. copy tpm_mfa.tbz (bzip2 tar archive) client machines with TPM
2. unpack the archive file to some directory (archive_root_dir) by command **tar -xjf mfa_pam.tbz -C <source directory>**
3. change current directory to **<source directory>/mfa**
4. edit Makefile.client and make necessary changes (C compile, path to TSS libraries and includes)
5. compile mfa_client_manager by start command:
   `gmake –f Makefile.server all`
   after successeful compitation executible file mfa_reg_ticket and shared PAM module pam_mfa.so.1 will be in MFA/bin directory.
6. make directory /usr/local/etc/mfa_pam,
   `mkdir /usr/local/etc/mfa_pam`,
   `chmod 0700 /usr/local/etc/mfa_pam`
   `cd /usr/local/etc/mfa_pam`
7. set server host name to file my.name,
   `echo "<server name>" > my.name`
8. make private key,
   `openssl genrsa –out my.key 2048`
9. write public part of key,
   `openssl rsa –in my.key –pubout –out my.pub`
10. copy mfa_reg_ticket execute file to place like /usr/local/bin/,
    `cp <source dir>/MFA/bin/mfa_reg_ticket /usr/local/bin/`
11. copy pam_mfa.so.1 to directory with PAM modules (/lib/security at LINUX environment or /usr/lib/ in FreeBSD),
    `cd <source dir>/MFA/bin/pam_mfa.so.1 /lib/security/`
12. edit PAM configuration for add pam_mfa.so module
13. make symbol link from pam_mfa.so.1 to pam_mfa.so,
    `cd /lib/security`
    `ln –s pam_mfa.so.1 pam_mfa.so`
14. add new registration tickets by start mfa_reg_ticket utility

7 Usage of MFA prototype for registration and authentication

7.1 Registration a new MFA user to the server

For Client platform registration execute any terminal-based application with enabled PAM MFA at server with the next parameters:

- user name: reg_<remote user name>;
- server name or IP.

For example:

ssh reg_irina@server

7.2 Logon to the server

To logon to the remote server you needn't any addition parameters. Only execute client application. For example:

ssh irina@server

7.3 MFA Prototype system list of files

Mfa list of files:

Makefile.client
Makefile.objects
Makefile.server
doc\install.txt
doc\interconnect_protocol.txt
doc\examples\pam.d
doc\examples\pam.d\ftpd
doc\examples\pam.d\sshd
doc\examples\protocol\authentication_no_tpm.txt
doc\examples\protocol\authentication_quote.txt
doc\examples\protocol\authentication_tpm_user_key.txt
doc\examples\protocol\registration_quote.txt
doc\examples\scripts\initialize_client.sh
doc\examples\scripts\initialize_server.sh
doc\examples\ssh\sshd_config

include\config_lib.h
include\config_llc_lib.h
include\credential.h
include\interconnect_hl_lib.h
include\interconnect_lib.h
include\log_lib.h
include\mfa_errors.h
include\mfa_pam.h
include\server_side.h
include\some_types.h
include\tpmless_lib.h
include\tpm_lib.h

src\config_lib.c
src\config_llc_lib.c
src\credential.c
src\interconnect_hl_lib.c
src\interconnect_lib.c
src\log_lib.c
src\mfa_client_manager.c
src\mfa_errors.c
src\mfa_pam.c
src\mfa_reg_ticket.c
src\server_side.c
src\tpmless_lib.c
src\tpm_lib.c

List of abbreviations

Listing of term definitions and abbreviations used in this document (IT expressions and terms from the application domain).

Abbreviation	Explanation
AIK	Attestation Identity Key
API	Application Programming Interface
MFA	MultiFactor Authentication
OS	Operating System
PCR	Platform Configuration Register
SSL	Secure Sockets Layer
TC	Trusted Computing
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSS	Trusted Software Stack

8 Referenced Documents

/1/ TCG Specification, Architecture Overview.
http://www.trustedcomputing.org
April 28, 2004,
Version 1.2

/2/ TCG Software Stack (TSS) Specification
January 6, 2006,
Version 1.2

/3/ PAM
http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/Linux-PAM_ADG.htmlhttp://msdn.microsoft.com
Version 0.99.6.0, 5. August 2006.

/4/ Secure Coding Guidelines
http://msdn.microsoft.com
2004

/5/ Improving Web Application Security: Threats and Countermeasures
http://msdn.microsoft.com

/6/ Writing Secure Code, Second Edition, by Michael Howard, David C. LeBlanc

/7/ OpenSSL Toolkit ***www.openssl.org/***

D6.3 Collection of all SWP deliverables (with nature=R) produced during month 13-24

ANNEX

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Deliverable

Deliverable reference number	IST-027635/D06.3/FINAL | 1.00
Deliverable title	D6.3 Collection of all SWP deliverables (with nature=R) produced during month 13-24
WP contributing to the deliverable	WP6
Due date	Oct 2007

Responsible Organisation	LDV,Lehrstuhl für Datenverarbeitung, TUM
Authors	Chun Hui Suen
Abstract	
Keywords	OpenTC WP6

Dissemination level	Public
Revision	FINAL | 1.00

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

1.

WP06d.1 Encrypted File Service Application Programming Interface Specification

Project number	IST-027635
Project acronym	Open_TC
Project title	Open Trusted Computing
Deliverable type	Deliverable

Deliverable reference number	IST-027635/D06d.1/Final/1.0
Deliverable title	WP06d.1: Compartmented Security
WP contributing to the deliverable	WP6d Trusted Computing Based Encrypted File Service
Due date	Mar 2007
Actual submission date	April 2007

Responsible Organisation	PORT
Authors	Burak Oğuz, Baran Erdoğan, Bora Güngören
Abstract	Encrypted File Service comes with number of features that are supported by trusted computing infrastructure and Open Trusted Computing enhancements for compartmented operation on Xen and L4 hypervisors. EFS Application Programming Interface(API) simplifies inclusion EFS features in all type of applications
Keywords	

Dissemination level	Public | Confidential
Revision	

Instrument	IP	**Start date of the project**	1st November 2005
Thematic Priority	IST	**Duration**	42 months

Table of Contents

List of figures

List of Tables

1 Introduction

Encrypted File Service comes with number of features that are supported by trusted computing infrastructure and Open Trusted Computing enhancements for compartmented operation on Xen and L4 hypervisors. These features are also useful by application programmers whom may want to use EFS functionalities on their programs and may want to use trusted security service for ensuring trusted secure data storage for their programs.

EFS Application Programming Interface(API) simplifies inclusion EFS features in all type of applications

EFS API is a communication interface between EFS Core Service and applications. All inputs and output messages shall be handled externally by applications using it. Service related messages will be given as output parameter to API calling function.

1.1 EFS Block Diagram

EFS is composed of three main building blocks :

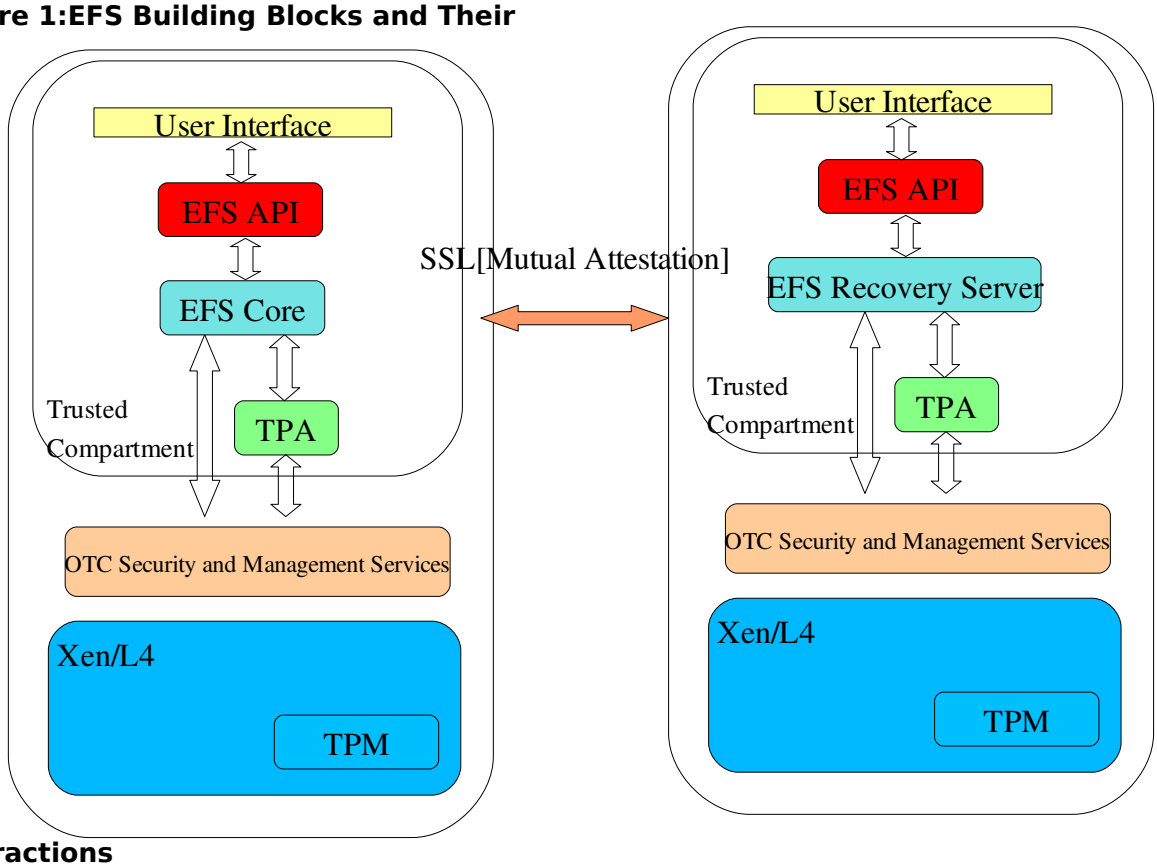
1. EFS Core : This block is the main operator block which manages file,key and recovery operations for each platforms. Also responsible from administrative tasks in non-networked environments.
2. EFS Recovery Server : Recovery Server manages recovery and administrative operations in networked environments.
3. EFS API : API is responsible for establishing communication between user interface (which is also an application) and EFS Core.

This relationship is shown in Figure 1

1.2 Target Audience

This document is written for application developers who are interested in using EFS's features. In order to use this API, developer should have basic data security and trusted computing knowledge. Furthermore, developer should know how to handle passwords and other user-oriented important data. Also developer should have knowledge on importing and using other API's in their source code.

Figure 1:EFS Building Blocks and Their



interactions

1.3 API Method Description

EFS API is described in the following way explained below.

<method name as header> **<method definition>** **<method explanation>** **<input parameters**(optional)> **<return values**(optional)> **<dependent methods**(optional)> **<sample usage in CLI**(optional)>

Table 1:General structure of an API method entry

Each EFS method is explained under the name of the same topic.
Sample EFS API method declaration is shown in Table 2 below.

```
int trustedUserKeyBackup(String backupUrl,
                                byte* recoveryPasswd,
                                byte* userPasswd)
```

Table 2:Sample method definition

As it is seen from Table2, method name is represented in bold. Following the declaration, input parameters of the method is specified.

Informative comment about each method is given following the method declaration. Sample informative comment is shown in Table 3.

Start of informative comment Trusted back up operation for user keys. **End of informative comment**

Table 3:Explanation part of method

Informative comment of method is followed by detailed explanation of method parameters. Sample “Parameters” section is shown in Table 4.

Parameters: `backupUrl` - URL of the directory which user keys will be backed up `recoveryPasswd` - password for S2K encryption. `userPasswd` - password for keys which are going to be backed up.

Table 4:Input parameters explained in detail

As shown in Table 5, “Returns” field follows “Parameters” field. In “Returns” field, return parameter of the method is explained in detail.

Returns: status flag SUCCESSFUL/FAILED

Table 5:Return value

If the method have dependencies and preconditions for execution, “Dependency” field will be used for specifying the methods dependencies.

Dependency: *openSession(String)*

Table 6:Dependency

Finally there will be usage scenarios which explain usage of the API method in EFS CLI. These part is a guide for application programmers, which explains logical flow of input and output data to method. Such as specifying the execution step where to ask for passwords or how to obtain some other user originated data. For EFS, CLI usage is taken as a reference and “Sample Usage in CLI” section is added afterwards “Dependency” section. This is shown in Table 7.

Sample Usage in CLI:

$ efscli --trusted-user-keys-backup /mnt/sdb1/foo/

Enter key password :

Enter key backup password :

Reenter key backup password :

User keys back up operation -> SUCCESSFUL/FAILED

Table 7:Sample usage scenarios

There is no sample source codes available in this document. For source code examples EFS CLI have to be examined.

1.4 Input Parameter Limitations

Input parameters have certain limitations which should be obeyed strictly.

- **String**s should not be longer than 256 characters.
- **byte*** should end with null character.
- **boolean** should be “true” or “false”.
- **int** should be taken as an 32 bit integer value with sign.

1.5 Return Value Limitations

Return values have certain limitations which should be obeyed strictly.

- **int** type return values should be one of these constant values defined in 2.1.1
- **int*** type defines a file descriptors and should end with an EOF.
- **String** type should be handled as in <string.h> library.
- **Vector** type should be handled as in <vector> library
- **Platform*** should end with an zero valued id entry.
- **PlatformResults*** should end with an zero valued id entry.
- **RecoveryList*** should end with an zero valued id entry.
- **RecoveryResults*** should end with an zero valued id entry.
- **KeyInfo*** should end with an zero valued id entry.

1.6 OS Dependency

EFS API will be platform-independent, this means EFS API may not be restricted to any operating systems, independent of the operating system's resource limitations and user restrictions. However, it is the case that EFS Core should work for that OS where API is going to run.

1.7 Abbreviations

Abrreviation	Description
API	Application Programming Interface
EFS	Encrypting File Service
CLI	Command Line Interface
RS	Recovery Server
RA	Recovery Agent
TPM	Trusted Platform Module
OS	Operating System

Table 8:Abbreviations Table

2 API Units

EFS API consists of 5 main parts.

- **File Operations**
  File operations are simple generic file encryption service operations which are encryption, decryption ,signing ,verifying and shredding.
  Remark: File recovery operation is distinguished from operations mentioned above because there is a need for administrative authorization for both local and networked file recovery operations.
- **User Operations**
  User operations cover operations related with end user like policy management or keystore password operations.
- **Key Management Operations**
  Key management operations contains EFS key operations which will be used by programmers. API programmers can create, delete, migrate, backup and restore EFS keys. They can also import keys from GNU PG and PGP
- **Recovery Agent Operations**
  Application programmers can use EFS API for file recovery actions used in both standalone and networked installations. They can manage file recovery actions by using this part of EFS API.
- **Recovery Server Operations**
  EFS offers trusted central file recovery option with EFS Recovery Server. Recovery server is also capable of distributing enforcing EFS policies to all trusted recovery agents registered. These operations are managed by EFS API functions explained in this section.
- **Administrative Operations**
  According to configured enforcing policies, EFS may need administrative authentication and management actions from application programmer. These functionalities are managed with the functions in this section.

2.1 Constants

EFS API calls have return values associated with the result of the action performed in API function. In order to clarify return values between EFS API functions and applications, constants in this section is introduced.

2.1.1 Return Types

2.1.1.1 SUCCESSFUL

```
static const int SUCCESSFUL
```

This is a static integer value which will be used as status flag that returns from a method. It symbolizes success from the called method.

2.1.1.2 FAILED

```
static const int FAILED
```

This is a static integer value which will be used as status flag that returns from a method. It symbolizes failure from the called method.

2.1.1.3 NOT_IMPLEMENTED

```
static const int NOT_IMPLEMENTED
```

This is a static integer value which will be used as status flag that returns from a method. It symbolized that the called method is not implemented.

2.1.2 File Operation Types

2.1.2.1 ENCRYPT

```
static const int ENCRYPT
```

This is a static integer value which will be used as a parameter to fileOperation method as operation. It symbolizes file encryption.

2.1.2.2 DECRYPT

```
static const int DECRYPT
```

This is a static integer value which will be used as a parameter to fileOperation method as operation. It symbolizes file decryption.

2.1.2.3 SIGN

```
static const int SIGN
```

This is a static integer value which will be used as a parameter to fileOperation method as operation. It symbolizes file signing.

2.1.2.4 VERIFY

```
static const int VERIFY
```

This is a static integer value which will be used as a parameter to fileOperation method as operation. It symbolizes file verifying.

2.1.2.5 SHRED

```
static const int SHRED
```

This is a static integer value which will be used as a parameter to fileOperation method as operation. It symbolizes file shredding.

2.1.3 Encryption Algorithm Types

2.1.3.1 AES-ECB

```
static const int AES-ECB
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes AES-ECB encryption scheme.

2.1.3.2 AES-CBC

```
static const int AES-CBC
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes AES-CBC encryption scheme.

2.1.3.3 DES-ECB

```
static const int DES-ECB
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes DES-ECB encryption scheme.

2.1.3.4 DES-CBC

```
static const int DES-CBC
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes DES-CBC encryption scheme.

2.1.3.5 DESede-ECB

```
static const int DESede-ECB
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes DESede-ECB encryption scheme.

2.1.3.6 DESede-CBC

```
static const int DESede-CBC
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It symbolizes DESede-CBC encryption scheme.

2.1.3.7 BLOWFISH

```
static const int BLOWFISH
```

This is a static integer value which will be used as a parameter to fileOperation method as symmetric encryption algorithm to be used in this single operation. It

symbolizes BLOWFISH encryption scheme.

2.1.4 Key Types

2.1.4.1 KEY_BIND

```
static const int KEY_BIND
```

This is a static integer value which will be used as a parameter to key import operation operation. It symbolizes that given key will be used for binding purposes.

2.1.4.2 KEY_SIGN

```
static const int KEY_BIND
```

This is a static integer value which will be used as a parameter to key import operation operation. It symbolizes that given key will be used for signing purposes.

2.1.4.3 KEY_LEGACY

```
static const int KEY_LEGACY
```

This is a static integer value which will be used as a parameter to key import operation operation. It symbolizes that given key will be used for both binding and signing purposes.

2.2 File Operations

2.2.1 getLastCommandStatus

```
String getLastCommandStatus()
```

Start of informative comment

Returns detailed log for the last executed EFS command.

End of informative comment

Returns:

This function returns detailed output log of the last executed command. Implemented to get detailed log of EFS last EFS executed command.

2.2.2 hasOpenSession

```
int hasOpenSession()
```

Start of informative comment

Checks whether user has an ongoing EFS session. If not, program should ask for EFS session password of the user.

End of informative comment

Returns:

This function return status flag SUCCESSFUL/FAILED.

If SUCCESSFUL is returned, user has open EFS session.

If FAILED is returned, user does not have open EFS session. Application has to open EFS session for user with **openSession** command.

2.2.3 openSession

```
int openSession(byte* password)
```

Start of informative comment

Opens a new EFS session for the user if supplied user name and password is valid.

End of informative comment

Parameters:

`password` – user session password

Returns:

If password is correct a new session will be opened for the user and function will return SUCCESSFUL. Otherwise it will return FAILED.

2.2.4 checkUserPasswd

```
int checkUserPasswd(byte* password)
```

Start of informative comment

Checks validity of user password.

End of informative comment

Parameters:

`password` - Password of the user who sends request

Returns:

If user password is correct and opens keystore then method will return SUCCESSFUL. Else function will return FAILED.

2.2.5 fileOperation

```
int fileOperation(int operation,
                        String inputFile,
                        String outputFile,
                        int algorithm,
                        boolean isTar,
                        boolean open,
                         boolean shred,
                        String keyURL,
                        String keyType,
                         byte* userKeyPass
                        boolean verbose)
```

Start of informative comment

If user does not have an ongoing session, user have to be asked for password.

End of informative comment

Parameters:

`operation` - operation name (see 2.1.2)

- ENCRYPT
- DECRYPT
- SIGN
- VERIFY
- SHRED

`inputFile` - name of the input file (absolute URL like /home/foo/bar.txt)

`outputFile` - name of the output file. If null, output file will be named as inputFileName+".efs" . If no output file name is supplied then EFS will decrypt file to the file's original filename which was held in file header in decryption. (absolute URL like /home/foo/bar.txt.efs)

`algorithm` - name of the algorithm to be used in bulk file encryption. If null, Blowfish will be used as default symmetric encryption method. (see 2.1.3)

- AES-ECB
- AES-CBC
- DES-ECB
- DES-CBC
- DESede-ECB
- DESede-CBC
- BLOWFISH

`isTar` - creates an archive. Input file has to be a directory

`open` - Opens EFS encrypted file with associated program to its MIME type. File decryption is transparent to user.

	Applicable on EFS encrypted files.
shred	- Shreds input file after file encryption/decryption. Default value is true.
keyURL	- URL of the external asymmetric key to be used for encryption. If null, default EFS JKS keys will be used. (absolute URL like/home/foo/key.pub)
keyType	- type of the external asymmetric key to be used in encryption. Possible values are • PGP • GPG • JKS
userKeyPass	- password of the keys given by the user externally.
verbose	- detailed output mode

Returns:

This function returns status flag FAILED if there is an error. Otherwise it will return file descriptor of the processed file.

Dependency:

openSession(byte *)

checkUserPasswd(byte*)

Sample Usage in CLI:

1. *Authenticate user.*

   Password for key id : xxxxxx

   User authentication -> SUCCESFUL/FAILED
2. No Message (means successful operation)
3. File processes in verbose mode.

   file name before operation -> operation name -> file name after operation
4. Id of the key and type which is going to be used in file operation

   Keyid = <int> , Type = <string {PGP,GPG,JKS}>

2.2.6 decryptInVolatileMemory

```
int* decryptInVolatileMemory(String inputFile,
                        boolean open,
                        String keyURL,
                        String keyType,
                       byte* userKeyPass,
                        boolean verbose)
```

Start of informative comment

Decrypts an EFS encrypted file to volatile memory(RAM) instead of hard disk. In case of sudden power disruption, decrypted open data does not reside on hard disk.

Remark: If user does not have an ongoing EFS session, user have to be asked for EFS session password

End of informative comment

Parameters:

inputFile	- Name of an EFS encrypted input file (absolute URL like /home/foo/bar.txt.efs)
open	- Opens EFS encrypted file with associated program to its MIME type. File decryption is transparent to user. Applicable on EFS encrypted files.
keyURL	- url of the external asymmetric key to be used in encryption. If null, default EFS JKS keys will be used. (absolute URL like /home/foo/key.pub)
keyType	- type of the external asymmetric key to be used in encryption. Possible values are • PGP • GPG • JKS
userKeyPass	- password of the keys given by the user externally.
verbose	- detailed output mode

Returns:

This function returns status flag FAILED if there is an error. Otherwise it will return a memory pointer to the processed file.

Dependency:

openSession(String)

checkUserPasswd(byte*)

2.2.7 localFileRecovery

```
int localFileRecovery(String inputFile,
                      byte* recoveryPasswd, String X509CerPath, String
                       X509PrivPath)
```

Start of informative comment

Local file recovery command. Requires administrator authorization to get Recovery Agent Keys on local platform.

End of informative comment

Parameters:

`inputFile`	- name of the file to be recovered(absolute URL eg. /home/foo/bar.txt.efs)
`recoveryPasswd`	- password for recovery agent keys.
`X509CerPath`	- absolute path to X.509 certificate file which contains user's new certificate. If null EFS will generate new keys for user.
`X509PrivPath`	- absolute path to X.509 private key file which contains user's new private key. If null EFS will generate keys for user.

Returns:

status flag SUCCESSFUL/FAILED

Sample Usage in CLI:

$ efscli --recover foo.txt,bar.pdf

Password for Recovery Keys : xxxxx

Authorization Granted/Denied.

Recovery -> SUCCESFUL/FAILED

2.2.8 remoteRecovery

```
int remoteRecovery(String inputFile, String X509CerPath, String X509PrivPath)
```

Start of informative comment

Command for networked recovery. Recovery is performed on EFS Recovery Server. Requires administrative authority on recovery server.

End of informative comment

Parameters:

`inputFile`	- name of the file to be recovered. (absolute url like /home/foo/bar.txt.efs)
`X509CerPath`	- absolute path to X.509 certificate file which contains user's new certificate. If null EFS will generate keys for user.
`X509PrivPath`	- absolute path to X.509 private key file which contains user's new private key. If null EFS will generate keys for user.

Returns:

status flag SUCCESSFUL/FAILED

Sample Usage in CLI:

$ efscli --recovery-request foo.txt,bar.pdf

Recovery Request is sent to Recovery Server. Waiting for administrative approval.

Request approved/denied.

Recovery SUCCESFUL/FAILED

2.2.9 Sample CLI Commands for File Operations:

$ efscli -i /home/xxx/bar.txt -op encrypt

Enter user password :
User authentication -> SUCCESFUL/FAILED

$ efscli -v -i /home/xxx/bar.txt.efs -op decrypt

Keyid = <int> , Type = <string {PGP,GPG,JKS}>

bar.txt.efs -> decrypt -> bar.txt -> SUCCESSFUL

$ efscli -v -i /home/xxx/foo.odt -o /home/xxx/myfoo.efs -op encrypt –
uk /mnt/sdb1/mykey.pub -kt GPG

Password for key '/mnt/sdb1/mykey.pub' :

foo.odt -> encrypt -> myfoo.efs -> SUCCESSFUL

$ efscli -v -i /home/xxx/foo.odt -op decrypt –uk /mnt/sdb1/mykey.priv -kt GPG

Password for key '/mnt/sdb1/mykey.priv' :

myfoo.efs -> decrypt -> foo.odt -> SUCCESSFUL

2.3 User Operations

Operations under this section is dedicated to single specific user who has open EFS session.

2.3.1 listKeys

```
KeyInfo* listKeys(byte* userPasswd)
```

Start of informative comment

Returns detailed list of user EFS keys.

End of informative comment

Parameters:

userPasswd – password for the user's current keystore.

Returns:

status flag SUCCESSFUL/FAILED

Dependency:

KeyInfo Class(see 3.3.1)

Sample Usage in CLI:

$ efscli --list-keys

Enter user password :

User authentication -> SUCCESFUL/FAILED

Keyid = 1, Name=MyKey, Hash=.... , Type = JKS

...

Keyid = 4, Name=FooKey, Hash=.... , Type = JKS

2.3.2 changeUserPasswd

```
int changeUserPasswd(byte* oldpassword,byte* newpassword)
```

Start of informative comment

Changes EFS session password of the user.

End of informative comment

Parameters:

oldpassword	- Old user password
newpassword	- New user password

Returns:

This method returns SUCCESSFUL if it can open keystore with old password and save it with new password.

Sample Usage in CLI:

$ efscli --change-user-passwd

Enter old user password :

User authentication -> SUCCESSFUL/FAILED

Enter new user password :

Enter new user password (again):

User password change -> SUCCESSFUL/FAILED

2.3.3 listMACPolicies

```
Vector<String[]> listMACPolicies()
```

Start of informative comment

Lists mandatory access policies enforced by EFS specified by administrator.

End of informative comment

Returns:

This function returns a map which contains mandatory policy names with regarding policy value.

2.3.4 listDACPolicies

```
Vector<String[]> listDiscretionaryPolicies(byte* password)
```

Start of informative comment

Lists discretionary access policies which are specified by individual EFS user.

End of informative comment

Parameters:

`password` - Current user password

Returns:

This function returns a map which contains discretionary policy names matched with specified policy value. Also whole policies are followed by value formats(i.e. date Y-m-d)

2.3.5 setDACPolicy

```
int setDACPolicy(byte* password,
                        String policyName,
                        String policyValue)
```

Start of informative comment

Sets discretionary access policy with given name to a given value. If policy value does not fit to given format, function will return failure. If policy value does not fit mandatory access policy EFS will return error.

End of informative comment

Parameters:

password	- Current user password
policyName	- Name of the policy that user want to change
policyValue	- Value of the policy that user want to change

Returns:

This function returns successful if given password and policy name are correct and policy value fits into the predefined for the specified policy.

2.3.6 resetDACPolicy

```
int resetDACPolicy(byte* password)
```

Start of informative comment

Resets discretionary access policies to default predefined values.

End of informative comment

Parameters:

`password` - Current user password

Returns:

This function returns successful if given password is correct.

2.3.7 setTrustedSignOnValue

```
int setTrustedSignOnValue(byte* userPassword,
                          String signOnPhrase)
```

Start of informative comment

Sets trusted sign on value which can be used in checking platform authentication.

End of informative comment

Parameters:

password	- Current user password
signonvalue	- 256 byte authentication value which will be used for trusted sign on.

Returns:

This function returns successful if given password is true and trusted sign on value is not null.

2.3.8 checkTrustedSignOn

`String checkTrustedSignOn()`

Start of informative comment

Returns the trusted sign on value which has been predefined by the user

End of informative comment

Returns:

This function returns trusted sign on value if password is correct and a trusted sign on value is set before.

2.3.9 clearTrustedSignOnValue

```
int clearTrustedSignOnValue(byte* password)
```

Start of informative comment

Clears the trusted sign on value and unregisters TPM key dedicated to it.

End of informative comment

Parameters:

`password` - Current user password

Returns:

This function returns successful if user password is correct and a trusted sign on value is defined before.

2.4 Key Operations

2.4.1 checkAdminPasswd

```
int checkAdminPasswd(byte* password)
```

Start of informative comment

Checks whether compartment administrator password is correct or not.

End of informative comment

Parameters:

password	- Password of the EFS administrator on compartment

Returns:

If platform administrator password is not correct, method returns FAILED. Else it returns SUCCESSFUL.

2.4.2 generateUserKey

```
int generateUserKey(byte* userPass,int force, byte* adminPass)
```

Start of informative comment

Generates file encryption key for user to be used for file operations. If user has generated keys, must be used with force=1 parameter to regenerate keys.

End of informative comment

Parameters:

`userPass`	- Authentication password for user keys
`force`	- If force is used method will generate a new keystore by deleting the old one.
`adminPass`	- Password of the administrative user.

Returns:

If one of the user or administrator passwords is wrong, function will return FAILED. If there exists a user keystore and force=1 will return FAILED.

Sample Usage in CLI:

```
$ efscli --generate-user-key

Enter administrator password :
Enter key password :

User key generation -> SUCCESSFUL/FAILED
```

2.4.3 addNewUserKey

```
int generateUserKey(byte* userPass, int keyType)
```

Start of informative comment

Generates new file encryption key for user to be used for file operations.

End of informative comment

Parameters:

userPass	- Authentication password for user keys
adminPass	- Type of the key (see 2.1.4) • KEY_BIND • KEY_SIGN • KEY_LEGACY

Returns:

If user password is wrong, function will return FAILED.

2.4.4 deleteUserKey

```
int deleteUserKey(byte* userPasswd,int keyid)
```

Start of informative comment

Deletes asymmetric key pair of the user which are used in file operations.

End of informative comment

Parameters:

userPasswd	- Password for user keys
keyid	- Identification number of the key which is going to be deleted.

Returns:

If user password and key identification is correct method will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --delete-user-key 3

Enter user password :

User key deletion -> SUCCESSFUL/FAILED

2.4.5 RATrustedBackup

```
int RATrustedBackup(String backupUrl,byte* recoveryPasswd, byte*
backupPasswd )
```

Start of informative comment

Trusted backup of the Recovery Agent keys on local platform.

End of informative comment

Parameters:

backupURL	- url of the directory which recovery agent keys will be backed up(absolute url like /mnt/sda1/keys)
recoveryPasswd	- Password for recovery agent keys
backupPasswd	- Backup password for restore operation.

Returns:

If back up URL is valid, recovery password is correct and backup password is not null it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --trusted-backup-ra-keys /mnt/sdb1/foo/

Enter Recovery password :

Authorization granted/denied

RA keys back up -> SUCCESSFUL/FAILED

2.4.6 trustedKeyBackup

```
int trustedKeyBackup(String backupUrl,
                           byte* backupPasswd, byte* adminPass)
```

Start of informative comment

Trusted backup of EFS Key tree.

End of informative comment

Parameters:

backupUrl	- URL of the directory which EFS keys will be backed up(absolute url like /mnt/sda1/keys)
backupPasswd	- password for S2K encryption.
adminPass	- Password of the administrative user.

Returns:

If back up URL is valid, administrator password is correct and backup password is not null it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --trusted-key-backup /mnt/sdb1/foo/

Enter administrator password :

Authorization granted/denied

Enter EFS Key Tree Recovery password :

Reenter EFS Key Tree Recovery password :

EFS Key Tree back up -> SUCCESSFUL/FAILED

2.4.7 trustedKeyRestore

```
int trustedKeyRestore(String restoreUrl,
                          byte* backupPasswd, byte* adminPass)
```

Start of informative comment

Restore operation of keys on a platform. Needs EFS key tree password and a recovery password for S2K decryption.

End of informative comment

Parameters:

`restoreUrl`	- URL of the directory which EFS keys will be restored. If null, they will be restored to default directory.(absolute url like /mnt/sda1/keys)
`backupPasswd`	- password for S2K decryption.
`adminPass`	- Password of the EFS administrator.

Returns:

If restore URL is valid, administrator password is correct and backup password is not null it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --trusted-key-restore

Enter administrator password :

Authorization granted/denied

Enter EFS Key Tree password :

EFS Key Tree restore -> SUCCESSFUL/FAILED

2.4.8 trustedKeyMigrate

```
int trustedKeyMigrate(byte * migrationPassword)
```

Start of informative comment

This command migrates EFS key tree from one trusted platform/compartment to other one.

Note: WP05 Key migration and backup services are unspecified. Waiting for further clearance.

End of informative comment

Parameters:

`migrationPassword` - Migration password

Returns:

status flag SUCCESSFUL/FAILED

2.4.9 trustedUserKeyBackup

```
int trustedUserKeyBackup(String backupUrl,
                         byte* backupPasswd,
                         byte* userPasswd)
```

Start of informative comment

Trusted back up operation for user keys.

End of informative comment

Parameters:

backupUrl	- URL of the directory which user keys will be backed up(absolute url like /mnt/sda1/keys)
backupPasswd	- password for S2K encryption.
userPasswd	- password for keys which are going to be backed up.

Returns:

If back up URL is valid, user password is correct and backup password is not null it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --trusted-user-keys-backup /mnt/sdb1/foo/

Enter key password :

Enter key backup password :

Reenter key backup password :

User keys back up operation -> SUCCESSFUL/FAILED

2.4.10 trustedUserKeyRestore

```
int trustedUserKeysRestore(String restoreUrl,
                           byte* backupPasswd,
                           byte* userPasswd)
```

Start of informative comment

Trusted restore operation of user keys.

End of informative comment

Parameters:

`restoreUrl`	- URL of the directory which user keys will be restored. If null they will be restored to default directory. (absolute url like /mnt/sda1/keys)
`backupPasswd`	- password for S2K decryption.
`userPasswd`	- password for keys which are going to be restored.

Returns:

If back up URL is valid, user password is correct and backup password is not null it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --trusted-user-keys-restore

Enter key backup password :

Enter user password :

User keys restore operation -> SUCCESSFUL/FAILED

2.5 Recovery Agent Operations

2.5.1 sendRARegistrationRequest

```
int sendRARegistrationRequest(byte* adminPass)
```

Start of informative comment

Sends registration request to recovery server for this recovery agent.

End of informative comment

Parameters:

`adminPass` - Password of the administrative user.

Returns:

If administrative password is true and an established trusted channel exists, then it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --register-ra

Enter Administrative Password :

Request delivery SUCCESFUL/FAILED

2.5.2 sendRAUnregistrationRequest

```
int sendRAUnregistrationRequest(byte* adminPass)
```

Start of informative comment

Sends unregistration request to recovery server for this recovery agent.

End of informative comment

Parameters:

`adminPass` - Password of the administrative user.

Returns:

If administrative password is true and an established trusted channel exists, then it will return SUCCESSFUL.

Sample Usage in CLI:

```
$ efscli  --unregister-ra

Enter Administrative Password :
Request delivery SUCCESFUL/FAILED
```

2.5.3 recoveryAgentListUpdate

```
int recoveryAgentListUpdate(byte* adminPass)
```

Start of informative comment

Updates recovery agent list on registered Recovery Server if AIK credential is reissued manually.

End of informative comment

Parameters:

`adminPass` - Password of the administrative user.

Returns:

If administrative password is true and an established trusted channel exists, then it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --ra-list-update

Enter Administrative Password :

Request delivery SUCCESFUL/FAILED

2.6 Recovery Server Operations

2.6.1 distributeMACPolicies

```
int distributeMACPolicies(byte* adminPass)
```

Start of informative comment

Distributes current MAC policy to **all** registered Recovery Agents on Recovery Server.

End of informative comment

Parameters:

`adminPass` - Password of the administrative user in RS.

Returns:

If administrative password is true and an established trusted channel exists, then it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --policy-distribution newpolicyfile.pol

Enter Administrative Password :

Distribution SUCCESFUL/FAILED

2.6.2 distributeRAPrivileges

```
int distributeRAPrivileges(byte* adminPass)
```

Start of informative comment

Distributes or updates privilege levels assigned to individual trusted compartments or platforms which are represented by Recovery Agents

End of informative comment

Parameters:

`adminPass` - Password of the administrative user.

Returns:

If administrative password is true and an established trusted channel exists, then it will return SUCCESSFUL.

Sample Usage in CLI:

$ efscli --distribute-privileges newpriviledgefile.priv

Enter Administrative Password :

Distribution SUCCESFUL/FAILED

2.7 Administrative Operations

In order to complete administrative operations administrative user should have to be logged in RS. Credential checks for this administrative user will be done by RS.

2.7.1 listRegisteredPlatforms

```
Platform* listRegisteredPlatforms()
```

Start of informative comment

Lists registered platforms on Recovery Server.

This command is only valid in Recovery Server

End of informative comment

Returns:
List of registered platforms

Dependency:
Platform Class(see 3.1.1)

Sample Usage in CLI:

$ efscli --list-registered-platforms

RA id - Registration Date - Update Date - Meta

1 – 12.03.2007 – 17.03.2007 – PlatformLocationAndSomeOtherMetaData

.....

23 – 14.03.2007 – 17.03.2007 – PlatformLocationAndSomeOtherMetaData

2.7.2 distributeMACPoliciesToSpecified

`PlatformResults* distributeMACPoliciesToSpecified(int* targetPlatformIds,int length)`

Start of informative comment

Distributes policy file to the specified target platforms.

End of informative comment

Parameters:

`targetPlatformsIds`	- Platform ids which policies are going to be distributed
`length`	- number of target platforms

Returns:

Returns policy distribution results per platform, if specified target platform identifications are correct.

Dependency:

PlatformResults Class(see 3.1.2)

Sample Usage in CLI:

```
$ efscli  --policy-distribution-platforms newpolicyfile.pol --target-platforms 4,7,12
```

Distribution of policies to the platform 4 -> SUCCESFUL/FAILED

Distribution of policies to the platform 7 -> SUCCESFUL/FAILED

Distribution of policies to the platform 12 -> SUCCESFUL/FAILED

2.7.3 listRegistrationRequests

`Platform* listRegistrationRequests()`

Start of informative comment

Lists the platform which have sent registration request to Recovery Server

End of informative comment

Returns:
List of the platforms which have registration requests

Dependency:
Platform Class(see 3.1.1)

Sample Usage in CLI:

$ efscli --list-registration-requests

RA id / Operation Request

27 -> Registration

.....

32 -> Registration

2.7.4 listUnregistrationRequests

`Platform* listUnregistrationRequests()`

Start of informative comment

Lists the unregistration requests pending on Recovery Server

End of informative comment

Returns:
List of the platforms which have unregistration requests

Dependency:
Platform Class(see 3.1.1)

Sample Usage in CLI:

$ efscli --list-unregistration-requests

RA id / Operation Request

3 -> Unregistration

.....

15 -> Unregistration

2.7.5 permitRegistrationRequests

`PlatformResults* permitRegistrationRequests(int* platformIds, int length)`

Start of informative comment

Approval of registration requests

End of informative comment

Parameters:

`platformIds`	- ids of the platforms which are going to be approved for registration
`length`	- number of platform identification numbers

Returns:

Registration approval results per platform, if specified platform identification numbers are correct.

Dependency:

PlatformResults Class(see 3.1.2)

Sample Usage in CLI:

$ efscli --permit-registration-request --target-platforms 4,7,12

Plaform 4 - Registration Request Approval -> SUCCESFUL/FAILED

Plaform 7 - Registration Request Approval -> SUCCESFUL/FAILED

Plaform 12 - Registration Request Approval -> SUCCESFUL/FAILED

2.7.6 denyRegistrationRequests

`PlatformResults* denyRegistrationRequests(int* platformIds, int length)`

Start of informative comment

Denial of registration requests

End of informative comment

Parameters:

`platformIds`	- ids of the platforms whose registration requests are going to be rejected
`length`	- number of platform identification numbers

Returns:

Registration denial results per platform, if specified platform identification numbers are correct.

Dependency:

PlatformResults Class(see 3.1.2)

Sample Usage in CLI:

$ efscli --deny-registration-request --target-platforms 4,7,12

Plaform 4 - Registration Request Denial -> SUCCESFUL/FAILED

Plaform 7 - Registration Request Denial -> SUCCESFUL/FAILED

Plaform 12 - Registration Request Denial-> SUCCESFUL/FAILED

2.7.7 permitUnregistrationRequests

`PlatformResults* permitUnregistrationRequests(int* platformIds,int length)`

Start of informative comment

Approval of unregistration requests

End of informative comment

Parameters:

`platformIds`	- ids of the platforms which are going to be approved for unregistration
`length`	- number of platform identification numbers

Returns:

Unregistration approval results per platform, if specified platform identification numbers are correct.

Dependency:

PlatformResults Class(see 3.1.2)

Sample Usage in CLI:

$ efscli --permit-unregistration-request --target-platforms 4,7,12

Plaform 4 - Unregistration Request Approval -> SUCCESFUL/FAILED

Plaform 7 - Unregistration Request Approval -> SUCCESFUL/FAILED

Plaform 12 - Unregistration Request Approval -> SUCCESFUL/FAILED

2.7.8 denyUnregistrationRequests

`PlatformResults* denyUnregistrationRequests(int* platformIds, int length)`

Start of informative comment

Denial of unregistration requests

End of informative comment

Parameters:

`platformIds`	- ids of the platforms whose unregistration requests are going to be rejected
`length`	- number of platform identification numbers

Returns:

Unregistration denial results per platform, if specified platform identification numbers are correct.

Dependency:

PlatformResults Class(see 3.1.2)

Sample Usage in CLI:

$ efscli --deny-unregistration-request --target-platforms 4,7,12

Plaform 4 - Unregistration Request Denial -> SUCCESFUL/FAILED

Plaform 7 - Unregistration Request Denial -> SUCCESFUL/FAILED

Plaform 12 - Unregistration Request Denial-> SUCCESFUL/FAILED

2.7.9 listRecoveryRequests

`RecoveryList* listRecoveryRequests()`

Start of informative comment

Lists the recovery request that comes in Recovery Server

End of informative comment

Returns:
List containing recovery items

Dependency:
RecoveryList Class(see 3.2.1)

Sample Usage in CLI:

$ efscli --list-recovery

Recovery id / RA id / RA Meta / Operation / fileName

1 / 4 / PlatformLocationAndSomeOtherMetaData / Recovery / foo.txt.efs

2 / 34 / PlatformLocationAndSomeOtherMetaData / Recovery / bar.pdf.efs

2.7.10 permitRecoveryRequest

`RecoveryResults* permitRecoveryRequest(int* recoveryIds, int length)`

Start of informative comment

Approval of recovery requests

End of informative comment

Parameters:

`recoveryIds`	- ids of the recovery requests which are waiting for administrative approval
`length`	- number of platform identification numbers

Returns:

Recovery results per platform, if specified platform identification numbers are correct.

Dependency:

RecoveryResults Class(see 3.2.2)

Sample Usage in CLI:

```
$ efscli  --permit-recovery 2

Recovery request approved
File bar.pdf has been recovered
```

2.7.11 denyRecoveryRequest

`RecoveryResults* denyRecoveryRequest(int* recoveryIds, int length)`

Start of informative comment

Denial of recovery requests

End of informative comment

Parameters:

`recoveryIds`	- ids of the recovery requests which are waiting for administrative approval
`length`	- number of platform identification numbers

Returns:

Recovery denial results per platform, if specified platform identification numbers are correct.

Dependency:

RecoveryResults Class(see 3.2.2)

Sample Usage in CLI:

```
$ efscli  --permit-recovery 2

Recovery request denied
File bar.pdf could not be recovered
```

2.7.12 configureRS

```
int configureRS(String confFile)
```

Start of informative comment

Configures Recovery Server with the given configuration file.

End of informative comment

Parameters:

`confFile` - new configuration file(absolute url like /root/efsrs.conf)

Returns:

If configuration of RS has errors, function will yield FAILED. Else it will return SUCCESSFUL.

Sample Usage in CLI:

```
$ efscli  --rs-conf

RS Configuration update -> SUCCESFUL / FAILED
```

2.7.13 configureRSPolicy

```
int configureRSPolicy(String policyFile)
```

Start of informative comment

Configures policies in Recovery Server with the given policy file.

End of informative comment

Parameters:

`policyFile` - new policy file (absolute url like /root/efsrs.pol)

Returns:

If policy file of RS has syntax or semantic errors, function will yield FAILED. Else it will return SUCCESSFUL.

Sample Usage in CLI:

```
$ efscli  --rs-policy

RS Policy update -> SUCCESFUL / FAILED
```

2.7.14 configureRAPrivilege

```
int configureRAPriviledge(String privilegeFile)
```

Start of informative comment

Configures priviledges in Recovery Server with the given privilege file.

End of informative comment

Parameters:

`priviledgeFile` - new privileges file (absolute url like /root/efsrs.priv)

Returns:

If privilege file of RS has syntax or semantic errors, function will yield FAILED. Else it will return SUCCESSFUL.

Sample Usage in CLI:

```
$ efscli  --rs-priviledge

RS Priviledge Table update -> SUCCESFUL / FAILED
```

3 Appendices : Other Public Classes

These classes will be used as helper classes which can be used both in delivering and formatting the command output.

3.1 Appendix A : Platform Classes

3.1.1 Platform

Platform class helps keeping and processing information about the platforms that are registered to RS or have registration request to RS. This class will be used to output platform data which is easy to format.

Constructor :

```
Platform(int id,
                String registrationDate,
                String lastUpdateTime,
                String metaInformation)
```

Creates a platform object with given information.

Parameters:

`id` - - identification number of the platform
`registrationDate` - - registration date of the platform
`lastUpdateTime` - - Last list update time of the platform.
`metaInformation` - - Other information like physical place and users

Getters :

```
int getId()

String getLastUpdateTime()

String getMetaInformation()

String getRegistrationDate()
```

Setters :

```
void setId(int id)

void setLastUpdateTime(String lastUpdateTime)

void setMetaInformation(String metaInformation)

void setRegistrationDate(String registrationDate)
```

Table 9:Platform Class Declaration

3.1.2 PlatformResults

A helper class for displaying results of platform RA registration and unregistration requests to RS.

Constructor :

```
PlatformResults(Platform platforms,
                      int result)
```

Creates a PlatformResults object which can keep the results of operations like permitRegistrationRequest etc.

Parameters:

`platforms` – platform that send the request
`result` – result of the registration/unregistration request

Getters :

```
Platform getPlatforms()

int getResult()
```

Setters :

```
void setPlatforms(Platform platforms)

void setResult(int result)
```

Table 10:PlatformResults Class Declaration

3.2 Appendix B : Recovery Classes

3.2.1 RecoveryList

Helps keeping and listing recovery requests which arrives RS.

Constructor :

```
RecoveryList(int recoveryId,
                    Platform platform,
                    String filename)
```

Helps keeping recovery request lists.

Parameters:

recoveryId - identification number of the recovery request
platform - platform which the recovery request comes from
filename - name of the file which is going to be recovered

Getters :

```
String getFilename()

Platform getPlatform()

int getRecoveryId()
```

Setters :

```
void setFilename(String filename)

void setPlatform(Platform platform)

void setRecoveryId(int recoveryId)
```

Table 11:RecoveryList Class Declaration

3.2.2 RecoveryResult

Helps keeping and listing results of recovery requests which arrives RS.

Constructor :

```
RecoveryResults(int recoveryId,
                      RecoveryList list,
                      int result)
```

Keeps the result of remote recovery opration.

Parameters:

recoveryId - identification number of the recovery id
list - list which keeps Recovery request
result - result of the recovery process

Getters :

```
RecoveryList getList()

int getRecoveryId()

int getResult()
```

Setters :

```
void setList(RecoveryList list)

void setRecoveryId(int recoveryId)

void setResult(int result)
```

Table 12:RecoveryResult Class Declaration

3.3 Appendix C : Key Information Class

3.3.1 KeyInfo

Helps keeping and listing user keys which is in user's keystore.

Constructor :

```
KeyInfo(int keyid,
              String keyName,
              byte* keyHash)
```

Creates a KeyInfo object for listing keys in he desired format.s

Parameters:

keyid - identification number of the key
keyName - name of the key
keyHash - hash value of the key

Getters :

```
byte* getKeyHash()

int getKeyid()

String getKeyName()
```

Setters :

```
void setKeyHash(byte* keyHash)

void setKeyid(int keyid)

void setKeyName(String keyName)
```

Table 13:KeyInfo Class Declaration