D6.1: Collection of all SWP deliverables (with nature=R) produced during months 1-12

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

Deliverable reference number	IST-027635/D6.1/PUBLIC | 1.10
Deliverable title	Collection of all SWP deliverables (with nature=R) produced during months 1-12
WP contributing to the deliverable	WP6
Due date	Apr 2007
Actual submission date	Apr 2007

Responsible Organisation	POL
Authors	See the cover page of each included internal deliverable
Abstract	Collection of all internal deliverables (with nature=R) produced during months 1-12: D06a.1: Preliminary DRM system specification D06e.1: MFA Requirements and Specification D06e.3: Intermediate MFA System Specification NOTE: a general explanation section has been added and sections 1, 3.6.1, 3.6.2 of D06a.1 have been updated
Keywords	

Dissemination level	Public
Revision	PUBLIC | 1.10

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

1 Introduction

This section has been added on request from the EC reviewers. Its purpose is to clarify the relationships between Workpackage 2 and Workpackage 6, namely how the requirements set in the former have impact on the latter. In particular this section explains which elements from Trusted Computing (TC) technology are used in WP6 to meet the requirements specified in WP2.

2 Workpackage 6's context and purpose

WP6 is related to applications that make use of Trusted Computing and virtualisation technologies to reach security goals difficult to achieve using other technologies or obtainable at a higher price. The foundations of such technologies are hardware and software components: (1) the Trusted Platform Module (TPM) which is a low cost chip shipped with most of recent platforms together with the (2) new generation of Intel and AMD processors and chipsets that provide hardware assistance for virtualisation and hardware support for the platform integrity protection and measurement and (3) open source Virtual Machine Monitors (VMM) like Xen and Fiasco, an implementation of L4 micro-kernel family.
Since the core workpackages (WP3 to WP5) are in charge to design and develop a security framework (OpenTC) based on the mentioned technologies, WP6's purpose is to show which security benefits can be achieved using such framework in different application areas.

Moreover two of five WP6 applications are special cases: Encryption File Service (EFS, in Sub-workpackage 6.d) and MultiFactor Authentication (MFA, Sub-workpackage 6.e) are applications that show the use of the technology but they can be also considered as components to be shipped with the framework.

3 Use of TC technology in Sub-workpackages 6.a and 6.e

3.1 *SWP06.a: Interoperable DRM solution based on MPEG-21*

This subsection applies to the following internal deliverable:

– D06a.1 Preliminary DRM system specification (included in main deliverable D6.1)

The objective of this Sub-workpackage is to design and develop a DRM system capable to balance the rights and obligations of all involved stakeholders (i.e. the end user and the content provider) to support a more fair use of DRM. Some relevant requirements for the stakeholders are:

– (content provider) guarantee that the user's system will play the multimedia content only in accordance with the terms of the license

– (user) support for user rights for playing the content on different devices, sharing the content among a limited number of close users and re-selling the content

– (user) user is in control of his platform: no hidden software needs to be installed by the content provider to protect the content from unauthorized uses

One key element to meet these requirements is the support for interoperable (i.e. standard) formats for both encrypted contents and licenses. Instead, the technological elements provided by OpenTC framework to satisfy the mentioned requirements are:

- open design and open source system: this allows the inspection by the users' community to verify that hidden components are not present (see for example the case of like the Sony rootkit)
- TPM on the user platform for collecting the integrity measurements of the platform and software components; these measurements will be used for
    - remote attestation performed by the content provider to check the integrity of the user's platform before sending the decryption keys
    - secure storage for keys and other relevant data which are bound to a good status of the user's platform (sealing), represented by specific sets of measurements
- isolation provided by virtualisation for separating the DRM code and the player into protected containers, to guarantee that license parsing and content decryption are performed separately in secure environments
- cryptographic services and support for PKI

For further details about the technical requirements from other workpackages, namely properties and services provided by OpenTC framework to this application, see main deliverable D02.2 (the updated version of D02.1), section 11.2.1.

3.2 SWP06.e: MultiFactor Authentication (MFA)

This subsection applies to the following internal deliverables:

- D06e.1 MFA Requirements and Specification (included in main deliverable D6.1)
- D06e.2 MFA Intermediate Specification (included in main deliverable D6.1)
- D06e.3 MFA Concept Prototype (included in main deliverable D6.2)

The objective of this Sub-workpackage is to design and develop a MultiFactor Authentication (MFA) system for client-server applications, namely a system that provides an additional factor for making the authentication stronger when accessing a service through a network.

This is an application of TC technology but not an end-user one: it is instead a subsystem that can be integrated in other systems or frameworks like OpenTC to enhance the robustness of the authentication.

For instance, the usual password-based authentication scheme for a user, cannot be considered robust, even though performed through a robust protocol. In fact the passwords can be stoled through a social engineering attack no matter which is the robustness of the authentication system. MFA can be used to increase the

The requirements for the parties involved in a networked interaction (client and server) for having a more robust authentication mechanism are

- the use of an additional factor based on the TPM to implement the platform authentication;

  - only users from registered client platforms can access a service
  - the user can can verify the identity integrity of the platform providing the service
- seamless integration into existing systems and authentication subsystems

MFA is designed to be a general purpose subsystem that can be used with a variety systems. However there are specific uses for OpenTC. Given the the high level of assurance that OpenTC should provide, MFA can be used as component of OpenTC framework for allowing its remote management only from a registered workstations. However it can also be used completely contained within an OpenTC framework running on single physical platform for accessing network services provided by some Virtual Machines to other Virtual Machines.

The technological elements provided by OpenTC framework to satisfy such requirements are:

- TPM on the client for performing the signature over a random challenge with a protected key to authenticate the user's platform
- TPM on the server for locally encrypting the MFA policy database and bind it to a good status of the platform (sealing)
- isolation provided by virtualisation for separating the security critical service for remote management service - for instance sshd for management from a remote console - which is security critical from the other applications and services

For further details about the MFA see D02.2 (the updated version of D02.1), section 11.6.

The Concept Prototype (see internal deliverable D6e.2) is an example to experiment and show that is possible to add a further authentication based on the TPM to an existing network applications (like ssh/sshd) and authentication framework (like Linux PAM). Therefore only part of the planned features/capabilities have been included in this preliminary prototype: for instance the mutual authentication has not been implemented.

The final prototype, instead will be a featured component that will provide an API for remote access to the service; the preliminary specification of such API has been as “MFA intermediate specifications” (see internal deliverable D6e.3).

4 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**
API	Application Programming Interface
DRM	Digital Right Management
MFA	MultiFactor Authentication
PKI	Public Key Infrastructure
WP	Workpackage
SWP	Sub-Workpackage
TC	Trusted Computing

5 Referenced Documents

/1/ 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.

/2/ OpenTC: D02.2 Requirements Definition and Specification (April 2007)

WP06a Preliminary 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.1/FINAL | 1.10
Deliverable title	Preliminary DRM system specification
WP contributing to the deliverable	WP6
Due date	Apr 2007
Actual submission date	Apr 2007

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

Dissemination level	Public
Revision	FINAL | 1.10

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 preliminary 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.

The formats for the data exchanged between the external modules, like the between the OpenTC Player and DRM Core, have not yet been defined. Moreover the interactions with the services provided by OpenTC framework are just outlined and exposed in terms of requirements, since the definition of the interfaces to those services is still an ongoing activity within the workpackages three to five. The exact details of these interactions and definition of the formats for the data exchanged between the components of the DRM system will be included in the final specifications.

The principal scope of the OpenTC 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 OpenTC 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 and 4 include the design specifications of the system. Section 5 shows the requirements for this application within the OpenTC System. Section 6 describes a preliminary API used to access the DRM Core, and its key functions. Section 7 is a normative API definition in C++, while sections 8 and 9 provide glossary and references to the terms and technologies used in the DRM system.

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 is an operating system component 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 secure application 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 actions:

– Initialization of the system
– Download content
– View / Play 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	Initialization of the system
Description	The administrator initializes the DRM System within the OpenTC framework.
Actors	Administrator
Preconditions	The OpenTC framework was started.
Postconditions	The DRM Core was initialized.
Comment	
Normal Flow	1. The Administrator installs a DRM-Core and starts it in a separate compartment. 2. The Administrator installs user space applications, that can be executed in the secure environment for rendering of the content. 3. The administrator 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 and the TSS Stack. 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
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 initializes 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 starts a mutual authentication round. The keys may only be transferred when the existence of a trusted DRM-core within a secure environment is detected at the receiver side, thus the download procedure may start after a successful authentication. The authentication is planned to be based on remote attestation and generation of attestation identity keys (AIKs) by the TPM. 6. The DRM Core exchanges the content keys. The keys are managed by the DRM-core, which stores them in a central key store along with a Digital Item Identifier (DII). The same identifier resides in

	the license within the content container. The content key download procedure may be separate to the content download procedure, but always takes place under the DRM-core's control. 7. The license for the content is checked and preprocessed. The important information of the license is stored in the sealed store.

Use Case Unique ID	/ UC 30 /
Title	View / Play content
Description	The user tells the player that he wants to view 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 license from the container file along with the requested action (e.g. play, print, modify 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 4.3. The following sections will explain the individual components of the system in detail.

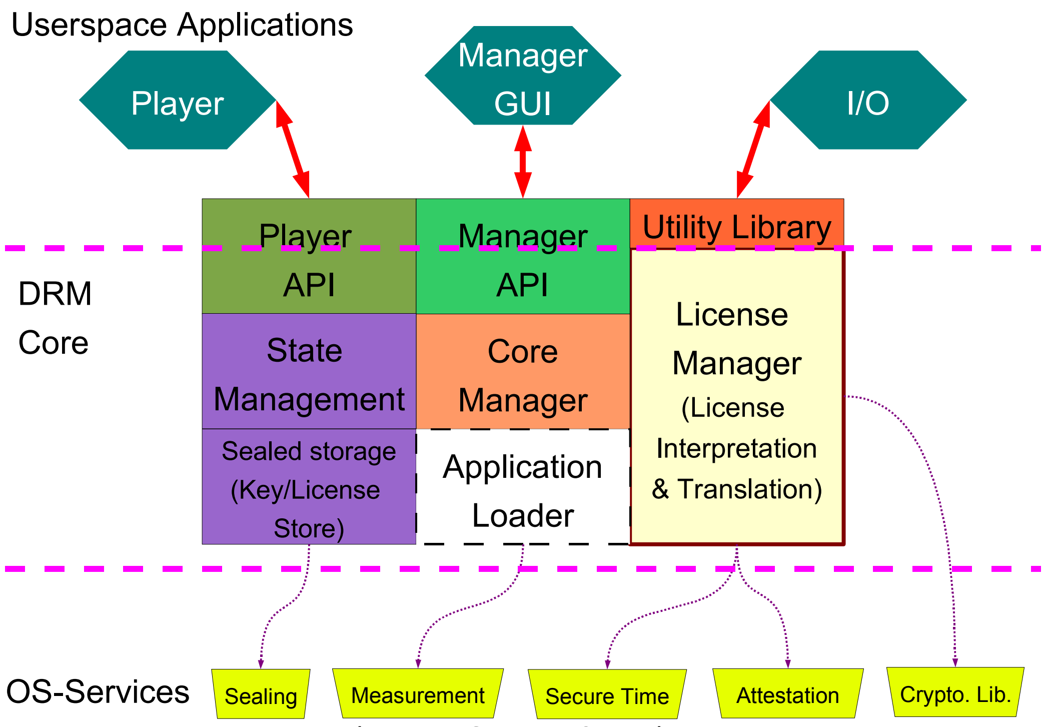

Figure 1: System Overview

3.2 *Player API*

3.2.1 *Registration*

Each Player who wants to access a protected content must register with the DRM Core first. During Registration a mutual authentication is performed, so that the Player and DRM Core validate the integrity and security status of the other party.

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. The specific command set is not defined in this preliminary document.

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

3.2.2 Content Key Handling

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 standardised 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. Legacy players, which cannot access the API directly, are also supported by our architecture. Players of that kind are not aware of the DRM Core, but maybe favored by users for whatever reason. These cases are handled by an IO-Socket interface, in which 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 only has to support the content’s type and be connected to the IO-Socket through a plug-in. The player receives the unprotected content from the socket and can render it. The IO-socket 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, such as inserting new licenses into the DRM Core and requesting for attestation keys. The Manager GUI uses this API (defined in `ManagementInterface`), 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 4.3.1 and 4.3.2.

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 e.g. 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 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, meaning that the resulting systems will be beneficial not only for the content provider, but also the consumer of a content. It is thus planned that all participants in the system 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 has to be defined beforehand, with a specified maximum number of devices.

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.

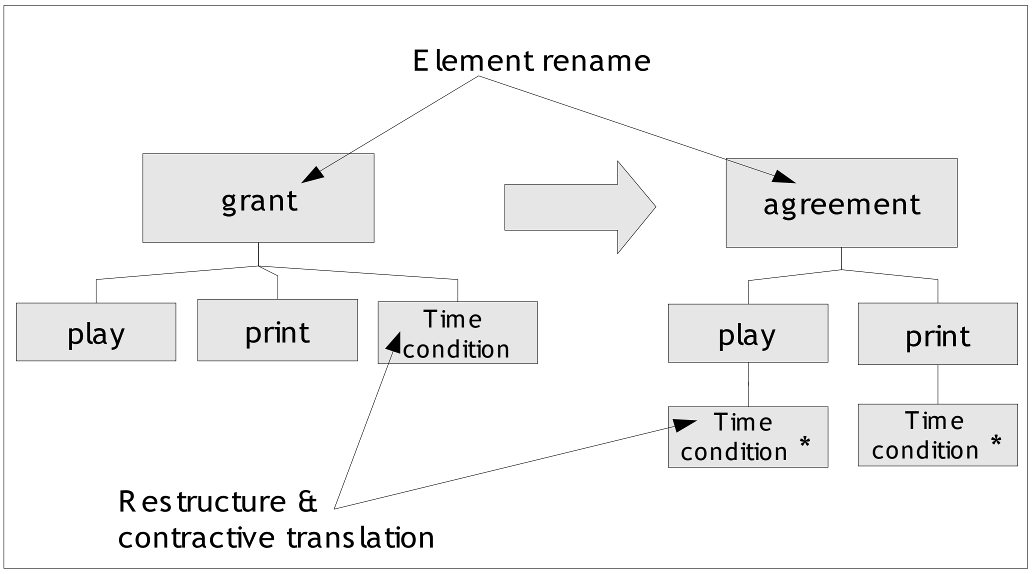

Figure 2: License Translation

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* store

The sealed store consists of two parts: the key store and the license 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 organised 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.

3.9 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 no security weakness is exposed through the new tool.

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 OS.

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

4. Component Interaction

4.1 Functional parts of the DRM Core

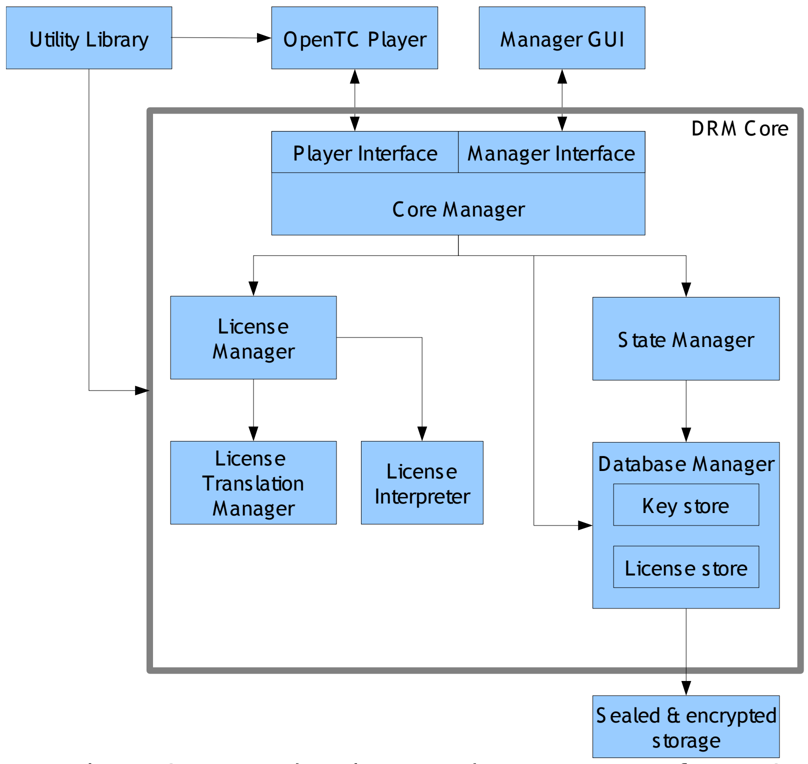

Figure 3: Internal and external components of DRM Core

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 3 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.

4.2 Sequence diagram

Figure 4 shows the sequence diagram for interaction between the player and different components within the DRM Core. The player application first performs an initial

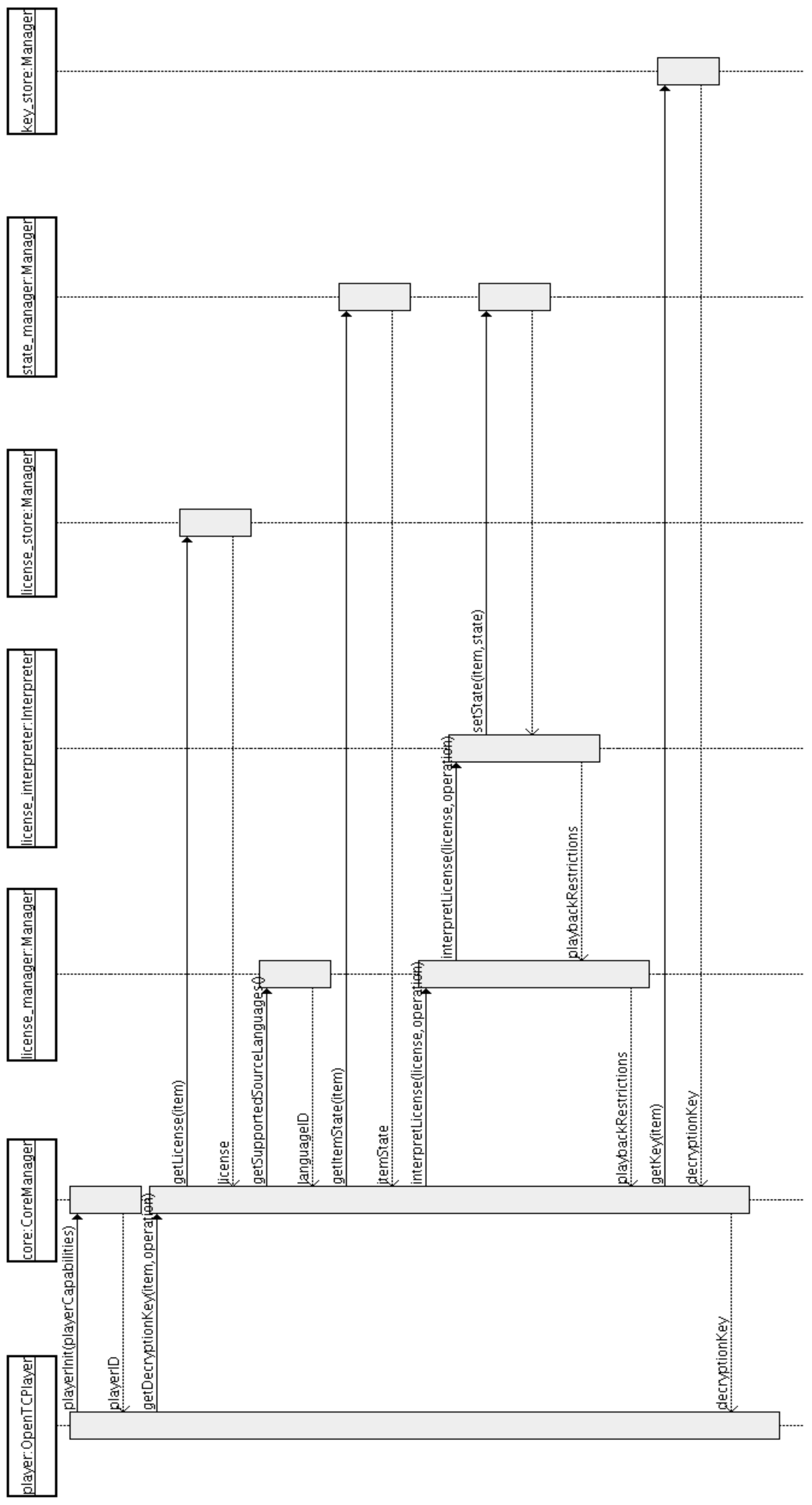

Figure 4: Sequence diagram for media playback

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.

4.3 DRM system and XEN/L4 virtualization environments

4.3.1 DRM components and compartments

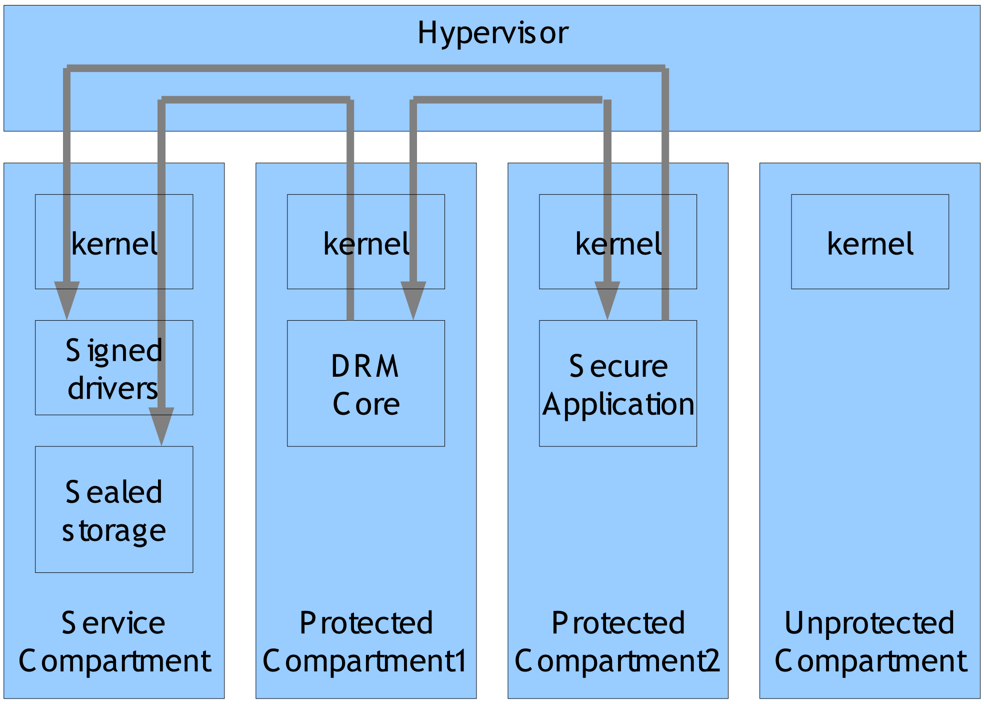

Figure 5: Virtualization of the DRM Core

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 5 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. Sealed storage protects the key and license stores described in sections 3.8.1 and 3.8.2.

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.

4.3.2 *Interfaces between compartments*

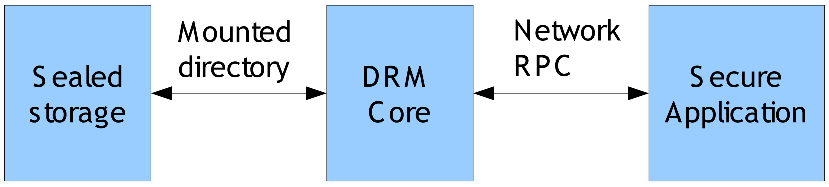

Figure 6: Interface Chain

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. This RPC standard will not be defined in this preliminary specification, but will be based on a standard RPC protocol, with a similar API as described in section 3.2.

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. 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.
- **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.
    - Sealed Storage. The DRM Core will use sealed storage for its license and key databases.
    - 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 of whatever kind. It would be beneficial if the underlying system can provide a system-wide policy management facility, so that DRM-related configuration can be seamlessly integrated into the management tool.

6. Technical API Specification

This section describes the API of critical classes and interfaces within the DRM Core. This specification is based on the C++ language. Namespace used for the project is defined to be `de_tum_ldv_opentc`.

6.1 External Interfaces

Class interfaces which are to be accessed / called from outside the DRM Core are defined in this section.

6.1.1 class PlayerInterface

This class provides the main external interface to the OpenTC player. It is defined `virtual` and is implemented by the `CoreManager` class.

6.1.1.1 playerInit()

Function: Initialize the player with an XML description of the `playercapabilities`

Parameter:

Name	**Type**	**Description**
`playerCapabilities`	`string`	XML string describing player capabilities

Return value:

	Type	**Description**
	`int`	integer identifier representing the particular player

6.1.1.2 getDecryptionKey()

Function: Get content key from the DRM Core. This is the main operation used by the player application to inform the DRM Core of a start of an digital item operation (e.g. "Play") and to obtain the appropriate content key.

Parameter:

Name	**Type**	**Description**
`playerID`	`int`	integer identifier representing the particular player from `playerInit()`
`item`	`ItemReference`	Reference to digital item

Return value: *none*

6.1.1.3 getSupportedREL()

Function: Query the DRM Core for supported REL languages

Parameter: *none*

Return value:

	Type	**Description**
	`list<REL>`	List of languages supported by the Core

6.1.2 class ManagementInterface

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

6.1.2.1 `insertLicense()`

Function: insert a signed license from a mutually attested source into the DRM Core for a particular digital item

Parameter:

Name	**Type**	**Description**
`license`	`string`	XML string of license
`relType`	`REL`	REL language identifier
`item`	`ItemReference`	Reference to digital item

Return value: *none*

6.1.2.2 `getAttestationKey()`

Function: Obtain an attestation key to download trusted content from a trusted source

Parameter:

Name	**Type**	**Description**
`playerID`	`int`	integer identifier representing the particular player from `playerInit()`
`relType`	`REL`	REL language identifier
`serverURL`	`string`	URL string of attestation server

Return value:

Type	**Description**
`Key`	Attestation key

6.1.3 class Utility

This is a utility class which provides a common mechanism for downloading tools for commonly used functions, such as decryption and decoding, for both the DRM Core and player.

6.1.3.1 `getTool()`

Function: Function to retrieve a specific tool for a specific platform and application, using a unique XML tool description

Parameter:

Name	**Type**	**Description**
`toolDescription`	`string`	XML tool description (based on MPEG-IPMP)

Return value:

Type	**Description**
`function ptr`	pointer to tool function

6.2 Internal Interfaces

This section defines class interfaces used between important modules within the DRM Core, supporting a more modular approach.

6.2.1 Class DatabaseManager

This is an internal interface used to connect to the secure database supporting the DRM Core. It is used for persistent secure storage of licenses, states (of digital items) and content keys for the respective digital items.

6.2.1.1 `getLicense()`

Function: Retrieve license for a particular digital item

Parameter:

Name	**Type**	**Description**
`item`	`ItemReference`	Reference to digital item

Return value:

Type	**Description**
`License`	License object

6.2.1.2 `setState()`

Function: Save state for a particular digital item

Parameter:

Name	**Type**	**Description**
`item`	`ItemReference`	Reference to digital item
`state`	`ItemState`	Item state to modify

Return value: *none*

6.2.1.3 `deleteLicense()`

Function: delete license for a particular digital item (and all corresponding states and keys)

Parameter:

Name	**Type**	**Description**
`item`	`ItemReference`	Reference to digital item

Return value: *none*

6.2.1.4 `getItemState()`

Function: Retrieve the state for a particular digital item

Parameter:

Name	**Type**	**Description**
`item`	`ItemReference`	Reference to digital item

Return value:

Type	Description
`ItemState`	Item state retrieved

6.2.1.5 `getDecryptionKey()`

Function: Retrieve content key for a particular digital item

Parameter:

Name	Type	Description
`item`	`ItemReference`	Reference to digital item

Return value:

Type	Description
`Key`	content key

6.2.2 Class InterpreterInterface

This is an internal interface used for every module which works as a license interpreter (to parse and decide if a player is allowed to play, and under what conditions) This is implemented by the `MPEG21Interpreter` class which provides interpretation for MPEG-21 licenses.

6.2.2.1 `interpretLicense()`

Function: Interpret license from a generic license object, given the entire authorization request parameters. When positively authorized, a player restriction description is returned as string, otherwise an exception is raised.

Parameter:

Name	Type	Description
`licenseGroup`	`list<License>`	List of all license objects to interpret together
`item`	`ItemReference`	Reference to digital item
`state`	`ItemState`	Item state
`system`	`SystemState`	System state
`operation`	`string`	Operation taken on Digital Item

Return value:

Type	Description
`string`	XML string of playback restriction description

6.2.2.2 `getSupportedLanguage()`

Function: Query the interpretation module for the supported REL language in run-time.

Parameter: none

Return value:

Type	Description
`REL`	supported REL language

6.2.3 class SystemState

This class is an extraction of the possible system states, such as secure time, platform and owner credentials.

6.2.3.1 `checkSysteState()`

Function: Query a particular system state

Parameter:

Name	Type	Description
`state`	`string`	String ID of state to retrieve

Return value:

Type	Description
`string`	Value of system state

6.2.4 class CoreManager

This class is inherited from `PlayerInterface` and `ManagementInterface`, and implements the `PlayerInterface` and `ManagementInterface`. It is the main entry point for both the OpenTC player and Manager GUI.

6.2.5 class LicenseManager

This is a manager class to a license management module, which handles interpretation and translation of licenses, in different REL languages.

6.2.5.1 `interpretLicense()`

Function: Interpret the license given the full authorization request parameters.

Parameter:

Name	Type	Description
`licenseGroup`	`list<License>`	List of all license objects to interpret together
`item`	`ItemReference`	Reference to digital item
`operation`	`string`	Operation taken on Digital Item

Return value:

Type	Description
`string`	XML string of playback restriction description

Exceptions:

Type	Description
`ERROR_LICENSE_INVALID`	License is invalid
`ERROR_OPERATION_DISALLOWED`	Operation is completely not allowed

6.2.5.2 `translateLicense()`

Function: Translate the license into another REL language, with a restriction description string (describing the capabilities of the target player)

Parameter:

Name	Type	Description
`sourceLicense`	`License`	Input license object to be translated
`targetLicenseLang`	`REL`	REL language to translate to
`targetRestrictions`	`string`	XML string of output license

Return value:

Type	Description
`License`	License object after translation

6.2.5.3 `getSupportSourceLanguages()`

Function: Query for all supported input REL languages

Parameter: none

Return value:

Type	Description
`REL`	supported input REL language

6.2.5.4 `getSupportTargetLanguages()`

Function: Query for all supported output REL languages

Parameter: none

Return value:

Type	Description
`REL`	supported output REL language

6.2.6 class LicenseTranslationManager

This is a manager class to handle translation of licenses to different REL languages, under given constrains.

6.2.6.1 `translateLicense()`

Function: Translate license to another supported REL language

Parameter:

Name	Type	Description
`sourceLicense`	`License`	Input license object to be translated
`targetLicenseLang`	`REL`	REL language to translate to
`targetRestrictions`	`string`	XML string of output license

Return value:

Type	Description
`License`	License object after translation

6.2.6.2 `getSupportSourceLanguages()`

Function: Query for all supported REL languages

Parameter: none

Return value:

Type	Description
`list<REL>`	List of all supported source REL languages

6.2.7 class StateManager

This is a manager class to handle user states(limits for operations) for each digital items.

6.2.7.1 `checkLimit()`

Function: Retrieve limit for a particular operation on a particular digital item

Parameter:

Name	Type	Description
`item`	`ItemReference`	Reference to digital item
`operation`	`string`	Operation taken on Digital Item

Return value:

Type	Description
`int`	limit for operation

6.2.7.2 `decrementLimit()`

Function: Decrease limit for a particular operation on a particular digital item

Parameter:

Name	Type	Description
`item`	`ItemReference`	Reference to digital item
`operation`	`string`	Operation taken on Digital Item

Return value:

Type	Description
`int`	new limit after decrement

6.2.7.3 `getSystemState()`

Function: Get a particular system state

Parameter:

Name	Type	Description
`state`	`string`	String ID of state to retrieve

Return value:

Type	Description
`string`	value of state

6.3 Helper Classes

This section defines commonly used helper classes.

6.3.1 *class ItemReference*

Class to encapsulate the reference to a particular digital item. Contains the fields:

- `int DI`
- `string URL`

6.3.2 *class ItemState*

Class to encapsulate a single state object

- `int item_ID`
- `string state`

6.3.3 *class Key*

Class to represent a content key of generic type

6.3.3.1 `Key()`

Function: Construct the key object of a particular type and length

Parameter:

Name	**Type**	**Description**
`keyLength`	`int`	Length of key in bytes
`keyType`	`int`	integer key type based on the KeyType enumeration definition. (See 6.3.3)
`key`	`char*`	binary data to store in Key object

6.3.3.2 `getKey()`

Function: Retrieve binary data of the key

Parameter: none

Return value:

Type	**Description**
`char*`	binary data of Key

6.3.3.3 `getKeyType()`

Function: Retrieve the key type

Parameter: *none*

Return value:

Type	**Description**
`int`	integer representing key type based on the KeyType enumeration definition. (See 6.4.3.4)

6.3.3.4 `KeyType enumeration`

Name	Description
KEY_DES64	DES 64-bit key
KEY_AES128	AES 128-bit key
KEY_AES256	AES 256-bit key

6.3.3.5 `getLength()`

Function: Retrieve the key length

Parameter: *none*

Return value:

Type	Description
`int`	length of key (in bytes)

6.3.4 *class License*

Class to represent licenses of generic type

6.3.4.1 `License()`

Function: Construct the license object

Parameter:

Name	Type	Description
`rel`	`REL`	REL language of license
`text`	`string`	XML string of license

6.3.4.2 `isRELType()`

Function: Check if license is of a particular type

Parameter:

Name	Type	Description
`type`	`string`	Name of REL language to check with

Return value:

Type	Description
`string`	pointer to tool function

6.3.4.3 `getString()`

Function: Retrieve string representation of license

Parameter: *none*

Return value:

Type	Description
`string`	string representation of license

6.3.5 *class PlayerState*

Class to store data related to player initialized and authenticated by the DRM Core

6.3.5.1 `PlayerState()`

Function: Construct the object

Parameter:

Name	**Type**	**Description**
`id`	`int`	integer identifier representing the particular player from `playerInit()`

6.3.5.2 `getPlayerID()`

Function: Retrieve the player ID issued to the player

Parameter: *none*

Return value:

Type	**Description**
`int`	integer identifier representing the particular player from `playerInit()`

6.3.5.3 `getPlayerState()`

Function: Retrieve a particular player state

Parameter:

Name	**Type**	**Description**
`state`	`string`	player state to retrieve

Return value:

Type	**Description**
`string`	value of player state

6.3.5.4 `setPlayerState()`

Function: Set a particular player state

Parameter:

Name	**Type**	**Description**
`state`	`string`	player state to modify
`value`	`string`	value of player state

Return value: *none*

6.3.6 class REL

Class to represent a particular REL language type.

6.3.6.1 `REL()`

Function: Constructor using a string descriptor

Parameter:

Name	**Type**	**Description**
`language`	`string`	Name of REL language

6.3.6.2 `getName()`

Function: Retrieve string name of REL language

Parameter: *none*

Return value:

Type	**Description**
`string`	Name of REL language

7. Annex A : C++ Header definition

```
class PlayerInterface {

public:
            virtual int   playerInit(const std::string& playerCapabilities) = 0;
            virtual void getDecryptionKey(int playerID, const ItemReference& item,
            const std::string& operation, std::string& playbackRestriction,Key& key) = 0;
            virtual const std::list<REL> getSupportedREL() = 0;
};

class ManagementInterface {
            virtual void  insertLicense(const std::string& license, const REL& relType,
            const ItemReference& item) = 0;
            virtual void getAttestationKey(int playerID, const REL& relType,
            const std::string& serverURL, Key& key) = 0;//TODO - change Key&
};

class Utility {
            public:
            void* getTool(std::string toolDescription);
};

class DatabaseManager {
public:
            DatabaseManager();
            const License getLicense(const ItemReference& item);
            void setLicense(const ItemReference& item,const License& license,const ItemState&
state);
            void setState(const ItemReference& item,const ItemState& state);
            void deleteLicense(const ItemReference& item);
            ItemState getItemState(const ItemReference& item);
            const Key getDecryptionKey(const ItemReference& item);
};

class InterpreterInterface{
public:
            virtual std::string interpretLicense(std::list<License> licenseGroup,
            const ItemReference& item, const ItemState& state,const SystemState& system,
            const std::string& operation) = 0;
            virtual const REL& getSupportedLanguage()  = 0;
};

class SystemState {
            std::string checkSysteState(const std::string& state);
};

class CoreManager : public PlayerInterface, public ManagementInterface  {
            LicenseManager licManager;
            DatabaseManager dbManager;
public:
            CoreManager();
            int   playerInit(const std::string& playerCapabilities);
            void insertLicense(const std::string& license, const REL& relType,
            const ItemReference& item);
            void getDecryptionKey(int playerID, const ItemReference& item,
            const std::string& operation, std::string& playbackRestriction,Key& key)
            throw ( std::exception );
            void getAttestationKey(int playerID, const REL& relType,
            const std::string& serverURL, Key& key);
            const std::list<REL> getSupportedREL();
};

class LicenseManager{
            MPEG21Interpreter mpeg21Interpreter;
public:
            std::string interpretLicense(const std::list<License>& licenseGroup,
            const ItemReference& item, const std::string& operation)
            throw(ERROR_LICENSE_INVALID,ERROR_OPERATION_DISALLOWED);
            License translateLicense(License sourceLicense, const REL& targetLicenseLang,
            const std::string& targetRestrictions);
            const REL& getSupportSourceLanguages();
            const REL& getSupportTargetLanguages();
};

class LicenseTranslationManager{
public:
            License translateLicense(License sourceLicense, const REL& targetLicenseLang,
            const std::string& targetRestrictions);
            const std::list<REL> getSupportSourceLanguages();
};

class StateManager {
            int checkLimit(const ItemReference& item, const std::string& operation);
            int decrementLimit(const ItemReference& item, const std::string& operation);
            std::string& getSystemState(std::string state);
};

class ItemReference {
public:
            int DI;
            std::string URL;
};

class ItemState {
public:
            int item_ID;
            std::string state;
};

class Key{
private:
            int keyLength;
            int keyType;
            const char *data;
public:
            Key(int keyLength, int keyType, const char *key);
            const char* getKey() const;
            int getKeyType() const;
            int getLength() const;
            enum { KEY_DES64, KEY_AES128 , KEY_AES256 };
};

class License{
            private:
            std::string RELtext;
            REL relType;
public:
            License(const REL& rel, const std::string& text);
            bool isRELType(std::string type) const;
            bool isRELType(REL type) const;
            const std::string& getString() const;
};

class PlayerState {
private:
            int playerID;
public:
            PlayerState(int id);
            int getPlayerID();
            std::string checkPlayerState(const std::string& state);
};

class REL {
            std::string RELName;
public:
            REL();
            REL(std::string language);
            const std::string& getName() const;
};
```

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

D06e.1 MFA Requirements and Specification

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

Deliverable reference number	IST-027635/D06e.1/FINAL | 1.00
Deliverable title	MFA Requirements and Specification
WP contributing to the deliverable	WP06e.1
Due date	Apr 2006 - M06
Actual submission date	Sept 12, 2006

Responsible Organisation	INTEK
Authors	Irina Beliakova, Vladimir Bashmakov
Abstract	The goal of this application is to demonstrate the secure access to remote server that in addition to “what-you-know” information (password) requires the platform authentication through the use of TPM
Keywords	MFA, OPEN_TC, TPM

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

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.

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.

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

This document contains the MFA requirements and high level architecture specification for a preliminary release of the MFA system.

2 Threat analysis

Different threat types could be categorized by the general attacker goal:

- **Spoofing**. *Spoofing* is the attempt to gain access to a system by using a false identity. This can be accomplished by using stolen user credentials or a false IP address. After the attacker successfully gains access as a legitimate user or host, the elevation of privileges or abuse using authorization can begin.
- **Tampering**. *Tampering* is the unauthorized modification of data, for example as it flows over a network between two computers.
- **Repudiation**. *Repudiation* is the ability of users (legitimate or otherwise) to deny that they performed specific actions or transactions. Without adequate auditing, the repudiation attacks are difficult to prove.
- **Information disclosure**. *Information disclosure* is the unwanted exposure of private data. For example, a user views the contents of a table or file he or she is not authorized to open or monitors data passed in a plain text over a network. Some examples of information disclosure vulnerabilities include the use of hidden form fields, comments embedded in Web pages that contain database connection strings and connection details and weak exception handling that can lead to internal system level details being revealed to the client. Any of this information can be very useful to the attacker.
- **Denial of service**. *Denial of service* is the process of making a system or application unavailable. For example, a denial of service attack might be accomplished by bombarding a server with requests to consume all available system resources or by passing it malformed input data that can make an application process crash.
- **Elevation of privilege**. *Elevation of privilege* occurs when a user with limited privileges assumes the identity of a privileged user to gain privileged access to an application. For example, an attacker with limited privileges might elevate his or her privilege level to compromise and take control of a highly privileged and trusted process or account.

Each threat category has a corresponding set of countermeasure techniques that should be used to reduce the risk of attacks. These countermeasures are summarized in the following table:

Threat	Typical Countermeasures
Spoofing the user identity	● Use strong authentication ● Do not store secrets in plain text ● Do not pass credentials in plain text over the wire authentication ● Protect authentication cookies with Secure Sockets Layer (SSL)
Tampering with data	● Use data hashing and signing ● Use digital signatures ● Use strong authorization ● Use tamper-resistant protocols across communication links ● Secure communication links with protocols that provide message integrity
Repudiation	● Create secure audit trails ● Use digital signatures
Information disclosure	● Use strong authorization ● Use strong encryption ● Secure communication links with protocols that provide message confidentiality ● Do not store secrets (for example, passwords) in plain text
Denial of service	● Use resource and bandwidth throttling techniques ● Validate and filter input
Elevation of privilege	● Follow the principle of least privilege ● Use least privileged service accounts to run processes and access resources

3 Functional requirements

3.1 Roles and Actors

- **User:** the entity that performs the requests for using the remote services and wants to be authenticated. In some scenarios the user and the administrator may be the same person.
- **Administrator:** the system administrator on the server/client side installs the software and performs all system operations.
- **Authentication:** in computer security authentication is the process of attempting to verify the identity of the sender of a communication such as a request to log in. The sender being authenticated may be a person.
- **MFA:** Multifactor Authentication. The general idea of a MFA architecture is to use several types of credentials to authenticate the user during the logon to remote server computers in order to enforce a strong authentication security. In our case MFA is two-factor authentication system: user identity and TPM-equipped platform identity.
- **Login:** login is the process of receiving access to a computer system by identification of the user in order to obtain credential to permit access. It is an integral part of computer security procedure. Logon to system with multifactor authentication uses two credentials: user password and TPM-equipped platform identity to permit access to resources.

3.2 Use cases

3.2.1 Client/server Installation of MFA components

Use case unique ID	/UC 10/
Title	Client/server Installation of MFA components
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-11-15
Short description/purpose(s)	The client and server components are installed on two systems by the system administrator. After the installation is completed the user has access to the client computer components and the system administrator has access to server computer components.
Roles	Administrator
Includes	● An installed and running Open_TC framework on the client and server platform. ● Multifactor authentication services require fully a implemented Trusted Software Stack (TSS) for Linux according to TCG specification. TSS stack must support basic TCG functionality including the generation of Attestation Identities Keys (AIK), extending and retrieving the values of the TPM Platform Configuration Registers (PCRs). ● The security services such as OpenSSL. ● Crypto/PKI services. ● OpenTC Certificate Authority (CA) to provide the cryptographic certificate infrastructure.
Preconditions	(none)
Postcondition	The demo applications of the OTC security architecture can be used.

Normal Flow	The administrator executes and completes the installation procedure.

3.2.2 Client Initialization

Use case unique ID	/UC 100/
Title	Client initialization
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The administrator starts the utility to initialize the TPM and create an AIK.
Roles	Administrator
Preconditions	This TPM platform has not been initialized before; UC 10 should be done before this action.
Postcondition	Initialized platform can be used to log on into the remote server.
Normal Flow	1. The authorized administrator invokes the initialize utility to take the TPM ownership. 2. An AIK is created inside the local repository. 3. A request for an AIK certificate is built and sent to the OpenTC Privacy CA. 4. The AIK is activated and the AIK certificate is being retrieved. 5. OpenTC Privacy CA holds the copy of this certificate for the authentication purposes, and the platform as the unique identifier.

3.2.3 Platform registration

Use case unique ID	/UC 110/
Title	Platform registration
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The administrator registers the TPM-equipped platform with remote server.
Rationale	In a physically separated environment this is similar to registering a computing platform from a network.
Roles	Administrator (client/server)
Preconditions	UC 100 should be done before this action
Postcondition	User from the client trusted platform can remote log on to the server
Normal Flow	1. The administrator invokes the platform registration utility. 2. The system authenticates the administrator. 3. The administrator selects the server to register to. 4. The system registers the TPM platform credentials, created in UC 100, with selected remote server.

3.2.4 Platform unregistration

Use case unique ID	/UC 120/
Title	Platform unregistration
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The administrator unregisters the TPM-equipped platform with remote server.
Rationale	In a physically separated environment this is similar to unregistering a computing platform from a network.
Roles	Administrator (client/server)
Preconditions	UC 110 has been done before this action
Postcondition	User from the client trusted platform cannot remote log on anymore to the server
Normal Flow	5. The administrator invokes the platform unregistration utility. 6. The system authenticates the administrator. 7. The administrator selects the server to register to. 8. The system unregisters the TPM platform credentials, registered in UC 110, with selected remote server.

3.2.5 User Registration

The possibility of a new user registration relies solely on the existing infrastructure on the client and remote server. The MFA system does not intend to interfere in any way the security practices existing in the infrastructure.

On the server the user rights and access policy are controlled by the operational system.

3.2.6 Client logon

Use case unique ID	/UC 200/
Title	Client logon
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The user wishes to use a remote service.
Roles	User
Includes	/UC 220/ User Authentication
Preconditions	1. The client user is going to log on from the client computer already registered. 2. /UC 400/ should be done before this action
Normal Flow	1. The user notifies the local system (client) that s/he wishes to use a service running in a remote computer. 2. The user enters the user name and name or address of the remote server running the wanted service. 3. According to the security policy the remote system authenticates the user (see /UC 220/). 4. A session with the remote service is started.

3.2.7 Client logoff

Use case unique ID	/UC 210/
Title	Client logoff
Authors	Irina Beliakova, Vladimir Bashmakov
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The user wishes to end current working session with the remote server.
Roles	User
Includes	(none)
Preconditions	The user has already started a session with the server (see /UC 200/).
Postcondition	The user cannot use the server anymore without performing a new logon session.
Normal Flow	1. The user notifies the local system (client) that s/he wishes to end a session. 2. The user will be disconnected from the server.

3.2.8 User/platform authentication

Use case unique ID (per WP or SWP)	/UC 220/
Title	User/platform authentication
Authors	Irina Beliakova. Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The client asks the remote server to authenticate the user/TPM-equipped platform according to the policy set for the user
Roles	User
Includes	(none)
Preconditions	/UC 400/ should be done before this action.
Normal Flow	1. The system asks the user for authentication data. 2. The user enters data. 3. The remote system authenticates the user and TPM-equipped platform.

3.2.9 Using a remote server secure application/service

Use case unique ID	/UC 300/
Title	Using a remote server secure application/service
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The user starts a session with the remote secure application/service
Roles	User
Includes	/UC 200/ Client logon /UC 210/ Client logoff
Preconditions	1. /UC 110/ should be done before this action. 2. /UC 400/ should be done before this action.
Normal Flow	1. The user notifies the system that s/he wishes to start a logon session. 2. The system opens a new logon to application session (see /UC 200/). 3. The user works with the application 4. The user closes the application session (see /UC 210/). 5. The user-interface of the application disappears.

3.2.10 Server policy configuration

Use case unique ID	/UC 400/
Title	Server policy configuration
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The administrator starts the utility to configure the user logon and policy setting.
Roles	Administrator
Preconditions	/UC 10/ and /UC 100/ should be done before this action.
Postcondition	The user from the trusted client can log on to the remote server using the TPM-equipped platform identity .
Normal Flow	1. The authorized administrator invokes the Credential and Policy manager to set access right for logon for list of users. 2. The administrator set the policy allow to log on by verifying the user password or / and TPM-equipped platform identity.

3.2.11 Platform registration on the server

Use case unique ID	/UC 410/
Title	Platform registration on the server
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	The server registers the TPM-equipped platform. The server accepts the TPM credentials from clients and saves them in the Master Repository.
Rationale	In a physically separated environment, this is similar to registering a computing platform from a network.
Roles	Administrator
Preconditions	/UC 100/ should be done before
Postcondition	Server can be used for remote MFA access.
Normal Flow	1. This use case is triggered /UC 110/ 2. The server invokes the register service. 3. The system authenticates the administrator. 4. The register service saves TPM credential in Master Repository.

3.2.12 Server logon session

Use case unique ID	/UC 510/
Title	Server logon session
Authors	Irina Beliakova, Vladimir Bashmakov (INTEK)
Use case revision number	01
Use case revision date	2006-01-16
Short description/purpose(s)	After TPM client logon is successful the user have access to remote server resources according user permissions
Roles	User
Includes	
Preconditions	The client user is going to logon from the computer already registered. 1. /UC 110/ should be done before . 2. /UC 400/ should be done before
Normal Flow	1. Server user logon session starts 2. Authentication server requests the user, TPM credentials. 3. According to the security policy the remote system authenticates the user and TPM platform (see /UC 320/).

4 High level design

The design and implementation of the MFA system has a goal to prevent the listed threat by using as countermeasure technique the TPM-equipped platform identity for trusted remote access to server.

The multifactor authentication, including both user and 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.

Multifactor authentication to remote server should include components executed at server and client computers. Client components register the trusted platform with the remote server. Client components use the TSS at interface to the local TPM. A user can login to a remote server once the platform registration is completed.

The parameters of multifactor authentication system should be controlled by authentication policies configured on the remote platform.

4.1 Client operations

1. *Platform registration with the remote server.*
   An existing user with administrative rights can register the platform authentication information, namely the AIK and PCR's.
   The AIK is created inside the local TPM, the AIK certificate request is built and sent to the Open_TC Privacy CA. PCRs are extended during the bootstrap with the integrity measurements client computer software. The credentials are saved on the remote server.
2. *Standard user registration with remote server using the preexisting infrastructure.*
3. *Logon to the remote server.*
   The Logon Application (LAP) establishes secure communication channel with a remote server using OpenSSL or other secure channel. The LAP then retrieves the TPM Platform Configuration Registers (PCRs) values, platform AIK, and collects the User Identity (UI). PCRs and Challenge are signed with the platform AIK to ensure the information security. This authentication information is being sent to the remote server for authentication.

   The server queries the Open_TC Privacy Certificate Authority (CA) to certify the platform AIK. Then it verifies TPM PCRs. If both steps succeed, the server proceeds with user authentication, which relies on the existing authentication infrastructure and is out of the scope of this scenario.

4.2 Server functionality

1. *Register the platform and store platform information.*
   The server supports the platform registration. This procedure involves registering platform AIK certificate and PCR's.

2. *Register the user and store user information.*
   The server can verify user credentials. This basically relies on the existing authentication infrastructure.

3. *Edit user/platform policy for multifactor access.*
   The system administrator can change the user policies on the server to determine the required credentials in order to log on to the server.

4. *Enforce policy for server access.*
   The server can verify user credentials with defined policy set and reject logon if the multifactor authentication fails.

4.3 MFA system architecture

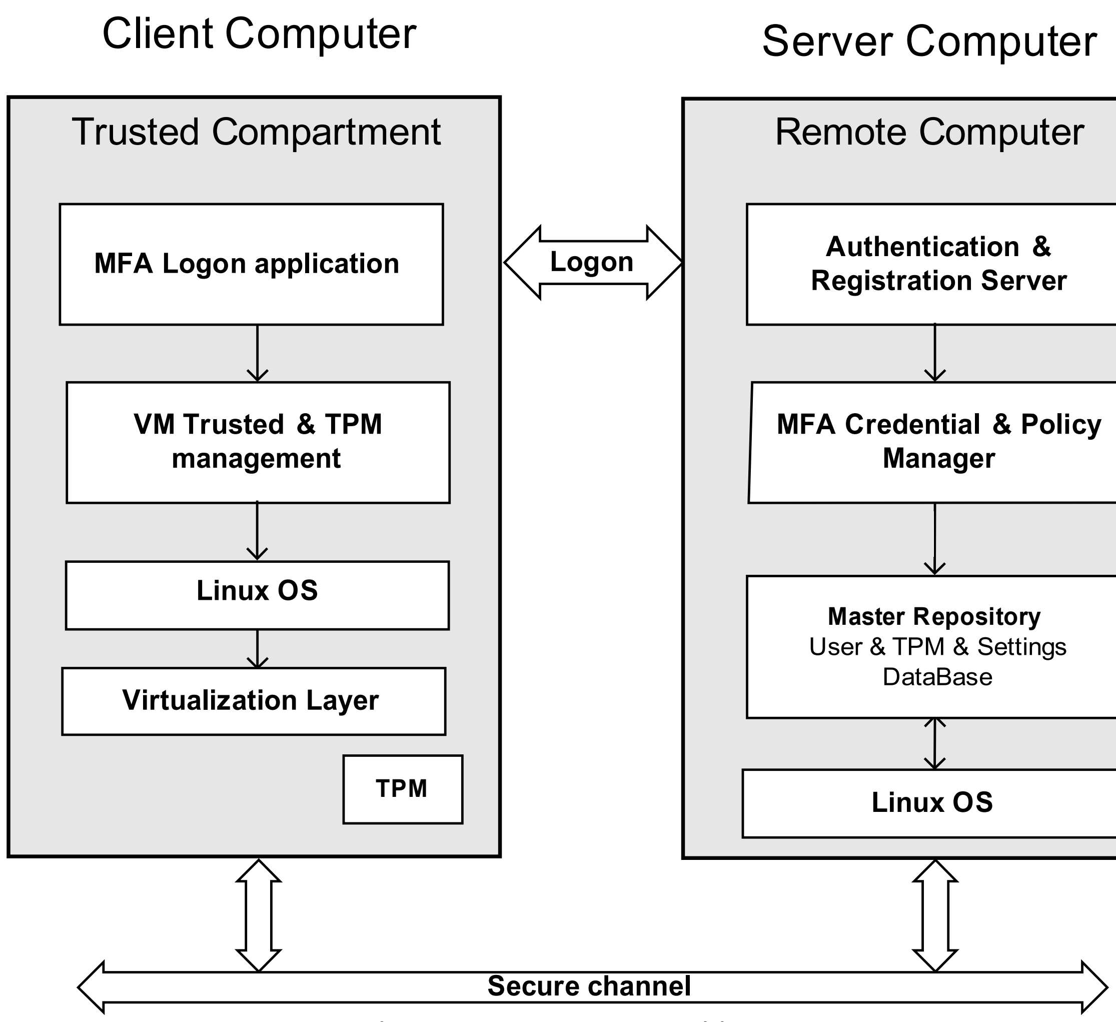

Figure 1: MFA system architecture

A preliminary MFA system architecture is depicted in figure 1 and the components are:

- **OpenTC framework.** *Open Trusted Computing framework* includes a **virtualization layer** that is managing multiple compartments and security services on the client.
- **Client.** *Client* is a computer system that accesses a (remote) service on another computer by some kind of network. For everyday tasks such as word processing, surfing the Internet or image processing the user is working with his well known, general purpose operating system. This operating system could run on top of a hypervisor in parallel to other compartments and security services.
- **Server.** *Server* is a computer system that provides services to other computing systems (clients). The entity the user wants to use to do some computer works.

The server defines the policy whether a connecting client user machine is to be considered “trusted” or not.

- **Credential & Policy manager.** *The credential & policy manager* is responsible to manage and store the user's credentials & policies. The credentials used when authenticating the user with the remote server. The credential manager ensures that these credentials are securely stored. Secure **Master Repository** is used to save and keep the secure data. MFA includes administrative functionality to configure authentication methods and access rights and settings.
- **TPM.** *Trusted Platform Module* is a chip that provides Trusted Computing features.
- **PCR.** *Platform Configuration Registers*. TPM volatile memory section contains platform configuration registers. These registers contain the integrity measurements (hashes) of the firmware and loaded software. During the system boot the BIOS, the BIOS extensions, MBR, the boot loader (namely GRUB) stages are measured and any designated files, such as the kernel. These measurements are used to extend the various PCRs.
- **Client components.** MFA components installed on the client.
- **Remote server components.** MFA components installed on the remote server.

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

Since the specification of the services and interfaces provided by the OpenTC framework are currently an ongoing work, the details of the interactions among MFA and such services will be specified in the final MFA specifications.

5 Required services from sublayers

- An installed and running Open_TC framework on the client platform.
- Multifactor authentication services require fully implemented 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) management.
- The security services such as OpenSSL for Open_TC will be used.
- Crypto/PKI services - from SWP03c.
- Open_TC Certificate Authority (CA) to provide the cryptographic certificate infrastructure.

6 Environment requirements

- A linux distribution (Fedora 5, OpenSUSE 10 or Damn Small Linux)
- PAM Framework

- OpenSSL Library
- TSS 1.2 library
- TPM Tools 1.2.4
- TSS Test Suite_27022006
- Linux 2.6.x
- gcc 3.4.x
- eclipse-3.1

7 Platform requirements

The Product is compatible with standard Linux environment with TPM platform installed and initialized.

The MFA system Product supports features and functionality enabled by a Trusted Platform Module (TPM) that requires the following components to be installed:

- Infineon TPM SLD 9630 TT 1.1b with TPM firmware 1.05 or higher; Infineon provides a firmware update via a tool based on the TCG field upgrade approach that will be deployed by means comparable to driver updates.
- TPM TestSuite –27022006 IBM
- TPM driver and TSS.

8 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
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

9 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/

D6e.3: Intermediate MFA System Specification

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

Deliverable reference number	IST-027635/D6e.3/PUBLIC | 1.00
Deliverable title	MFA Concept Prototype
WP contributing to the deliverable	WP6
Due date	Aug 2006 - M08
Actual submission date	Sept 12,2006

Responsible Organisation	INTEK
Authors	Irina Beliakova, Vladimir Tsisser
Abstract	This document describes MFA System which use trusted platform features for a secure, multifactor authentication to remote computers. System allows to make remote logon by using TPM credentials and/or password. The document contains system architecture, components, their interactions.
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

List of figures

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 TPM with the remote server. Client components uses the local TPM through TSS. The user can login to a remote server once the platform registration is completed.

The parameters of multifactor authentication system can be controlled by authentication policies.

This document gives high level design specification of a MFA system: the components protocol messages exchanged by the components and a preliminary API.

Since the specification of the services and interfaces provided by the OpenTC framework are currently an ongoing work, the details of the interactions among MFA and such services will be specified in the final MFA specifications.

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 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.

An installed and running Open_TC framework 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.
- Crypto and PKI services developed in SWP03c and SWP05d.

For a more comprehensive description of the MFA requirements, the reader should refer to the deliverable: “D06e.1 MFA Requirements and Specification”.

3 High level design specification

The MFA architecture includes two components components types:

1. Client components

   Register credential utility

2. Remote server components:

   Authentication and Registration Server

   Platform/User identities database: Master Repository

   Credential & Policy Manager

The client and server components are installed on two systems by the system administrator. After 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. (Fig.1)

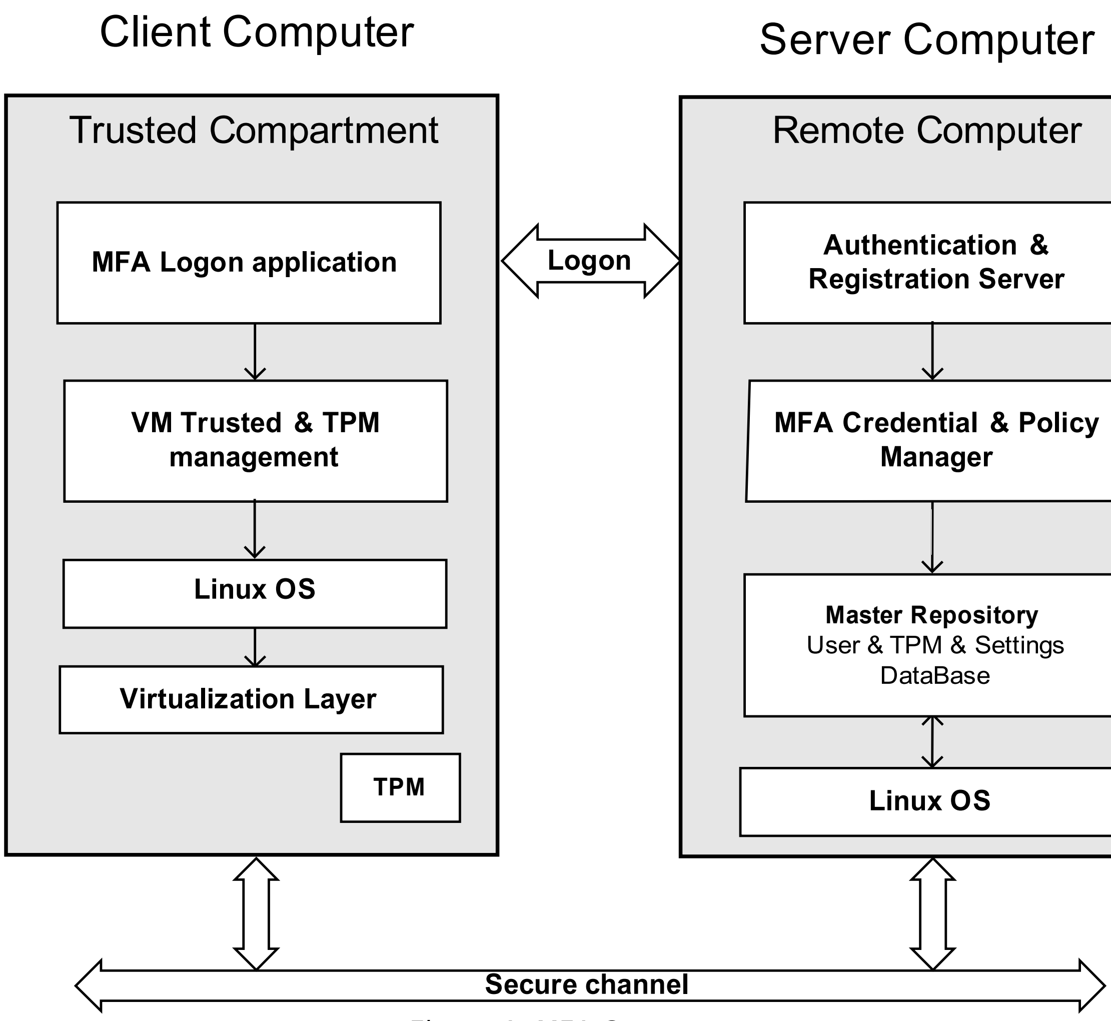

Figure 1: MFA Components

4 Functionality

4.1 Actions

There are three actions that can be performed on the client:

1. **Platform registration with the remote server.** An existing user with administrative rights can register the platform authentication information: Platform AIK and platform Configuration Registers (PCRs)**.**
Platform AIK is created inside the local TPM, AIK certificate request is built and sent to the Open_TC Privacy CA. PCRs are extended during the bootstrap with the integrity measurements of client computer software. Retrieved TPM Platform Configuration Registers (PCRs) values and Platform AIK as Combined Platform Registration Information (COREG) are sent to the server. The server receives the COREG and uses it to create the Platform Identity – data, necessary for authentication.

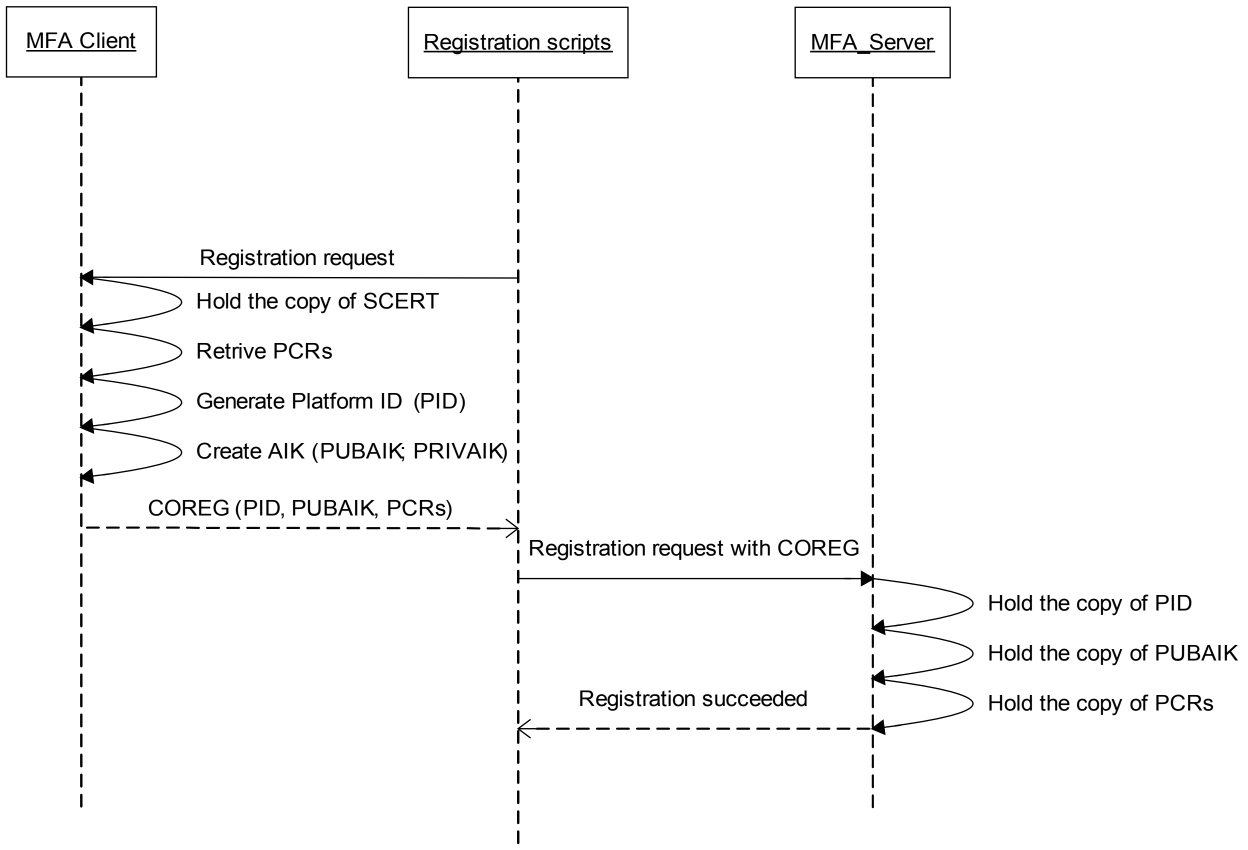

Figure 2: Platform registration

2. **User registration with remote server.** The possibility of a new user registration relies solely on the existing infrastructure. the application does not intend to interfere in any way the security practices existing in the infrastructure.
3. **Logon to the remote server.** A client can logon to the remote server. This

basically relies solely on the existing authentication infrastructure. In case the remote server logon system supports the MFA extension, a client platform authentication will be launched.

There are five actions that can be performed on the remote server:

1. **Register platform and store platform information**. The server supports the platform registration. This procedure involves registering Platform AIK certificate: the server saves the COREG in Master Repository.

2. **Platform authentication**. The server supports the platform authentication: the server logon system launches the platform authentication by calling a certain API function.

   MFA server application retrieves the user policy and check it. After a successful policy checking, the server establishes secure communication channel with a remote MFA client using Secure protocol (OpenSSL library).

   MFA server generates anti-reply challenge (CHALLENGE) and requests from MFA Client the PCRs signed using the AIK key.

   MFA client receives CHALLENGE and retrieves the TPM Platform Configuration Registers (PCRs) values and the Platform AIK, sets up the Combined Authentication Information (COAUTH):- Platform ID (PID), CHALLENGE - Platform TPM PCRs.

   COAUTH is signed with the Platform AIK to ensure the information integrity.

This Combined Authentication Information (COAUTH) is being send to the remote server for authentication.

   MFA server receives COAUTH and proceeds with the platform authentication. It retrieves stored the platform information by PID, it verifies the signature and involves the verification of the Platform TPM PCRs.

In this scenario network calls are synchronous. An established SSL channel is used as binary stream.

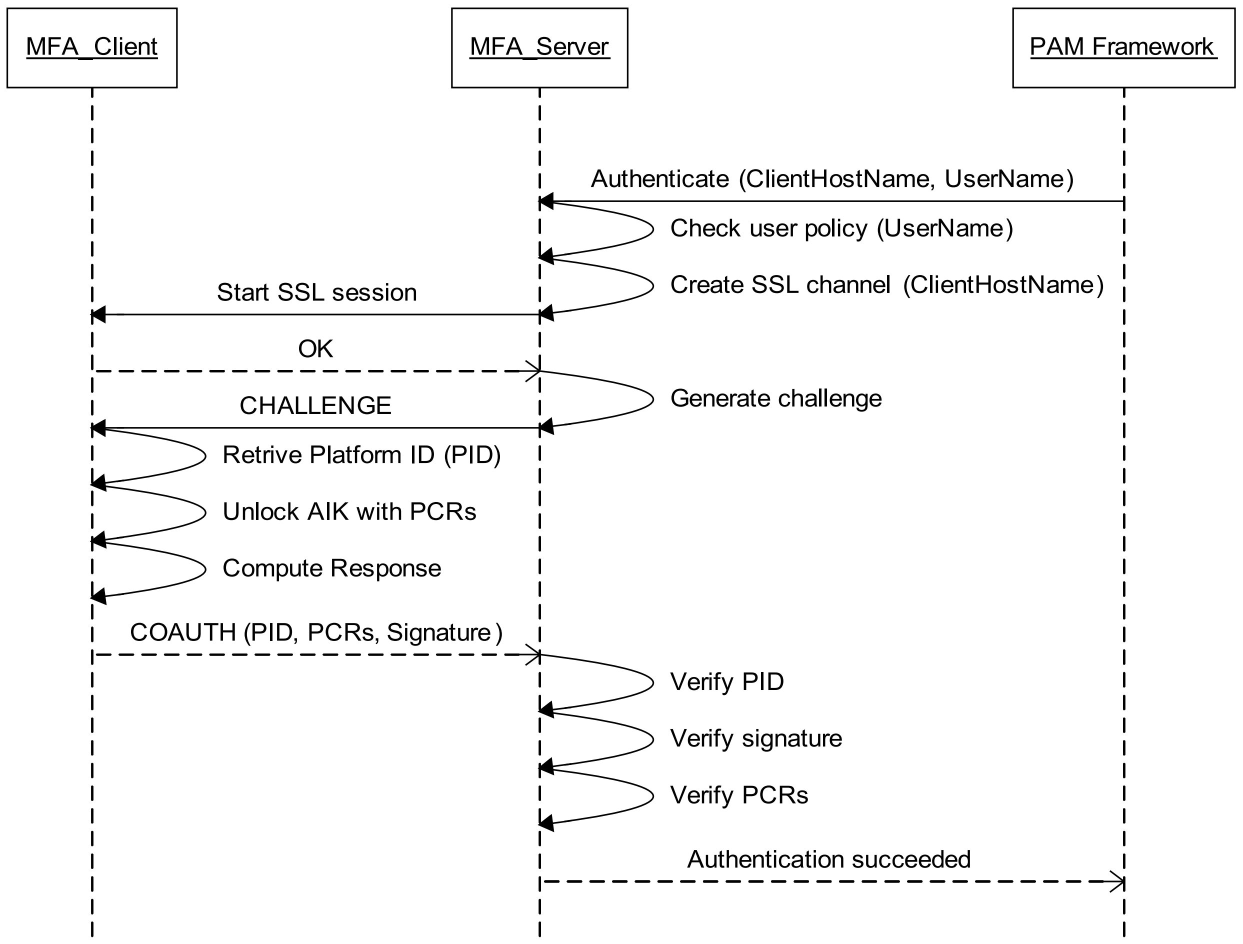

Figure 3: Platform authentication

3. **Authenticate user.** The server can verify user credentials. This basically relies on the existing authentication infrastructure.
4. **Edit user/platform policy for multifactor access.** The system administrator can change the user policies on the server to determine the required credentials in order to logon to the server.
5. **Enforce policy for server access.** The server can verify the user credentials against a defined policy set and reject the logon if multifactor authentication fails.

4.2 Policies

The policy determines what type authentication is required to access the remote service. The following policies will be defined:

- The platform authentication is required (optional if policy is not set).
- The list of platforms the user is allowed to login from (the user can use any registered platform if this policy is not set).

4.3 Data structure organization example

All data can be expressed as two tables. The first one is the “Registered Platforms” table. The second is a “Users Policies” table. “Registered Platforms” table consist of three columns: “Platform ID (PID)”; “PCRs”; “Public AIK”. “Platform ID (PID)” is the Primary Key (PK) of this table. “Users Policies” table also consists of three columns: “User Name”; “Policy”; “Platform IDs set” where “User Name” column is the Primary Key (PK) of this table.

“Registered Platforms” table example:

Platform ID (PID)	*PCRs*	*Certificate (Public AIK)*
123987	binary string	binary string
321789	binary string	binary string
456654	binary string	binary string

“Users Policies” table example:

User Name	*Policy*	*Platform IDs set*
Jack	exclude	NULL
Tom	include	321789; 456654
Bill	include	NULL

In this example Jack can use any client computer to login to the server. The TPM authentication will not be performed. Tom can use only two computers with a registered TPM to login to the server. Bill can use any computer with a registered TPM.

5 Preliminary MFA API

5.1 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
MFA_HCONTEXT	UINT32	Context object handle
MFA_FLAG	UINT32	Object attributes
MFA_RESULT	UINT32	result of a MFA interface command

Common return code defines:

Type	Definition
MFA_SUCCESS	Success
MFA_E_FAIL	Non-specific failure
MFA_E_BAD_ARGUMENT	One or more parameter is bad.
MFA_E_INTERNAL_ERROR	An internal SW error has been detected.
MFA_E_NOTIMPL	Not implemented
MFA_E_CANCELED	The action was canceled
MFA_E_TIMEOUT	The operation has timed out
MFA_E_OUTOFMEMORY	Ran out of memory

5.2 Structures

This section describes the structures defined by this API.

5.2.1 MFA_VERSION

This structure allows the MFA to communicate between various versions of client and server application components.

Definition:

```
typedef struct tdMFA_VERSION
{
  BYTE bMajor;
  BYTE bMinor;
} MFA_VERSION;
```

Parameters:

bMajor
This SHALL be the major version indicator for this implementation of the MFA specification. For version 1 this must be 0x01

bMinor
This SHALL be the minor version indicator for this implementation of the MFA specification. For version 1.0 this must be 0x00, for version 1.1, this must be 0x01.

5.2.1 MFA_CONTEXT_INFO

This structure provides information about context status. It used with mfa_S_Context_GetStatus and mfa_C_Context_GetStatus interface functions.

Definition:

```
typedef struct tdMFA_CONTEXT_INFO
{
  MFA_VERSION versionInfo;
  MFA_RESULT  lastError;
} MFA_CONTEXT_INFO;
```

Parameters:

versionInfo
Version data set by MFA interface function

lastError
Last occurred error in current context

5.3 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.
The MFA allocates memory for out parameters and provides a function to free the memory previously allocated by the MFA on a context object base. The calling application MUST free memory allocated by the MFA. The caller of the MFA interface functions is responsible for calling mfa_S_Context_FreeMemory or mfa_C_Context_FreeMemory for each call that produced allocation of memory.

The mfa_Context class represents a context of a connection to the server side MFA application core.

5.3.1 mfa_Context_Create

This method returns a handle to a new server side context object. The context handle is used in various functions to assign resources to it.

Definition:

```
MFA_RESULT mfa_Context_Create
(
    MFA_HCONTEXT* phContext    // out
);
```

Parameters:

phContext
Pointer to a variable that receives the handle to the created context object.

Return Values:

MFA_SUCCESS
MFA_E_BAD_ARGUMENT

5.3.2 mfa_Context_Close

This method destroys a server side context and releases all assigned resources.

Definition:

```
MFA_RESULT mfa_Context_Close
(
    MFA_HCONTEXT hContext    // in
);
```

Parameters:

hContext
Handle of the context object which is to be closed.

Return Values:

MFA_SUCCESS
MFA_E_INVALID_HANDLE
MFA_E_INTERNAL_ERROR

5.3.3 mfa_Context_GetStatus

This method provides operation status in specified context.

Definition:

```
MFA_RESULT mfa_Context_GetStatus
(
    MFA_HCONTEXT        hContext, // in
    MFA_CONTEXT_INFO** ppStatus  // out
);
```

Parameters:

hContext
Handle of the context object.

ppStatus
Pointer to a variable that receives a pointer to the context information structure. When the use of the pointer ends, free the returned buffer by calling the mfa_S_Context_FreeMemory function.

Return Values:

MFA_SUCCESS
MFA_E_BAD_ARGUMENT
MFA_E_INVALID_HANDLE
MFA_E_INTERNAL_ERROR

5.3.4 mfa_Context_FreeMemory

This method frees memory allocated by MFA application on a server side context base.

Definition:

```
MFA_RESULT mfa_Context_FreeMemory
(
    MFA_HCONTEXT hContext, // in
    BYTE*        rgbMemory // in
);
```

Parameters:

hContext
Handle of the context object.

rgbMemory
Pointer to the memory block to be deallocated.

Return Values:

MFA_SUCCESS
MFA_E_INVALID_HANDLE

MFA_E_INTERNAL_ERROR
MFA_E_INVALID_RESOURCE

5.3.5 *mfa_S_RegisterPlatform*

This method store platform registration information received from remote client as Platform Identity for future use in authentication process.

Definition:

```
MFA_RESULT mfa_S_RegisterPlatform
(
    MFA_HCONTEXT hContext,      // in
    BYTE*        pPltmInfoData, // in
    UINT32       pltmInfoSize   // out
);
```

Parameters:

hContext
Handle of the context object.
pPltmInfoData
Pointer to a buffer containing the platform registration information (COREG).
pltmInfoSize
Size of the information pointed to by the *pPltmInfoData* parameter, in bytes.

5.3.6 *mfa_S_Authenticate*

This method provides remote platform authentication.

Definition:

```
MFA_RESULT mfa_S_Authenticate
(
    MFA_HCONTEXT hContext,         // in
    MFA_UNICODE* userName,         // in
    MFA_UNICODE* platformHostName  // in
);
```

Parameters:

hContext
Handle of the context object which is to be closed.
userName
Pointer to the MFA_UNICODE string contain name of the user
platformHostName
Pointer to the MFA_UNICODE string contain name of the client host platform

5.3.7 *mfa_S_SetAccessPolicy*

This method sets user access policy. That data is used during authentication process.

Definition:

```
MFA_RESULT mfa_S_SetAccessPolicy
(
    MFA_HCONTEXT hContext,      // in
    BYTE*        pPolInfoData, // in
    UINT32       pPolInfoSize  // in
);
```

Parameters:

hContext
Handle of the context object.
pPolInfoData
Pointer to a buffer containing the user policy information.
pltmInfoSize
Size of the information pointed to by the *pPolInfoData* parameter, in bytes.

5.3.8 mfa_S_GetAccessPolicy

This method gets user access policy. That data is used during authentication process.

Definition:

```
MFA_RESULT mfa_S_GetAccessPolicy
(
    MFA_HCONTEXT hContext,      // in
    BYTE*        pPolInfoData, // out
    UINT32*     pPolInfoSize  // in/out
);
```

Parameters:

hContext
Handle of the context object.
pPolInfoData
Pointer to a buffer with user policy information.
pltmInfoSize
Pointer to size of the information pointed to by the *pPolInfoData* parameter, in bytes.

5.3.9 mfa_Platform_Initialize

This methods takes TPM ownership, create Platform AIK.

Definition:

```
MFA_RESULT mfa_Initialize
(
);
```

Parameters:

Return Values:

MFA_SUCCESS

5.3.10 mfa_C_GetPlaformData

This method retrieves the platform information.

Definition:

```
MFA_RESULT mfa_C_GetPlatformData
(
    MFA_HCONTEXT hContext,       // in
    BYTE**       ppPltmInfoData, // out
    UINT32*      pPltmInfoSize   // out
);
```

Parameters:

hContext
Handle of the context object.

ppPltmInfoData
Pointer to a variable that receives a pointer to the buffer which contain the platform information. When the use of the pointer ends, free the returned buffer by calling the mfa_S_Context_FreeMemory function.

pPltmInfoSize
Pointer to a variable that receives the size of the buffer pointed to the *ppPltmInfoData* parameter, in bytes. When the function returns, the UINT32 value contains the number of bytes stored in the buffer.

6 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

7 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/