GWD-TYPE: SEC
Category: RG
TC-RG
Version 0.1, Feb 12, 2006

The Daonity Team
HP Labs, China, Beijing, China
Wuhan University, Wuhan, China
Huazhong Univ. of Sci & Tech, Wuhan, China

Daonity Specifications
Part I - System Design

Status of This Document

This working draft document provides information to the Grid community the work of Daonity Project which aims to strengthen Grid Security Infrastructure (GSI) using the Trusted Computing (TC) technology of Trusted Computing Group (TCG). At this stage this document is for public comment purpose and does not define any standards or technical recommendations. As a working draft, revisions may be made at any time in the public commenting period which we designate to be the time space between GGF16 to GGF17, until a finalization announcement which we expect to be at GGF17. Distribution is unlimited.

Copyright Notice

Abstract

This document describes part of the work conducted by the Daonity team. Daonity is an R&D project in Trusted Computing Research Group (TC-RG), SEC Standard Area, Global Grid Forum, with a mission to strengthen Grid Security Infrastructure (GSI) of Globus Toolkit (GT) by applying the Trusted Computing (TC) technology of Trusted Computing Group (TCG). The part of the work described in this document is a specification of the system design of the Daonity system.

Contents

The Daonity Team
HP Labs, China, Beijing, China
Wuhan University, Wuhan, China
Huazhong Univ. of Sci & Tech, Wuhan, China

1. Introduction

Grid security is a key component in Grid computing. The mainstream Grid security solution, Grid Security Infrastructure (GSI) for Globus Toolkit (GT), offers comprehensive security services. This is achieved by applying public-key cryptography, cryptographic protocols methodologies and the necessary infrastructural supporting services in which public-key authentication framework (PKI) is the main component. Although an important development from the provisions of security services for distributed computing, it is considered that there is much room for further strengthening GSI in the direction that Grid security can make a clear distinction from distributed computing security. A desired distinction would be that security services for Grid security should manifest and facilitate the Grid feature of advanced resource sharing. It is reasonable to expect that to make such a distinction it requires to go beyond the conventions of distributed computing security solutions.

A useful security service from the trusted computing technology (TC) of Trusted Computing Group (TCG) is **behavior conformation** which can confine the users including the owner of a computing platform to behavior desired by a remote user who may be participating in a "gameplay" on the platform. This security service is practically achievable in the TC technology because in TC every platform has a tamper protection hardware module called Trusted Platform Module (TPM) to set the desired behavior for the software systems running on, and users using, the platform. In May 2005, the SEC Standard Area of Global Grid Forum set forth an investigation effort on how this feature of TC can make a positive impact on Grid security. This is the Trusted Computing Research Group (TC-RG). As following TCG's open specifications many computing platform manufacturers have been developing and marketing TCG products, part of the TC-RG's work is designated to starting exploring available TCG technologies and developing their trials for Grid security. This is the mission of Project Daonity.

An non-trivial result from the Daonity team's investigation is that, a TPM, even sparsely deployed in an early stage of the TCG technology deployment, can be a piece of shared security resource to enable strong and advanced Grid security solutions. Therefore, with TPM as a piece of shared resource, it is natural for the TCG technology to support the Grid feature of resource sharing even in security's own right. With this understanding and some insights gained in our investigation, the Daonity team has been, following the stipulation of the TC-RG Charter and Milestones, conducting a system development work to design and implement a TCG enabled GSI system. This working draft document specifies the full system design of the Daonity system.

1.1 Trusted Computing

In recent years, increased reliance on computer security and the unfortunate fact of lack of it, particularly in the open-architecture computing platforms, have motivated many efforts made by the computing industry. Among these is the development of Trusted Computing (TC). In 1999 five companies – (then Compaq), HP, IBM, Intel and Microsoft -- founded the Trusted Computing Platform Alliance (TCPA). The motivation of TCPA was to add trust to open-architecture computing platforms. In 2003 the TCPA achieved a membership of 190 plus companies, when it was incorporated to the Trusted Computing Group (TCG). TCG is a vendor-neutral and not-for-profit organization for designing, specifying and promoting industrial standards for the TC technology. The TCG work has so far been developed to contain sufficient innovations and become a standard methodology for adding trust and security to open-architecture computing platforms.

The TCG's approach to adding trust is to integrate to a computer platform a hardware module called Trusted Platform Module (TPM) with tamper-protection property and let it play the role of an in-platform trusted third party agent. It is intended that such an agent can function to enforce a conformed behavior for software systems running on the platform and the users using the platform. Under the tamper-protection assumption which is practically achievable using various cost-effective hardware design and implementation techniques, a software system or a user, including one which is under the full control of the platform owner (such as a root user in Unix or Administrator in Windows), cannot bypass security policy set by the TPM. This is the so-called *behavior conformation* property of the TC technology. For a very simple example of behavior conformation, the TPM can prevent the owner of a platform from accessing users' confidential data. It worths pointing out that this specific and simple case of behavior conformation will constitute the major technical contribution from Daonity to improving the current Grid security practice.

The following TCG documents which are relevant to the work of this document are available at the "Downloads area" of TCG's website <www.trustedcomputinggroup.org>:

1 TPM Main, Part 1, Design Principles, Speci_cation Version 1.2, Revision 85, 13 February 2005.
2 TPM Main, Part 2, TPM Structures, Speci_cation Version 1.2, Level 2 Revision 85, 13 February 2005.
3 TPM Main, Part 3, Commands, Speci_cation Version 1.2, Level 2 Revision 85, 13 February 2005.
4 TCG Speci_cation, Architecture Overview, Speci_cation Revision 1.2, 28 April 2004.

1.2 Grid Security Infrastructure

Similar to security services for distributed computing systems, important security services which GSI requires are: entity authentication, user authorization, message confidentiality, data integrity, and undeniability to a commitment. GSI achieves these services through innovative applications of standard public-key cryptographic solutions which we overview below.

For entity authentication, message confidentiality and data integrity, GSI includes both resource- and client-side mechanisms. In the resource side, GSI security mechanisms include X.509 credentials for identifying the resource. In the client side, these include facilities to create temporary credentials, called a *proxy*, for performing single sign-on and delegation. The client performs mutual authentication with the target resource using the above certificates and establishes a secure, encrypted communication channel by applying the Transport Layer Security (TLS) protocols. In GT4, GSI also supports message-level security which implements security services from WS-Security and WS-Secure Conversation specifications. By using this, it is possible to achieve security protection at per-message level for SOAP messages. In GT4, transport-level security using TLS protocols is set as the default because of its performance advantages. This setting will be deprecated as message-level security improves in performance, but that is not envisioned to happen or likely to happen in any near future.

For authorization, the client can choose to delegate its credentials to the resource to enable subsequent resource access without further intervention. This is done by the resource side making use of the grid map file to associate a presented certificate with a local user account. It can then spawn processes using the local user account's authorization privileges. Optionally, the client can make use of tools such as Community Authorization Service (CAS) for fine-grained authorization. Since GT4 messages are Simple Object Access Protocol (SOAP)-based, it is also possible to turn on message-level security to provide protection for and to ensure integrity of these messages.

GSI implements security services by applying standard security techniques from various standard bodies and security specifications from the Web services community. In GT4 security is composed of both Web-services-based and non-Web-services-based elements. GSI has been traditionally based upon public-key cryptographic techniques for all its security functionalities. It uses X.509 end-entity certificates (EECs) for establishing identities of persistent entities such as users and resources. It also introduces the notion of X.509 proxy certificates to support the delegation and establishment of temporary, often short-lived, entities. GSI treats both of these types of certificates equivalently.

1.3 Grid Security Requirements

The primary requirements by Grid security are:

1) Need for secure communications for the Grid setting of virtual organizations (VOs). A VO typically is composed of users and resource providers across conventional organizational boundaries. Thus a centrally-managed security solution won't suit GSI.
2) Ease of use by users. An important element in this requirement is the need for provisioning "single sign-on" for users, including delegation of credentials for computations that involve multiple resources and/or sites.
3) Applications of standard technologies. This not only facilitates fast and ready deployment of the Grid technologies, but also helps to ensure correct applications of security techniques.

1.4 Problem with the Current Grid Security Practice

GSI actually meets requirements (2) and (3) in Section 1.3 very well. In specific, these are achieved via innovative applications of public-key certification infrastructure (PKI), proxy certificates and MyProxy online servers. We shall omit describing these techniques here as they should already be familiar to the expected reader of this document.

It is however our understanding that the current GSI practice does not make a noticeable impact on meeting the requirement (1) in Section 1.3. Let us consider the most general setting for a VO of users and resource providers. In order for the VO to be able to define flexible and may be ad hoc security policies which need to be applicable to these entities in a uniformed manner, it is desirable that each of these entities has strong security means which can enforce them in the execution of the policy. For example, a VO policy may stipulate that a resource (or a file) can become usable (accessible) by a user only after the user has conducted certain work to have satisfied a collaboration or service policy. However, in the current GSI practice, security means that a user has an exclusive entitlement to an action provided a cryptographic credential is available. Indeed, in a non-TC environment, it is usually the case that a user has the full access to, and unlimited usage of, an owned cryptographic credential, and this is a consequence of missing a behavior conformation service. Without behavior conformation, it is very difficult for the VO to apply fine granularity control on VO policies. To put the problem in another way, the current GSI practice has coarse policy enforcement on VO entities: an entity is either an insider who then can do everything, or an outsider otherwise.

Conventional applications of standard security technologies for distributed computing do not permit the resultant Grid security practice of the current GSI to make a clear distinction from the counterparts in distributed computing.

1.5 Daonity's Contribution

Project Daonity attempts to solve the Grid security problem we discussed in Section 1.4 by making use of the behavior conformation security service from the TCG technology. In the first phase of the project we shall only consider to develop client side middleware systems to use TPMs. This phase 1 confinement is because we consider that improving policy enforcement is mostly needed in the client environment.

Although major computing platform vendors have been marketing TCG products for some time and as a result TPMs have already reached a non-trivial extent of deployment, we recognize and anticipate that TPMs as new hardware equipment to computing platforms cannot become universally available within a short period of time. In fact, it is likely that many users will continue using client platforms without TPM. We overcome this problem by allowing use of remote TPMs as shared security resource. This can be achieved by a cryptographic protocol solution which involves an extension to the functionality of the online services of the MyProxy server.

A high level description of this protocol solution to sharing of TPM can be as follows. Let Alice be a VO user whose platform has no TPM.

1) Alice registers with MyProxy using a conventional credential, such as one based on a shared password or a SecureID like mechanism (of course, the registration should go through out-band communications between Alice and MyProxy);
2) To use a remote TPM, Alice obtains a public key of the target TPM (according to the most basic property of behavior conformation, this implies that the matching private key is inside the TPM and cannot be accessed or used, even by the TPM owner, for any purpose other than a behavior conformation designation);
3) Alice then sends a request, with the public key of the target TPM, to MyProxy for creating a public-key based cryptographic (proxy) credential for her to use the target TPM;
4) MyProxy can double encrypt the private key of Alice's proxy credential, using the secret shared with Alice and the public key of the target TPM.

Thus, provided the TPM owner permits Alice to use the target TPM as a shared resource (the owner decides according to the rule of the gameplay), Alice can then enjoy strong and policy-conforming security services which are offered from the TPM owner.

In this way, not only can Daonity provide a strong Grid security solution suiting VO environments, but also the solution be provided without damaging GSI's single sign-on property.

There are also other innovative offers from Daonity's applications of the TC technologies. We shall detail them in the remainder of this document.

2. Daonity System Design – An Abstract View

In this section we provide an abstract view of the Daonity system.

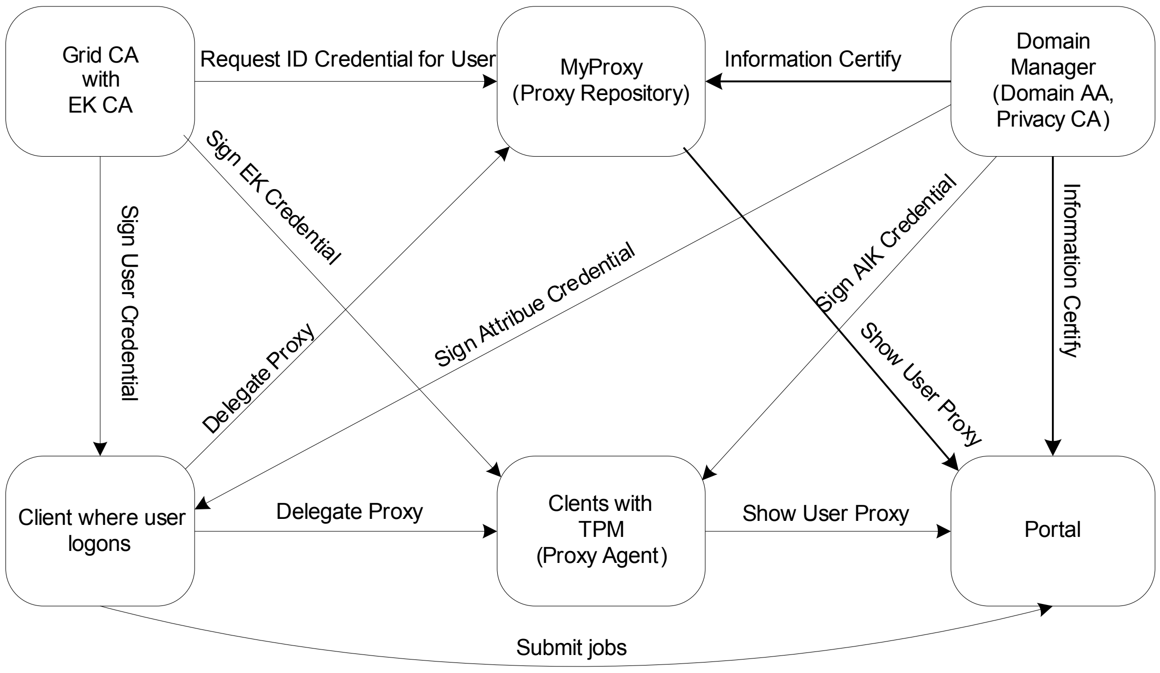

Figure 2-1. Daonity Network Topology

2.1 Daonity Network Topology

Figure 2-1 depicts the network topology of the Daonity system. Grid CA, MyProxy, Portal and User Client are components in the legacy GT. Grid CA certifies users and resources by issuing credentials to them. MyProxy helps users to manage short-lived proxy credentials. Portal is the gate of Grid services; many operations of a user can be performed on it. Daonity focuses on enhancing security using the TC technology. Although we anticipate that most of these servers and clients have TPMs, Daonity shall only require some of the client side platforms to be equipped so. As illustrated in Figure 2-1, there are three new components that are not in the original GSI. They are: (1) Endorsement-key (EK) CA and (2) Privacy CA which are the components in the TCG specification, and (3) Proxy Agent which is our add-on for Daonity. We will discuss these new modules and the workflow of Daonity in a moment.

2.2 Middleware System Architecture

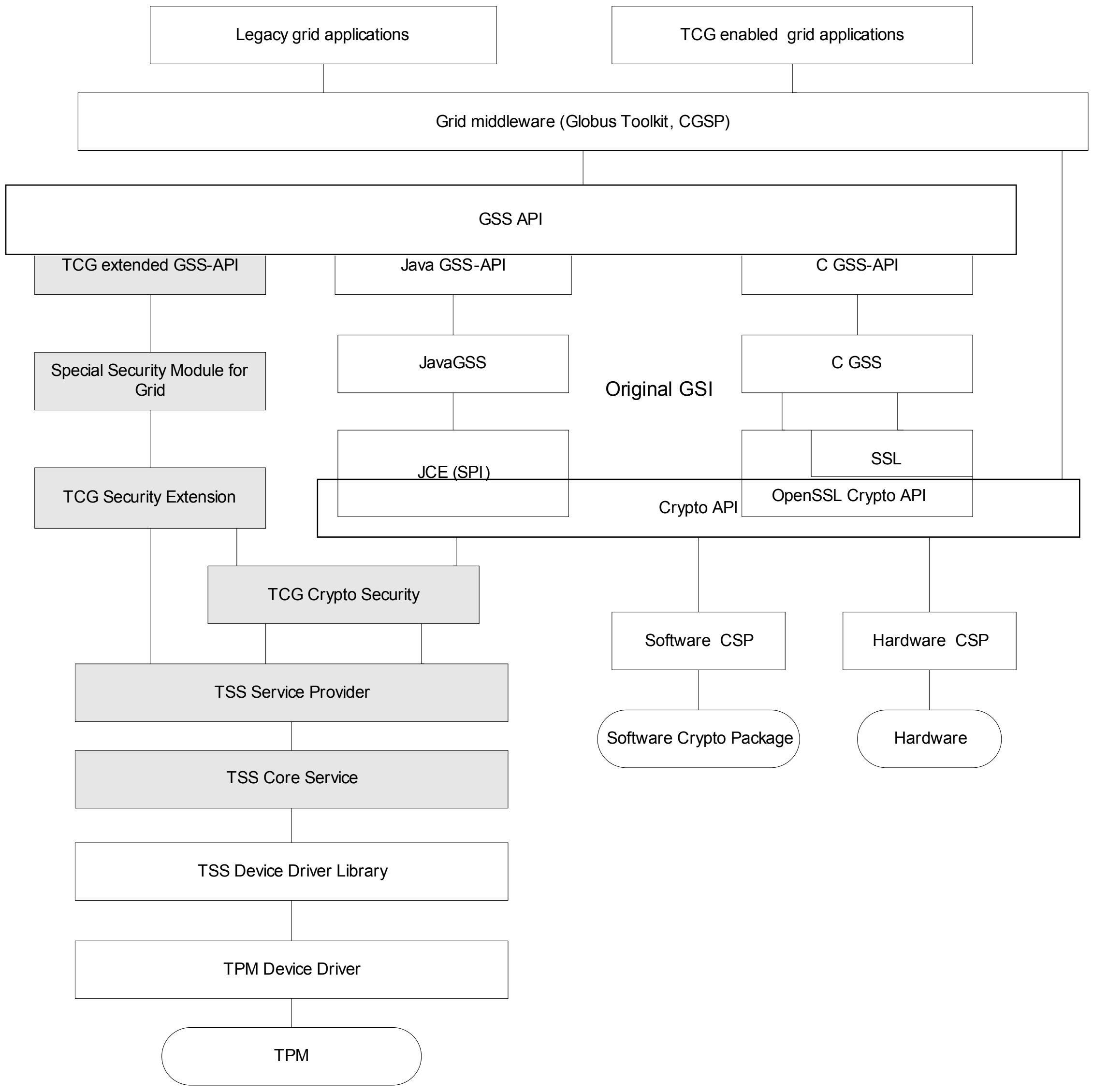

Figure 2-2. TC-enhanced Grid Security Infrastructure

Figure 2-2 depicts an architectural structure of a Grid security infrastructure which is enhanced by the TC technology. The work of Daonity focuses on the gray components in the diagram. All the work which is involved in Daonity is below the GSS API. Therefore, Grid applications in the legacy system can run in Daonity without modification. The work of Daonity can be vertically divided into two parts: GSI related and TSS (TCG Software Stack) related. TSS is a TCG component providing the system developer a standard means to interact with the TPM hardware and utilize the cryptographic function of the TPM. The work in the GSI related part is to enhance the original GSI functions. The original GSI gets cryptographic services by calling standard Crypto API (SPI in Java, OpenSSL Crypto API in C). Cryptographic service providers (CSP) are the code implementations of crypto algorithms which can be in either software (running in the general CPU) or hardware (running in, e.g., a smartcard or a USB token). With the same idea which TCG adopts, CSPs can be implemented in the TPM (with Crypto API calling the TCG Crypto Security Services). The “Special Security Module for Grid” includes certificate management, single sign on, authorization, and authentication.

2.3 Standard Approach to Applications of the TC Technology

In Daonity, the applications of the TC technology follow the TCG standard software approach. This is Trusted Software Stack (TSS for short).

Figure 2-2 illustrates the Daonity architecture, which is composed of the trusted hardware module (TPM), trusted software stack (TSS) components and Grid security infrastructure. TSS components provide fundamental resources to support the use of the TPM.

The primary design goals are:
- To supply a secure container for grid application
- To improve grid authorization and authentication
- To provide secure mobile grid access with key migration
- To manage TPM resources

Based on the TCG TSS specification, we classify our design into five Parts:
- Context Management (including TSP Context and TCS Context Management)
- Key Managements (including key tree hierarchy, key storage and key migration)
- File Protection Management
- TPM Management (including TPM Object Management, TPM Parameter Block Generator and TCG Device Driver Library)
- Session Authorization Management

- Context Management

It plays an important role in our architecture, including TSS context manager, TCS context manager and context communication. Through context, different modules and different layers can share knowledge and TPM can control all modules and layers in secure ways.

From the viewpoint of a user, TCS and TSP layer context provide a container to access TPM. It conceals different trusted hardware, different key management and different trusted attestation solution from user.

From the viewpoint of system, TSP layer context provide a channel to communicate with working objects, such as key management module, file protection module and etc. TCS layer context handles requirements from TSP layer context, authenticates TSP layer context and management special core service module such as KCM module, TPM PGB and etc.

- Key Management

There are several types keys in Daonity. All keys are protected by TCS PS (Persistent Storage) or TSP PS and accessed by UUID (Unified Unique Identity), except of some important key, which can’t be exposed outside TPM. UUID is a unique name of keys for application layer users.

In PS, parent key encrypts its children keys. When a user uses its key, he should have an authorization from its parent key. In TPM parent key and parent and parent key are decrypted and registered for decrypt its children key recursively. Once a user key is authorization, this user will get a key handle in TPM but still don’t know the key’s plain text.

Key Cache Management is a cache for restricted TPM. Its responsibilities are to cache the key handles that are used recently and schedule key handles when request comes.

When a key is migrated to another TPM platform, Key manager would wrap this key in special way and securely register this key in remote TPM.

- File Protection

File Protection is our special offers for Grid platform. In grid application, the file named GridMap is an urgent resource. In our protection solution, modifications in this file will be audited, authenticated and protected.

- TPM Management

TPM is our trusted hardware. According to three layer of TSS, we give a TPM Object of user layer provides high-level interface to TPM for application, a PBG of system layer for communication to the TPM and a TDDL for device driver layer.

- Session Authorization Management

TPM shared mode arise authorization and authentication among TSP, TCS and TPM layer. Session Authorization Management provides protection for session security. It communicates with TPM through OSAP or OIAP protocol defined in TCG specification.

3. GSI Innovation

This section describes necessary modifications to the GSI system due to applications of the TC technology.

3.1 Workflows

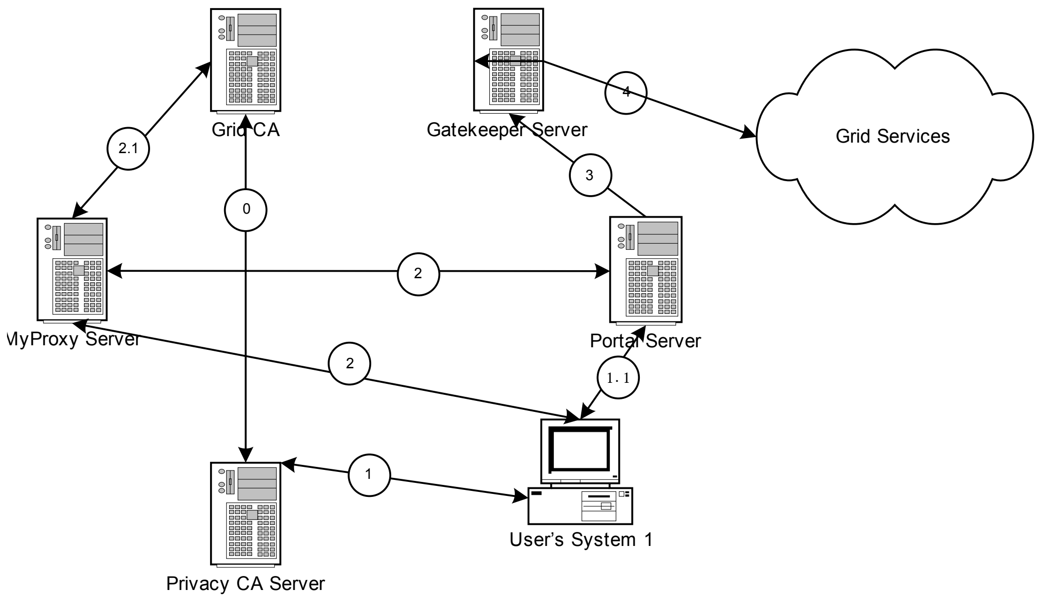

Figure 3-1. Workflow for Grid System

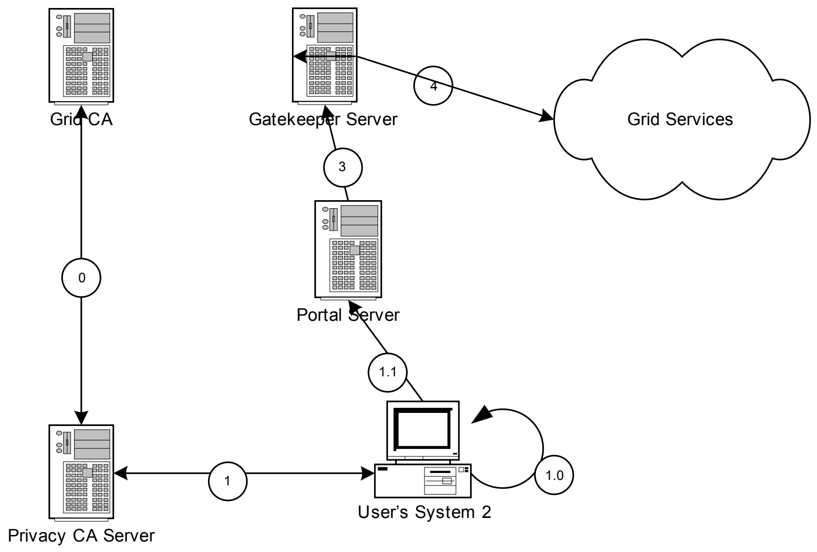

Figure 3-2. Workflow for Grid System

The whole Grid environment can be described as Figure 3-1. A user can submit his job to Grid Job Management Service directly; he can also submit his job through the Grid Portal indirectly. Before submitting his job, the user needs to acquire his identity certificate and proxy certificate. When submitting a job, the user must provide his proxy certificate so that the Job Management Server can deliver the job as the identity of the user. The user can apply for an identity certificate from Grid CA. The user can then create a proxy certificate to by him using the credential in his identity certificate. If a user does not have an identity certificate, he can also obtain a proxy certificate from MyProxy server. Proxy certificate can be delegated further if the certificate policies allow.

Because of the utilization of TCG technology, we use the Private CA Server, which can sign AIK (randomly generated public/private key pair with the private staying in the TPM) certificates, both for users' platforms and for services'. A Private CA is as a branch of the System CA. Two platforms that have AIK certificates and/or identity certificates can authenticate each other through using these credentials. An AIK certificate can be used as a platform identity credential. Its trust level is decided by the system's policy.

Proxy Certificate depository (MyProxy server) is an online certificate management server, which can manage users' certificates and proxy certificates. MyProxy can also provide the proxy certificate renewal service by negotiating with the Job Manager. As a security sensitive point in the grid system, MyProxy server needs higher security protection. Moreover, the system requires that users must use non-anonymous connection when they communicate with MyProxy server. Internal Components

3.2 Architecture

The system structure is as Figure 3-3.

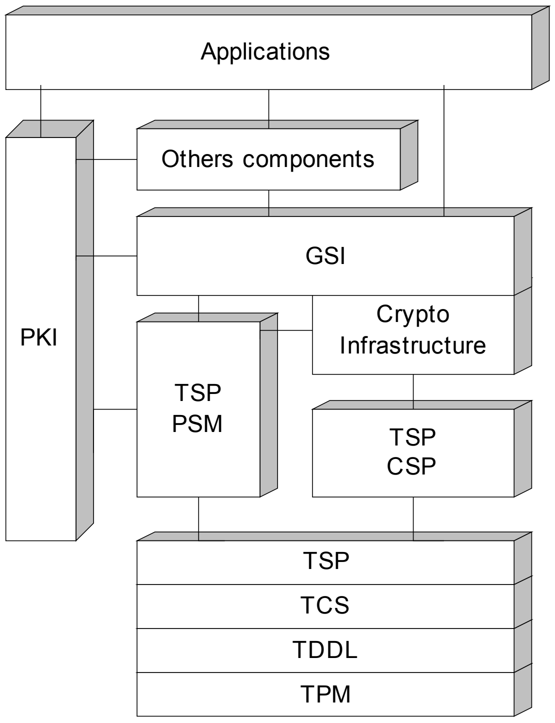

Figure 3-3. Architecture of GSI in Daonity

The modules in Figure 3-3 are as follows:

- TPM (Trusted Platform Model): This is a TCG hardware module. It provides low-level functions. These functions include input/output interface, Non-volatile storage, Attestation Identity Key, Program Code, Random Number Generator, SHA-1 Engine, Key Generation, RSA Engine, Opt-In, Exec Engine, TPM Device Driver, and so on.
- TDDL (TCPA Device Driver Library): This module provides a uniform interface for TCS module and hides possible differences of TPMs manufactured by different vendors.
- TCS (TCG Core Service): This module provides a series of services for all TCG service providers.
- TSP (TCG Service Provider): This module provides TCG services for application programs and mainly provides two functions: context management and cryptography services.
- TSP PSM (Persistent Storage Manager): This module provides storage protection service for the high-level users through using the function of cryptographic services and TSP.
- TSP CSP (cryptography service provider): This module encapsulates the cryptographic functions of the TPM and provides optional cryptographic services for Crypto API. It mainly

provides the functions of random key generation, RSA key generation, RSA encryption, SHA-1 hash, and so on.

- Crypto Infrastructure: This module includes OpenSSL API and JCE SPI. It provides encryption, decryption, signature, validation and authentication.
- GSI (Grid Security Infrastructure): This module provides the services of certificate management, authorization and authentication, etc.
- PKI (Public Key Infrastructure): This module provides the services of signature and management of identity certificates and proxy certificates.
- Other modules of Grid Platform: Other Grid Platform modules besides security modules.

3.3 Authentication

Any grid subject should be authenticated before participating in grid. User and Service, even platform need be certificated. Daonity focuses on security based on trusted computing. Authentication in Daonity includes two levels, a higher level which is for user authentication, and a lower level which is for platform authentication. Administrator can configure them.

3.3.1 User Authentication

Users submit their jobs with identity certificates. Users should logon grid portal to act as a grid user, where identity-based authentication is enforced.

3.3.2 Platform Authentication

Platform is a computer terminal through which user can access the grid service. Here platforms must have one TPM hardware and corresponding software components. Platform authenticates each other with Attestation Identity Certificates.

3.4 Authorization

Grid service that relates to sensitive resource must be managed correctly. Everyone should be authorized before access to grid service. There are two different authorization models, one is identity based and the other is attributes based. The authorization mechanism based on identity certificate converts unique ID to local account according to grid map file. The authorization mechanism based on attribute certificate (called CAS, community authorization service) converts unique ID to some attribute of the community, and then convert the attribute to local account.

3.4.1 Grid Map File

The authorization management of Globus platform is controlled by the “root” user of system, which satisfies the autonomous requirement of Grid system in some sense. On the other aspect, the overlap of grid platform controller and super user brings heavy administration burden. Moreover, because of the coarse-grain access privilege of “root” user, the authorization file (Grid Map-file) may be modified unintentionally, and then the integrity of the file can’t be guaranteed. It is necessary to provide fine-grain access restrict for Grid Map-file.
The Grid Map-file protection module utilizes the function provided by “File protection management” module (Section 4.3) which is a special module designed for the protection of critical files.
The function of Grid Map-file protection module is as follows:

1) Repeal the existing editing methods to Grid Map-file (the function is provided by the operation system, “vi” etc.), and design a special application (Protection Agent) to manage Grid Map-file. Only the modification by the special application is considered valid and permitted, and the actual and mirror Grid Map-file can be modified synchronously. Any other modification to the mirror Grid Map-file is considered invalid.

2) For any valid modification to Grid Map-file, create the log data and store it into PS, and put the digest of it into a in-TPM storage called Platform Configuration Register (PCR). (We will further detail the notion of PCR in Section 6.)

3) Related keys (encryption key, signature key …) are created by TPM and also stored in the PS of TPM.

The components of the Gridmap protection module are detailed as below:

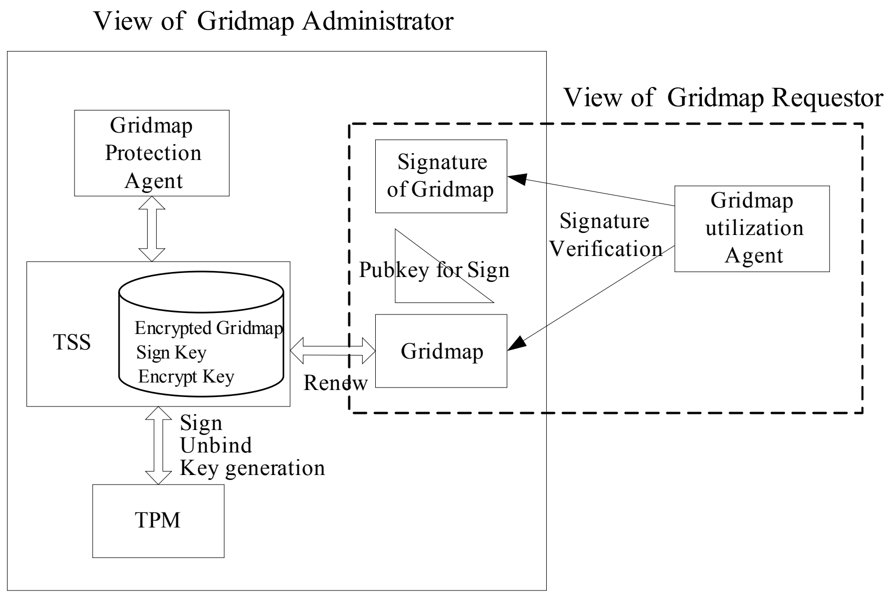

Figure 3-4. Architecture of Gridmap Protection Module

The user of Gridmap protection module is divided into two types: Gridmap Administrator and Gridmap Requestor.

The responsibility of Gridmap administrator is Gridmap modification, log creation, and resume of Gridmap file. The encryption and signature of data is achieved by TPM (call the TSS API).
The responsibility of Requestor is checking the validity of Gridmap file before making access control decision according to the entries of Gridmap. If invalid, the Requestor notifies the Gridmap Administrator to resume the file.

The two type responsibilities are respectively called “Protection Agent” and “Utilization Agent”. The use case of Gridmap protection module is as Figure 3-5.

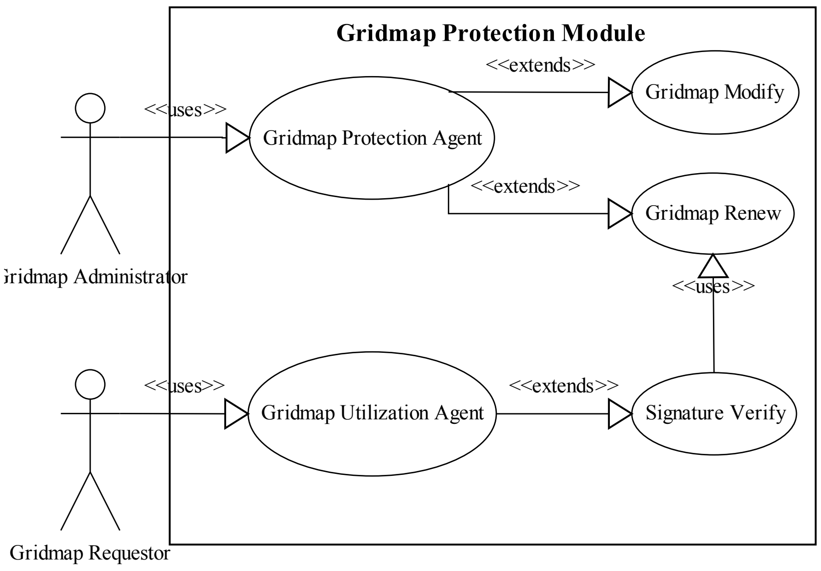

Figure 3-5. Use case of Gridmap Protection Module

3.4.2 Community Authorization Service (CAS)

Domain Attribute Authority is based on the original Globus authorization mechanism, Community Authorization Service. This service based on is an attribute based authorization model. A community is treated as a whole; and resource servers in the community grant their privileges to some attributes of the community rather than identity of a user. A user in community has two certificates, identity certificate, and community attribute certificate which is assigned by the CAS server and designated some attribute of the community. When the user access resources in the community, access control decision is made according to the community attribute in his attribute certificate.

User wishing to access a CAS-enabled resource first contact CAS server and request a CAS attribute credential, then resource server verify the credential and give appropriate rights to user which is the intersection of rights granted by the resource server to the community and the rights granted by the community to the user based on the user community attribution.

3.5 Credential Management

Credential management in Daonity focuses on the combination of Grid CA and EK CA, proxy certificates, MyProxy server and innovation of grid portal. The new notion of EK (endorsement-key) CA will be provided in 3.5.1 (b).

3.5.1 Certificate Authorities

a) Grid CA
Grid CA is an entity that issues certificates.

b) Endorsement Key CA
The endorsement key (EK) is a TCG notion. The EK CA issues the Endorsement certificate at the first use of TPM hardware after the verification of TPM hardware. An EK is a public/private key pair which is created to a new TPM (created by the TPM manufacturer). In the time of a new TPM being activated for the first time by the owner, the platform runs a EK protocol with the EK CA. The successful run of the protocol outputs an EK certificate. In all subsequent uses of the TPM, the EK certificate plays the role of the TPM's identity informing the remote user that the

communication is with a genuine TPM, rather than a, e.g., software simulation.

We need to deploy the EK CA and the Grid CA together in a TC enhanced grid security environment.

c) Privacy CA
The PCA is used for signing the Attestation certificate. Each platform can have an unlimited number of attestation certificates. An attestation certificate is an alias for an EK counterpart. The plurality of attestation certificates (keys) forms a pseudonymous way to serve a privacy need for the TPM user.

3.5.2 Credentials

a) User Certificates
A user certificate binds user's ID and the public part of an asymmetric key-pair. Grid CA will verify a certificate request and issue the certificate when receives the request from user endpoint or the grid portal. The user can select mechanisms for key pair generation. In the original GSI, the key generation uses software CSP of OpenSSL. On a trusted platform, the asymmetric key-pair can be generated by the key pair generator that is a hardware component inside the TPM. The private key generated by the TPM module is protected by a storage root key (SRK). Each TPM owner has a unique SRK to protect the confidential data and keys which are bound to the TPM and hence the platform in which the TPM is integrated (i.e., the confidential data and the key cannot be used by any other platforms). We will further detail the notion of SRK in Section 5.3.

When using portal to make the above operations, the user can even delegate the certificate request to the portal or the MyProxy Server.

b) Proxy Certificates
Proxy certificate is user's short-term certificate that is established while a user logons portal or submits job directly. Proxy certificate and user certificate have the same certificate structure; they are different in the certificate's validity time and the capability of delegation. The short-term proxy certificate replaces the long-term user certificate to complete certification and authorization to perform the job. In proxy certificate establishment procedure, the TPM module of the user or the MyProxy server can be used to establish proxy certificate safely. The proxy certificate is managed by the daemon of platform on which the submitted job is running.

The user can also give the management work of the proxy certificate to proxy certificate repository. Proxy certificate repository can complete the creating, storing and updating of proxy certificate. When user and the proxy certificate repository interact, the protection password of the proxy certificate is establish by the user. The server makes use of the persistent storage to protect all proxy certificates.

c) EK Certificates
The EK certificate provides the basic platform endorsement. At the first time using the TPM, the platform needs to obtain EK certificate from EK CA. The process of EK certificate acquisition is as follows:

-Taking out public key of EK
-Packing certificate request
-Sending request to EK CA
-Taking back the certificate

The structure of EK certificates is as follows:

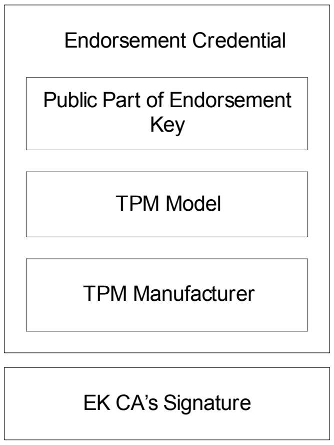

Figure 3-6. EK credential's structure

d) AIK Certificates

Attestation Identity Key (AIK) certificate is used to verify each other between trusted platforms. The process of creating AIK certificate is as follows:

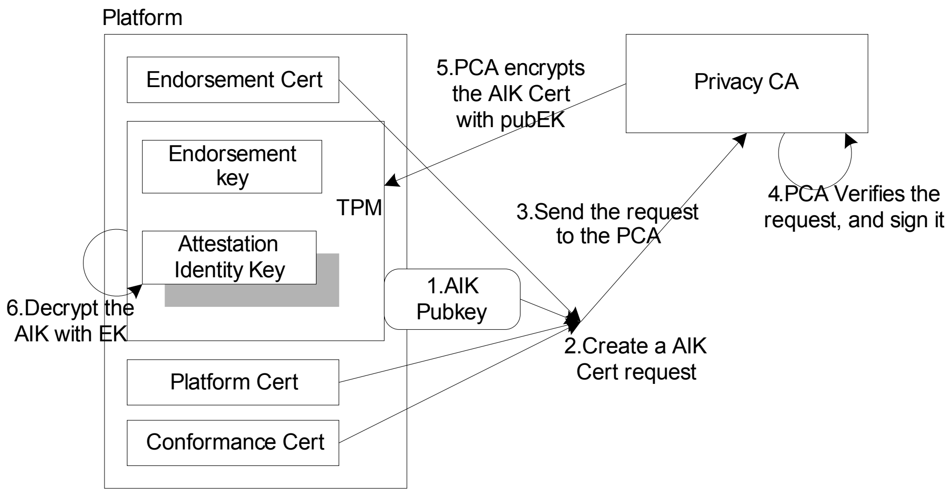
Platform
Endorsement Cert
Endorsement key
TPM
Attestation Identity Key
6.Decrypt the AIK with EK
1.AIK Pubkey
Platform Cert
Conformance Cert
5.PCA encrypts the AIK Cert with pubEK
Privacy CA
3.Send the request to the PCA
4.PCA Verifies the request, and sign it
2.Create a AIK Cert request

Figure 3-7. Flow of AIK request

The structure of AIK certificate is as follows:

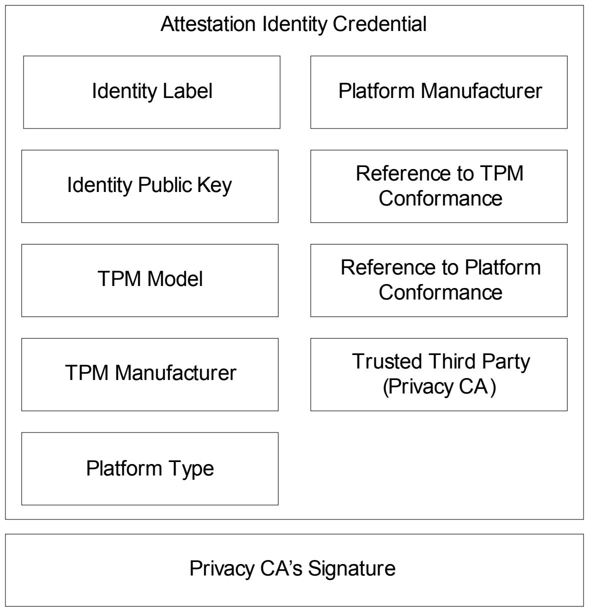
Attestation Identity Credential
Identity Label
Platform Manufacturer
Identity Public Key
Reference to TPM Conformance
TPM Model
Reference to Platform Conformance
TPM Manufacturer
Trusted Third Party (Privacy CA)
Platform Type
Privacy CA's Signature

Figure 3-8. AIK certificate's structure

e) Authentication Token

Authentication token according to the trusted proxy certificate can be used to carry out the certification and grants. Authentication token is a certificate encrypted by Authentication center's Public EK. (Probably in MyProxyServer) An authentication token shown to a logon endpoint means that the user specified in the token has been certified by the authentication center, so the endpoint can believe in the user if the authentication center is well-known.

3.5.3 MyProxy Protocols

Proxy certificate repository reserves the original function, providing the on-line proxy certificate management. Proxy certificate repository uses the management machine protected by TPM to protect the proxy certificates of the users. Each proxy certificate is protected by the TPM's authorization information for the user, while the server can't obtain the private key information. User can select the mode of protection and retrieve when he puts proxy to the repository. The protocol of proxy certificate repository based on trusted computing is as follows:

MyProxyInit <-> MyProxyServer protocol

The following illustrates a MyProxyInit process connecting to a MyProxyServer process and storing a proxy for later retrieval.

1) MyProxyInit will make a connection to the MyProxyServer at the host and port as specified by its configuration or the user.

Important: *MyProxyServer could be configured to require a trusted connection from client. When the client and the MyProxyServer have TPM component, it will be running in trusted model.*

2) MyProxyInit will initiate the GSSAPI context setup loop, with MyProxyServer accepting.

3) MyProxyInit will then send a message to MyProxyServer containing the following strings:

```
VERSION=MYPROXYv2
COMMAND=1
USERNAME=<username>
```

PASSPHRASE=<pass phrase>
LIFETIME=<lifetime>
And optional strings
RETRIEVER=<retriever_dn>
RENEWER=<renewer_dn>
CRED_NAME=<credential name>
CRED_DESC=<credential description>
The intent of the VERSION string is to allow the server to know if it is dealing with an outdated or newer client.<username> and <pass phrase> are the strings supplied by the user to be used for retrieval by the portal. <lifetime> is the ASCII representation of the lifetime of the proxy to be delegated to the portal, in seconds. <retriever_dn> and <renewer_dn> are strings specifying the retriever and renewer policy regular expressions. <credential name> assigns a name to the credential, allowing multiple credentials to be stored for a given username.

4) MyProxyServer will then respond with either an OK or an ERROR message.

5) Next, the server will perform delegation with the client. The MyProxyServer will generate a public/private key pair and send the public key to the MyProxyInit client as a certificate request. MyProxyInit client will sign the request with its proxy private key and send the new certificate and the entire certificate chain back to the server.

6) MyProxyServer will read the individual certificates, chain them back up together into new delegated credentials and store them. The server also stores the retriever and renewer DN strings if they are specified.

7) MyProxyServer will then respond with either an OK message if it successfully stored the proxy or an ERROR message if an error occurred. For example, an error might occur when the stored proxy already exists for the same username but belongs to another user.

8) At this point, both sides should close the connection.

MyProxyGet <-> MyProxyServer protocol

The following illustrates a MyProxyGet process connecting to a MyProxyServer process and retrieving a proxy for use.

1) MyProxyGet makes a connection to the MyProxyServer as indicated by its configuration or arguments.

2) MyProxyGet will initiate the GSSAPI context setup loop, with MyProxyServer accepting.

3) MyProxyGet will then send a message to MyProxyServer containing the following strings:
VERSION=MYPROXYv2
COMMAND=0
USERNAME=<username>
PASSPHRASE=<pass phrase>
LIFETIME=<requested lifetime>
The message can also contain an optional string:
CRED_NAME=<credential name>
The intent of the VERSION string is to allow the server to know if it is dealing with an outdated or newer client.<username> and <pass phrase> are the strings supplied by the user to the portal to be used for retrieval by the portal.

4) MyProxyServer will then respond with an OK, ERROR, or AUTHORIZATION message.

5) Next, the server will delegate the user credential to the client. The MyProxyGet will generate a

public/private key pair and send the public key to the MyProxyServer as a certificate request. MyProxyServer will sign the request with the private key of the stored user credential and send it as a new certificate back to the client along with the rest of the certificate chain of the stored credential.

6) MyProxyServer will then respond with either a OK message if it successfully completed sending the certificates or an ERROR message if an error occurred.

7) At this point, both sides should close the connection.

MyProxyStoreCertificate <-> MyProxyServer protocol

The following illustrates a MyProxyStoreCertificate process connecting to a MyProxyServer process and storing a proxy for later retrieval.

1) MyProxyStoreCertificate will make a connection to the MyProxyServer at the host and port as specified by its configuration or the user.

2) MyProxyStoreCertificate will initiate the GSSAPI context setup loop, with MyProxyServer accepting.

3) If MyProxyServer asks platform level authentication, GSSAPI context should be established with AIK credential before mutual authentication between user and server.

4) MyProxyStoreCertificate will then send a message to MyProxyServer containing the following strings:

```
   VERSION=MYPROXYv2
   COMMAND=5
   USERNAME=<username>
   LIFETIME=<lifetime>
```

and optional strings

```
   RETRIEVER=<retriever_dn>
   RENEWER=<renewer_dn>
   CRED_NAME=<credential name>
   CRED_DESC=<credential description>
   KEYRETRIEVER=<retriever_dn>
```

The intent of the VERSION string is to allow the server to know if it is dealing with an outdated or newer client. <username> is the strings supplied by the user to be used for retrieval by the portal. <lifetime> is the ASCII representation of the lifetime of the proxy to be delegated to the portal, in seconds. <retriever_dn> and <renewer_dn> are strings specifying the retriever and renewer policy regular expressions. <key retriever dns> is a string specifying the key retriever policy regular expression. <credential name> assigns a name to the credential, allowing multiple credentials to be stored for a given username. <credential description> can provide additional descriptive text to be displayed in MyProxyInfo requests, for example.

MyProxyServer will then respond with either a OK or an ERROR message.

5) Next, the client will send the users end-entity credentials to the server.

6) MyProxyServer will read the individual certificates and store it. The server also stores the retriever, renewer, or key retriever DN strings if they are specified.

7) MyProxyServer will then respond with either an OK message if it successfully stored the proxy or an ERROR message if an error occurred. For example, an error might occur when the stored proxy already exists for the same username but belongs to another user.

8) At this point, both sides should close the connection.

MyProxyRetrieve <-> MyProxyServer protocol

The following illustrates a MyProxyRetrieve process connecting to a MyProxyServer process and retrieving the end-entity credentials.

1) MyProxyRetrieve makes a connection to the MyProxyServer as indicated by its configuration or arguments.

2) MyProxyRetrieve will initiate the GSSAPI context setup loop, with MyProxyServer accepting.

3) If MyProxyServer asks platform level authentication, GSSAPI context should be established with AIK credential before mutual authentication between user and server.

4) MyProxyRetrieve will then send a message to MyProxyServer containing the following strings:

```
   VERSION=MYPROXYv2
   COMMAND=6
   USERNAME=<username>
   PASSPHRASE=<pass phrase>
   LIFETIME=<requested lifetime>
```

The message can also contain an optional string:

```
   CRED_NAME=<credential name/credential UUID>
```

The intent of the VERSION string is to allow the server to know if it is dealing with an outdated or newer client. <username> and <pass phrase> are the strings supplied by the user to the portal to be used for retrieval by the portal.

5) MyProxyServer will then respond with an OK, ERROR, or AUTHORIZATION message.

6) If the credential is protected by TPM, MyProxyServer will send a response asking for migration information.

7) Next, the server will retrieve the user migration credential and send it to the client.

8) At this point, both sides should close the connection.

Storage protection plays an important part in grid security. Protection mechanism based on trusted computing theory can resolve the problem of secure storage well. We need to provide a series of functions which encapsulate the low-level functions provided by TPM so that the program developers can use them easily. The BIO package in OpenSSL provides a powerful abstraction for handling input and output. Many different types of BIO objects are available for programmer. Referring to the implementation of other BIO objects, we add BIO_PS object and BIO_BLOB object to the BIO package. There are two models of storage protection in proxy certificate repository, which are illustrated in Figure 3-9.

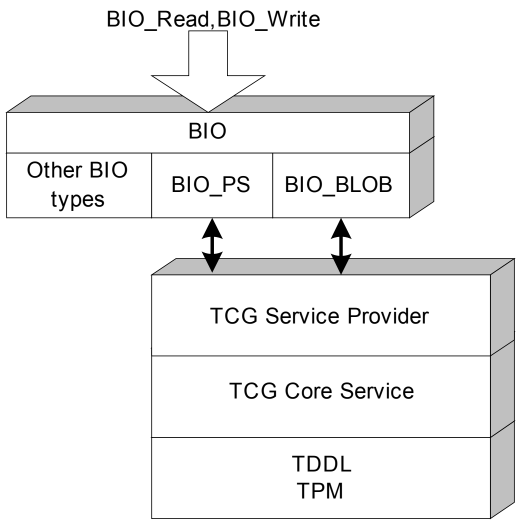

Figure 3-9. Storage protection

3.5.4 Authentication Center

For the SSO of the trusted computing, we strengthen the MyProxy server by providing functions similar to the authentication center.
The authentication center provides the guarantee of the platform's user identity information. User use this authentication token to attest himself to the remote machine.

3.5.5 Distributed Proxy Agent

In grid computing environment, machines and the members in the same domain has higher dependability. The trusted platform with a TPM becomes the distributed proxy agent, providing the possibility to share the trusted computing resource. The proxy agents take the work of proxy certificate repository largely, so all users in the domain can use the trusted computing platform to work.

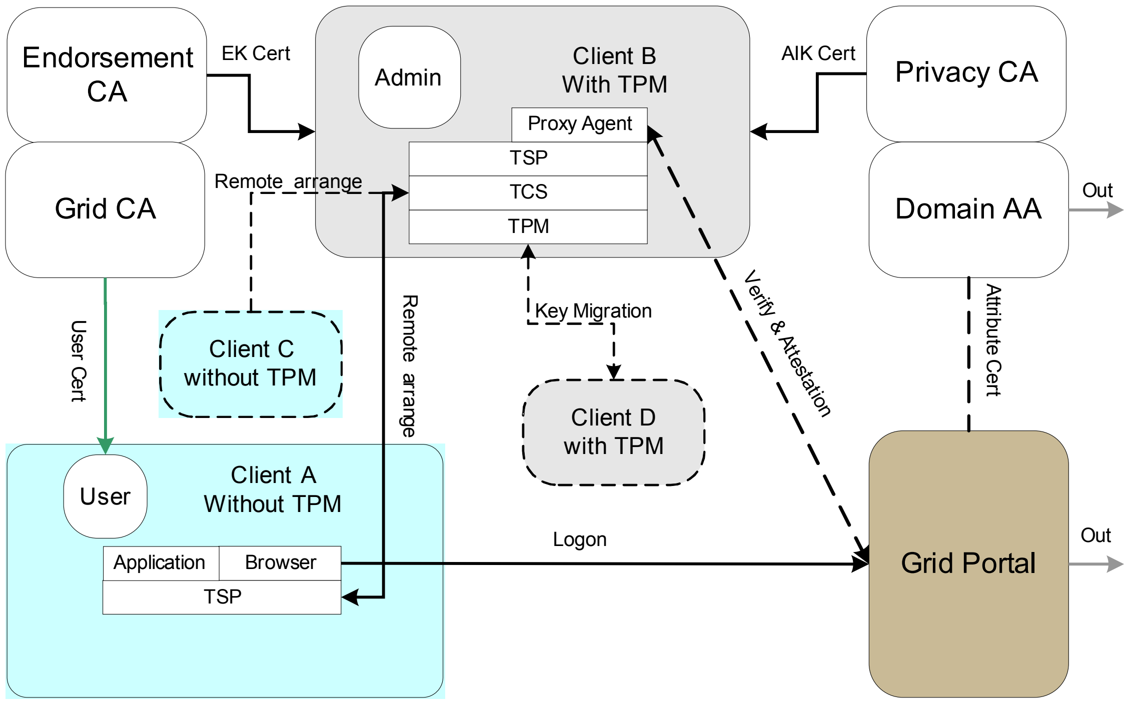

Figure 3-10. Distribute Proxy Agent

In Figure 3-10, the domain has two kinds of machines. Client A and Client C has not TPM, client B and D has TPM. When user Alice is performing on client A, she may select client B to act as a proxy agent for her. The proxy agent in client B is a daemon process for anyone in the same domain, which is upon the TCG software stack. The agent contacts with the Grid portal while user Alice wants to submit a job from the Grid portal.
There are three new protocols in Daonity include Remote Arrange, Key Migration and Attestation. Remote Arrange protocol is defined for clients different in TPM status. It works between local TSP layer and remote TCS daemon.
Attestation protocol is not detailed here.

3.5.6 Portal

a) User Registration

User registration is done on grid portal; user need submit enough personal information. After the approval of grid managers, a requestor becomes a legal grid user with an identity credential. The registration policy (accept the user or not) is determined by grid policy. Using trusted platform or not can be specified by administrator.
Important*: Here we assume that the grid environment uses the portal as its only entrance and anyone should be the user of portal website*

b) Platform Registration

The legal platform of grid environment can request the platform EK credential from grid portal at the beginning. And this operation needs platform administrator authorization – there is no limitation that platform administrator must be a user of grid. The platform should provide proper platform information and credential description while requesting credential.

3.6 Single Sign-On

There are two modes for single sign-on: a client based and a server based.

3.6.1 Client Based

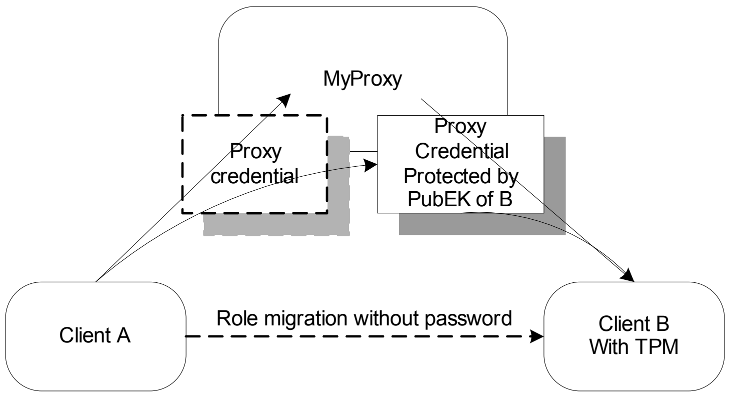
MyProxy
Proxy credential
Proxy Credential Protected by PubEK of B
Client A
Role migration without password
Client B With TPM

Figure 3-11. Client Based Single Sign-On

The proxy certificate that its related private key is protected by TPM does not have the single sign on function because the private key can't be used without pass phrase any longer. Daonity designs a new usage of certificate for SSO. As Figure 3-11 shows, the user on Client A can access the MyProxyServer and delegate a proxy certificate. When delegating his proxy certificate to the MyProxyServer, the user can specify which destination the proxy certificate will be used on. The destination may be a common host, a certain organization or some host with special attribute. Thus, MyProxyServer can encrypt that proxy certificate with the destination's public key which can be any public key binding to the destination platform. The proxy certificate includes the user's privacy information that is revealed to the destination. The destination will access MyProxy server directly and obtain the legal proxy certificate according to the user information. The destination decrypts the encrypted proxy credential, and if the identity in the decrypted proxy certificate actually accords with logon user, the destination will consider that the user is the right one. This mechanism realizes the single sign-on for the user.

3.6.2 Server Based

Server based model provides more security than client based model. User can restrict the usage of proxy certificate binding to a destination. For further exploiting the TPM platform, we design another SSO mechanism shown in Figure 3-12. This model needs some improvement in the MyProxy server that an authenticate server is added. It also needs that the MyProxy server has a TPM and corresponding certificates.

When user signs on the improved MyProxy server from client A, the server authenticate user's identity first, the authentication may base on user identity certificate. After this step, user can delegate a proxy certificate to MyProxy server. The proxy certificate is encrypted by the platform certificate private key of MyProxy server, and then returned to the user. As the Figure 3-12 described, the user will hold the encrypted proxy certificate that act as authentication token when user access other terminals, e.g. Client B. The terminal can authenticate the user's identity by verifying the encrypted proxy certificate.

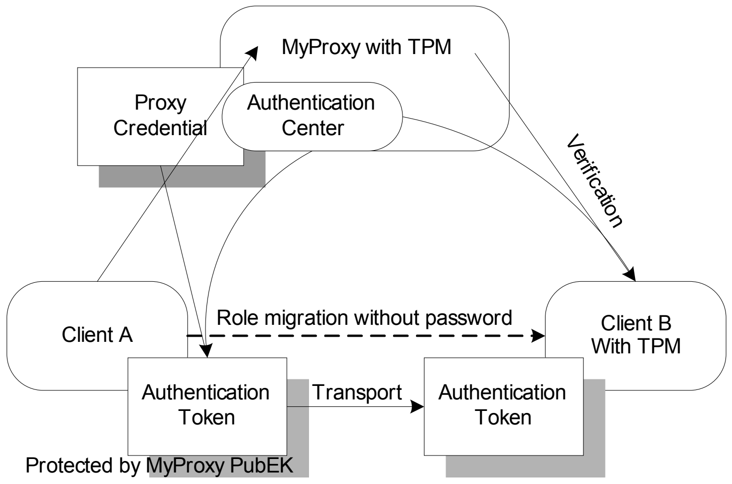

Figure 3-12. Server Based Single Sign-On

The assumption of this model is that users can hold his proxy certificate in security. User can store the encrypted proxy certificate in secure USB equipment or personal hand-hold equipment like PorKI. In sign-on port, the authentication to user identity is the authentication to MyProxy server. In this way, server based model achieves SSO function.

4. GSI Context Manager

There are two kinds of context in the TSS architecture: TSP context and TCS context. The context object in TSS environment is similar to the process context maintained by an operating system. TSP context contains information about the application's execution environment, such as the identity of functional modules; secret shared among secure modules and communication among remote TSS modules. Each object also relatively has its context handle to share knowledge.
The relation between context and other modules is very important for understanding how the context manager works. According to TSS specification, in TSP layer, each module is considered as an object, so we describe this relationship in object-oriented like language.

4.1 Context Relationship

In TSP layer, several types modules are defined as the following objects; we simplified the modules into five parts:

4.1.1 Context Class

This class contains information about the application's execution environment, such as the identity of the working object and the transaction/communication with other TSS-Software modules (e.g. TSS-Core-Service)

4.1.2 Policy Class

This class is used to configure policy settings and behaviors for the different user applications.

4.1.3 TPM Class

One purpose of the TPM object is to represent the owner for a TCG subsystem (TPM). The owner of a TPM is comparable with an administrator in the PC environment. For that reason there exists only one instance of the TPM class per context. This object is automatically associated with one policy object; which must be used to handle the owner authentication data.

4.1.4 Key and Cryptography Service Class

This class is used for encryption and decryption. And to join externally (e. g. user, application) generated data to a TCG-aware system (bound to PCR or Platform).

4.1.5 PCR Composite and Hash Class

This class provides a comfortable way to deal with PCR values (e.g. select, read, write). An object handle of such a class is used from all TSP functions that need PCR information in its parameter list.Provides a cryptographically secure way to use these functions for digital signature operations.

When current Application needs to use one class, its relative object will change to the **working object**. We use handle (Such as Context handle, Key handle, TPM handle, and etc) to access working object.

4.2 TSP Context

TSP Context realized the connection between application and TSS. Application creates context by calling Context_Create function, and then, it can call functions related to working objects in TSP, described in Figure 4-1.

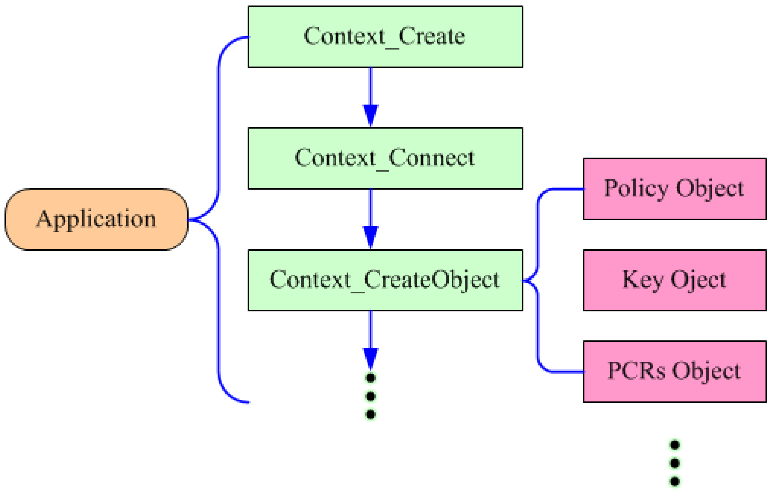

Figure 4-1. Create working object through Context

In TSP, Context makes relation with working objects by context handle, which are described as Figure 4-2:

Figure 4-2. Object Relationships in TSS

The relationship is described as follows:
 Application and Context
One application is corresponding to a context handle.

 Context and TPM

When context have been created, there is a TPM object is also created by using this context handle. So, they are corresponding to another.

- Context and Working Objects

One context handle may be corresponding to any working Objects. If Working Objects does not be created, there is no relation between Working Objects and context, which means Zero.

- Context and Policy

After creation of a context, a default policy is created and each new created object is automatically assigned to this default policy. The default policy for each working object exists until a new policy object is assigned to the working object.

- Policy and Working Objects

A new policy may be assigned to several objects like key objects, encrypted data objects or a TPM object. Each of these objects will utilize its assigned policy object to authorize TPM commands. TSP context object includes the secret mode (such as popup, callback and so on). When secret is needed to set, it must find the current TSP context handle, and then get the secret mode on which setting secret must base.

In every object, the TSP context handle is only corresponding to the TCS context handle. If one object needs to work from TSP to TCS, it must verify the TCS context handle to see whether it exists and whether it's corresponding to the TSP context handle. If verified, the object can work in TCS.

4.3 Memory Management and Context

Memory management in the TSP is based on a context object. Each allocated memory is associated with the TCS that the current TSP context is associated with. This will enable easy cleanup of memory allocated by a TCS for one or more TSP contexts that may close unexpected.

This rule can be applied to protect remote computing platform against memory overflow or memory leakage. We can define a clear memory boundary before application running, verify memory usage when resource is shared and release memory after task is over. It will lead to a more trusted grid-computing environment.

4.4 TSS Core Service Management

TCSD is a management process of the TCG Core Service, its functions detailed as follows.

- Manage TCS Service and
- Deal with TSP requests and call related modules

The role TCSD plays in TSS structure and the interactions among the components of TSS are described in Figure 4-3.

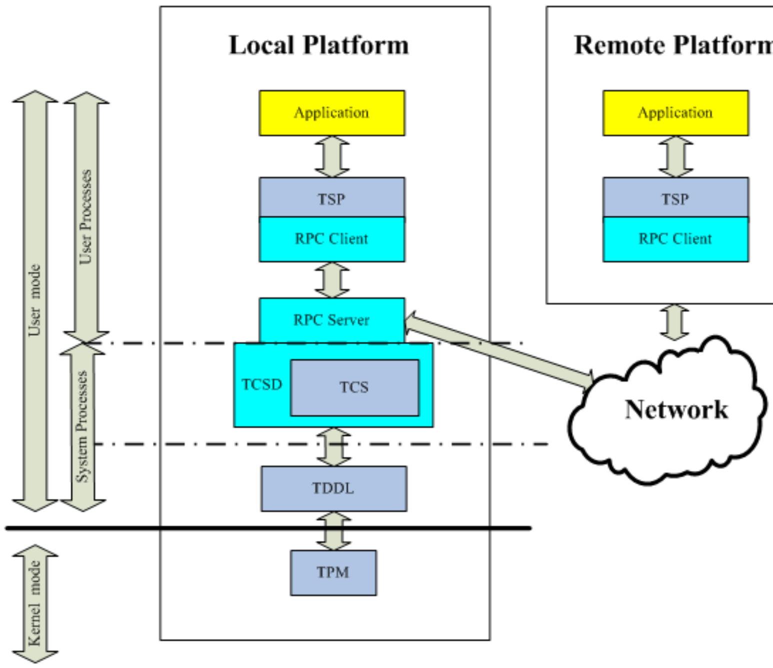

Figure 4-3. Relationship between TCSD and other TSS components

TCSD is divided into two parts: TCSD management component and TCSD RPC server component. The main components of TCSD and the flow chart can be seen from the figure Figure 4-4 below:

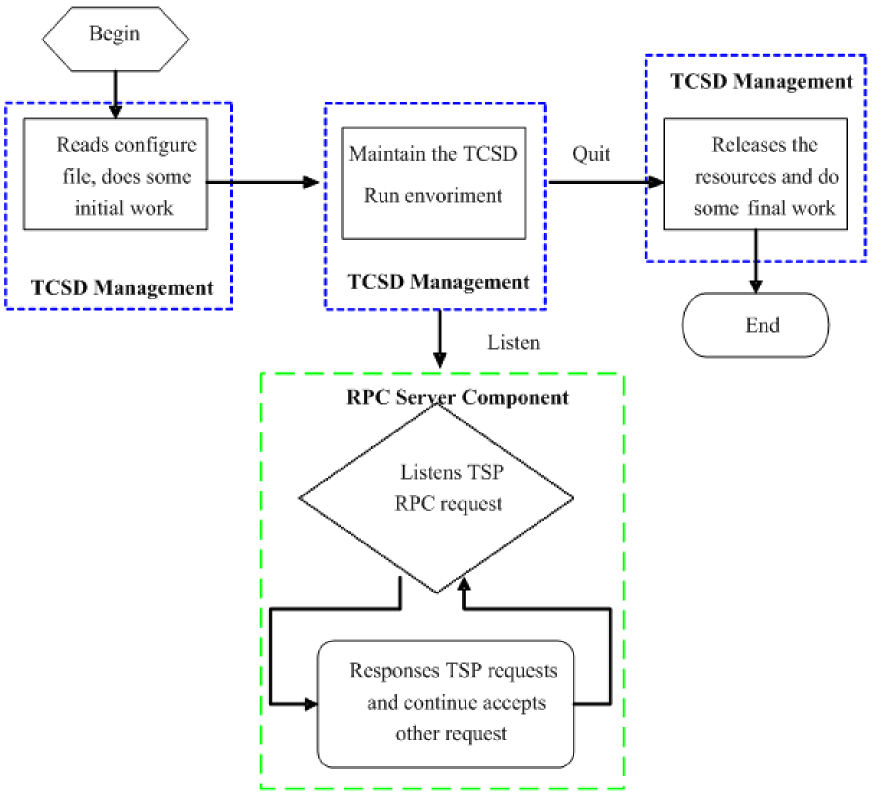
Begin
Reads configure file, does some initial work
TCSD Management
Maintain the TCSD Run envoriment
TCSD Management
Quit
TCSD Management
Releases the resources and do some final work
End
Listen
RPC Server Component
Listens TSP RPC request
Responses TSP requests and continue accepts other request

Figure 4-4. Workflow of TCSD

4.5 Design attention of the TCSD

TCSD acts as TCS service manager and maintains the resource related with TCS. In our design, TSP finds the TCS service number, sets the request data in a packet and sends to TCSD uses RPC way, and then it is blocked for result. TCSD receives the TSP request, gets the request data, find the TCS service table using the TCS service number, and sends the request data to related TCS services. After TCS service dealing with the request, TCSD gets the result from TCS service, sets the data to a packet and sends it to TSP. The TCSD is multi-thread and it can deal with multi request. User application thread which call TSP is waked up after receives the TCSD packet.

We separate the TCSD design in two parts:

The first part is TCS service management. It contains three parts:

1. TCS service initialization. Including reading configure file, doing some initial work such as PS region, cache and log.

2. TCS service maintains. Including managing related data and structure.

3. TCS service quit. Including releasing resource and save some data

The second part is TCSD RPC server. Including data exchange between TSP and TCSD, data exchange between TCSD and TCS, TCS service description and thread mechanism. So we design the packet format and other data structure.

The special attention should be paid for some factors that are listed below:

- Daemon process of system. TCSD is the daemon process of system, acts as a system service. So it is important to design the TCSD from system service layer. The dependability is the most important.

- Configuration of TCS. In order to dynamic change some parameters after TCSD has been installed, the TCS service should be configurable. We use the configure file to set and get some parameters.

- Security. The TCSD is a system service, it has to read and write the files, so it is important to set the user right and group right correctly to protect the files and TCSD security, prevent privacy data from revealing.

- Working mechanism. According to the TSS specification, TCS should be able to response multi TSP requests and remote TSP requests. So we must consider RPC and multi-thread. We use RPC method and multi-thread to design TCSD, in the application layer we will use Web Service to describe and find TCS service in future.

- Resource management. When starting and stopping TCS service, related resource should be allocated and released, log event should be written. Because of the multi-thread mode, we also have to consider the mutually exclusive resource to prevent dead-lock.

- Authorization between TSP and TCSD. The TCSD and TSP communicate from RPC method. Though some platforms have no TPM, it can use TCS service by using shared TPM mode. So it is important to design a secure protocol to ensure application layer and transfer layer communication security, described in Figure 4-5 as follows.

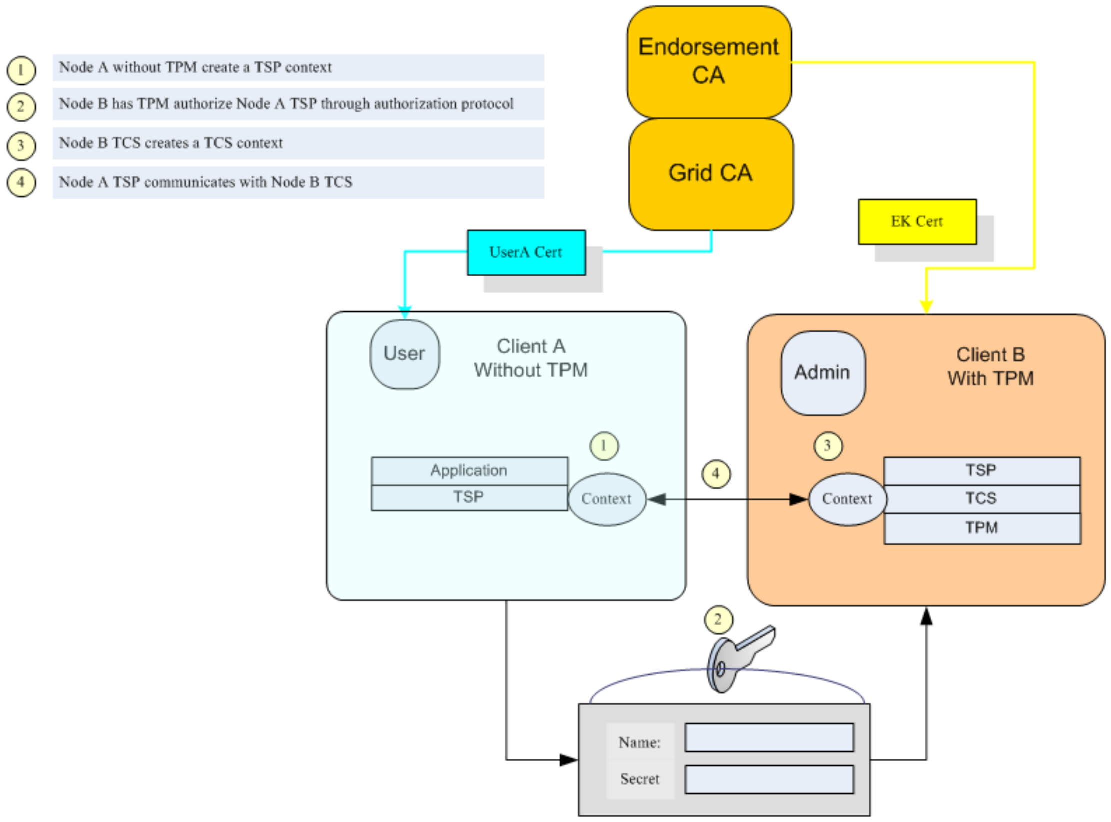
1 Node A without TPM create a TSP context
2 Node B has TPM authorize Node A TSP through authorization protocol
3 Node B TCS creates a TCS context
4 Node A TSP communicates with Node B TCS
Endorsement CA
Grid CA
EK Cert
UserA Cert
User
Client A Without TPM
Application
TSP
Context
Admin
Client B With TPM
TSP
TCS
TPM
Context
Name:
Secret

Figure 4-5. TPM-Shared Communication

5. Key Management

In this section we describe the design of the key management system.

5.1 Term Definitions

Name	Description
Application key handle	key handles used by application
key slot cache	the cache is used to storage the key handle which pointed the address in the TPM
Identity key	it is non-migratable signing key that are exclusively used to sign data originated by the TPM
UUID	a sign for the key, it is a structure including timelow , timemid, timehigh, bClockSeqHigh, bClockSeqlow
Opaque data	the secret data such as the private key, the data which has not been encrypted
Key type	TCG defines 7 key types: Signing key, Storage key, Identity Key, Endorsement Key, Bind key, Legacy Key, Authentication Key
KCM (Key Cache Manage)	Handles key-caching whenever required. The Key cache manager typically uses TPM_SaveKeyContext and TPM_LoadKeyContext for the key caching
TCPA SK	TCPA Specific Knowledge, it provide PBG with ordinal ,tag ,etc
KCMS	(Key Cache Manage Storage)The storage of the KCM
PBG	The Parameter Block Generator uses “TCG Specific Knowledge” to concatenate its input parameters and other parameters to a TPM Parameter Block command

BS KEY	“Byte Stream” format structure of a TCG Key

5.2 Overview

The Key Manager Services allow definition of a persistent key hierarchy. The persistent key hierarchy consists of storage keys that make up the base storage key structure that will exist before any user may attempt to load a key. Additionally the persistent key hierarchy may contain system specific leaf keys as for instance identity keys.

All keys, which should be internally managed by the Key Management Services of TSS, must be registered in the storage spaces of TCS (system) Persistent Storage (PS) or TSP (User) Persistent Storage. Each key registered in one of these PS will be referenced by its UUID (a sign of the key for short) and called a persistent key in the view of TSS.
Keys once registered in PS will keep effective in PS unless they are unregistered or migrated. Migration is another condition. When the key is loaded in the TPM, The PS will be valid all the time.
When the TCS is not restarted or the key is not evicted from the Key Cache Manager Service, Application key handles which got from a load key command are usually valid, and it will not stay valid across boots.
Using the key Management supported by TSS will simplify the whole mechanism of loading a key into the TPM from a calling context’s point of view. And it will enhance the security because the TPM in computer protect the private information. Because the private information which is encrypted by the user `s private key is shielded to the owner of the TPM.

5.3 Model of Key Storage

The Root of Trust for Storage (RTS) protects keys and data entrusted to the TPM .The RTS manages a small amount of volatile memory where keys are held while performing signing and decryption operations. Inactive keys may be encrypted and moved off-chip to make room for other more active keys. Management of the key slot cache is performed external to the TPM by a Key Cache Manager (KCM). The KCM interfaces with a storage device where inactive keys may be stored indefinitely. The RTS doubles as a general purpose protected storage service allowing opaque data also to be stored.

There are three key types that are not opaque to the TPM. They are AIK keys, Signing keys and Storage keys. Key types will be discussed in more detail later. Two keys are embedded in the TPM, see the below Figure 5-1, the Storage Root Key (SRK) and the Endorsement Key (EK), which are embedded in the TPM. These keys cannot be removed from the TPM.

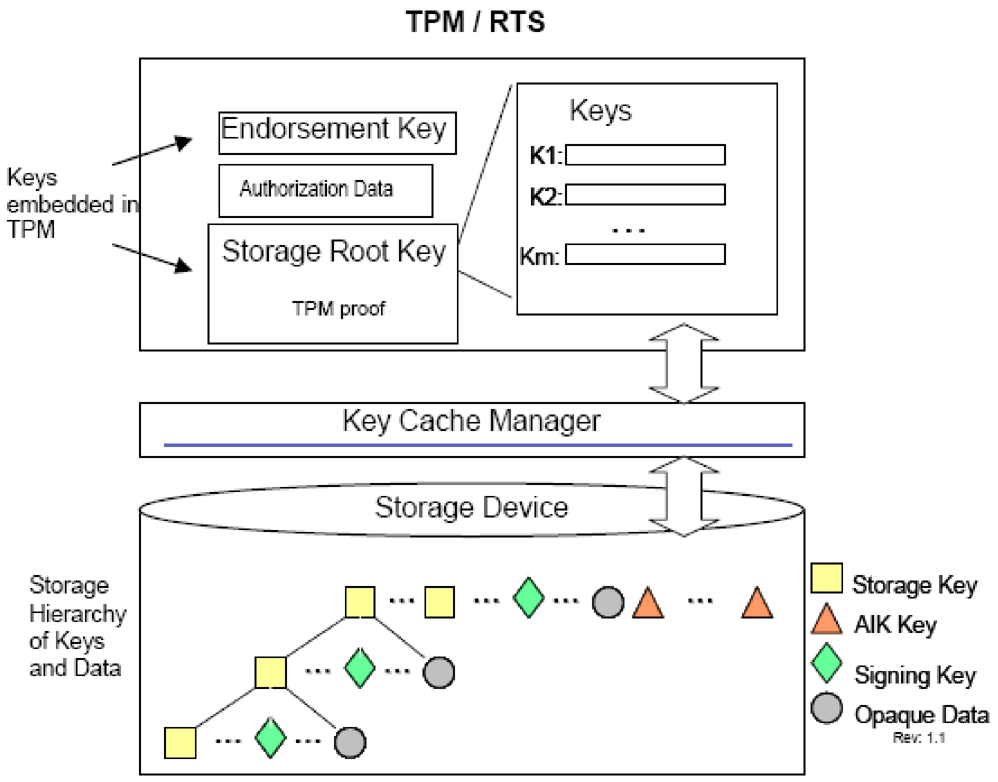

Figure 5-1. Root of Trust for Storage (RTS) Architecture

TCG defines 7 key types. Each type carries with it a set of restrictions that limits its use. TCG keys can be classified broadly as either signing or storage keys.
The 7 key types are listed as follows:

- *Signing keys* are asymmetric general purpose keys used to sign application data and messages. Signing keys can be migratable or non-migratable. Migratable keys may be exported / imported between TPM devices.
- *Storage keys* are asymmetric general purpose keys used to encrypt data or other keys. Storage keys are used for wrapping keys and data managed externally
- *Identity Keys* (a.k.a. AIK keys) are non-migratable signing keys that are exclusively used to sign data originated by the TPM (such as TPM capabilities and PCR register values).
- *Endorsement Key* (EK) is a non-migratable decryption key for the platform. It is used to decrypt owner authorization data at the time a platform owner is established and to decrypt messages associated with AIK creation. It is never used for encryption or signing.
- *Bind keys* may be used to encrypt small amounts of data (such as a symmetric key) on one platform and decrypt it on another.
- *Legacy Keys* are keys created outside the TPM. They are imported to the TPM after which may be used for signing and encryption operations. They are by definition migratable.
- *Authentication Keys* are symmetric keys used to protect transport sessions involving the TPM.

5.4 The Hierarchy of the Key Storage

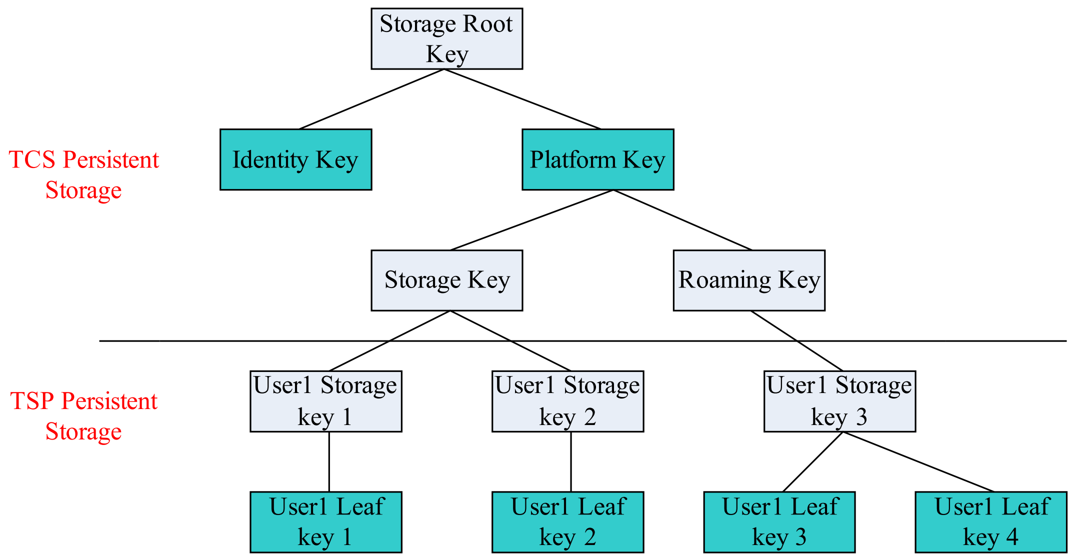

Figure 5-2. Hierarchy of the key storage

The same color keys in the key hierarchy diagram above are mandatory storage keys and are addressed by fixed UUID, they have the same attributes (e.g. migratable, auth) and are stored either in the persistent storage of TCS or the persistent storage of TSP on all platforms. Keys stored in the user specific persistent storage of TSP can be addressed by the same UUID for each user but of course the UUID will still reference a different user storage key.

5.5 Key Management in TSS

We now describe the TSS key management techniques which follows the TCG specifications.

5.5.1 Key Cache Manager

The TCS Key Cache Manager Service (KCM) allows caching keys to manage the restricted resources of a TPM. The KCM is responsible to manage the restricted resources of the TPM and to hide these restrictions from the calling applications. An application can load a key in to the TPM by utilizing the KCM functionality and can assume that this key is available for further use. The KCM is responsible to ensure that a key, which has already been loaded by an application, is available in the TPM, when the application requires that key for a certain command. If all TPM resources are in use, the KCM has to free resources in order to load a key or to get the required key back into the TPM.

5.5.2 The Structure of Key Cache Manager

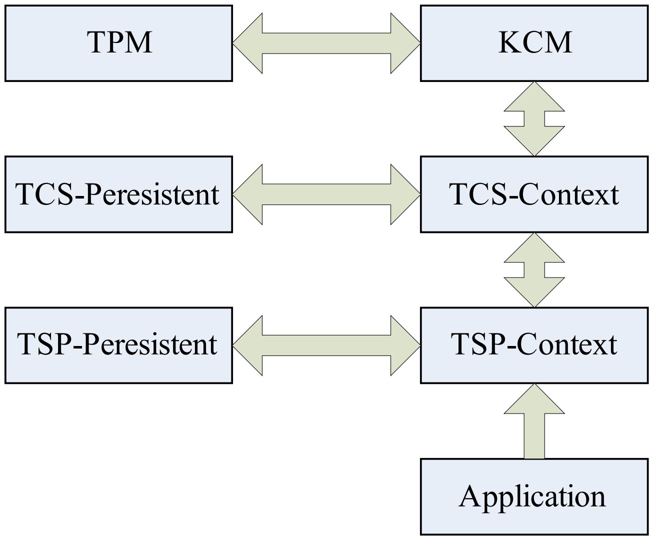

Figure 5-3. Key Management Architecture

An application must load a key into the TPM by utilizing the KCM. The KCM returns an application key handle to the caller and manages a mapping mechanism between the returned application key handle and the actual TPM key handle. The actual TPM key handle will change whenever a key has to be unloaded from the TPM by the KCM in order to free resources since another key has to be loaded and the KCM reloads the key into the TPM again. The application key handle returned to the calling application remains constant as long as the key is not reloaded by the application itself. The KCM may implement a model for indexing; storing and retrieving Blobs contained on KCM managed storage devices. This may also include management of pass-phrases necessary for using keys in the TPM. The structure of the map include the information of the key storage such as UUID, cache flag, offset address in blob, public data size, blob size, public key, UUID, TCS-handle, TPM-handle.

5.5.3 Persistent Storage in TSS

As defined in TCG specification, TPM can act as a portal to keep arbitrary amounts of data and keys confidential. But the room in TPM is limited. So, we should offer Persistent Storage space as a service to functions outside the TPM. The TCG Software Stack (TSS) enables such a service.

This enables applications to provide functions such as user association, key archive, and key restoration, and enables the efficient migration of Subsystem (subsystem) information from one platform to another within a heterogeneous PC environment. For a user application the persistent storage looks like a data archive, therefore the main function set is associated to such function sets.

There are two Persistent Storages in our design. The first is TCS Persistent Storage. The other is TSP Persistent Storage. To be a criterion, we use the unify form to them. We define two files, the index file and the content file. The index file stores indexes, which are convenient to search the detailed information of the key. The index has three members: the first is the Nickname of the user, the second is the UUID of the key, and the last is the address of the key in the Content file. The Content file has the public information of the key, such as UUID, Parent UUID, Public Data, Key size, Cache flag, Public Key, the offset address in the blob.

5.5.4 Cryptography Service in TSS

Original GSI fully depends on the cryptography service provided by OpenSSL. For improved security requirements, we provide a design depending on the trusted hardware and OpenSSL.

There are two types crypto service in our design: asymmetric and symmetric crypto service, For asymmetry key, we will use the encryption and decryption mechanism of RSA. TPM can provide the RSA mechanism for the application. The private key stored in TPM cannot be seen out as plaintext.

Considering TPM's specification, symmetry crypto service still needs other modules out of TPM. In our project, we choose OpenSSL. Since symmetric keys can be protected by some asymmetric keys and asymmetric keys protected by TPM, symmetric crypto service is obviously more secure.

5.6 Key Cache Manager Functions for TPM's Interface

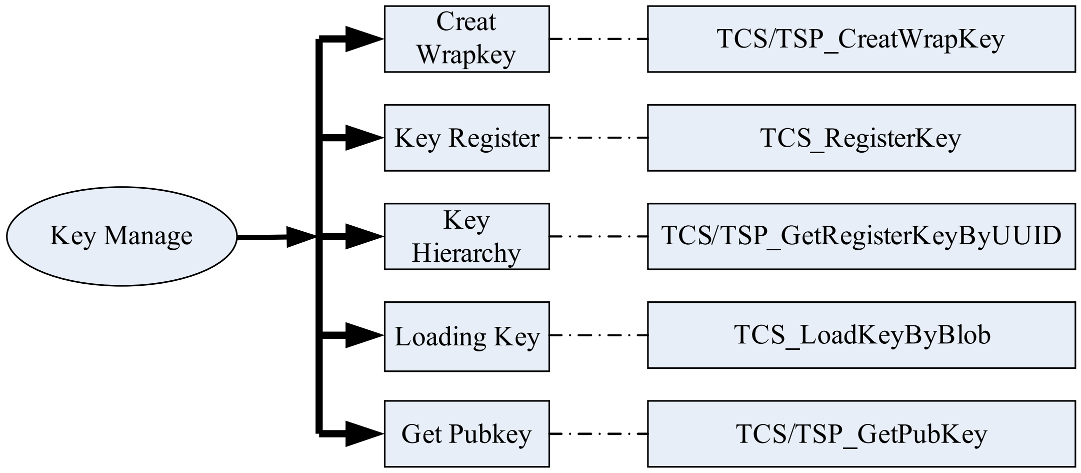

Figure 5-4. Key Cache Management

5.7 TSS Load Key Flow Diagram

Load key Flow Description as described the following diagram

Case 1: TSP_LoadKeyByUUID, Key registered in TSP Persistent Storage (TSP PS), parent key authorization is not required. Key registered in TSP PS means it is a user key, it should search in the TSP PS to find its information (uuid, cache flag, blob address and so on) and wrapping key, we do not concern the auth. So, if get its wrap key (a system key), then we should know it is loaded in the TPM or not, this information can be found in the KCM by the function TCS_LoadkeyByblob, If not, we will fail to load it or loading the wrapping key first. It just like a tree, you should look for his parent node from the leaf node. When the wrapping key has been testified in the KCM .It selects the key and its wrapping key information to the PBG.PBG put them to TPM in the form of string at the command TPM_Loadkeycontext. Then this key can be loaded in the TPM.

Case 2: TSP_LoadKeyByUUID, Key registered in TCS Persistent Storage (TCS PS), parent key authorization is not required, the progress is similar to the Case 1, the difference is the key is a system key, it storages in the TCS PS. So it need not to research it in the TSP PS, it only look into the PS in TCS to find its information (uuid, cache flag, blob address and so on) and wrapping key by the function TCS_LoadkeyByUUID , we do not concern the auth, and the later is the same to Case 1.

Case 3: TSP_LoadKeyByUUID, Key registered in TSP Persistent Storage (TSP PS), parent key authorization is required. We can find the abstract information about parent key, and some parameters, then the Black Box operate them to export auth data, and it can call the TCS_LoadkeyByblob to visit KCM. Then the later is the same to Case1.

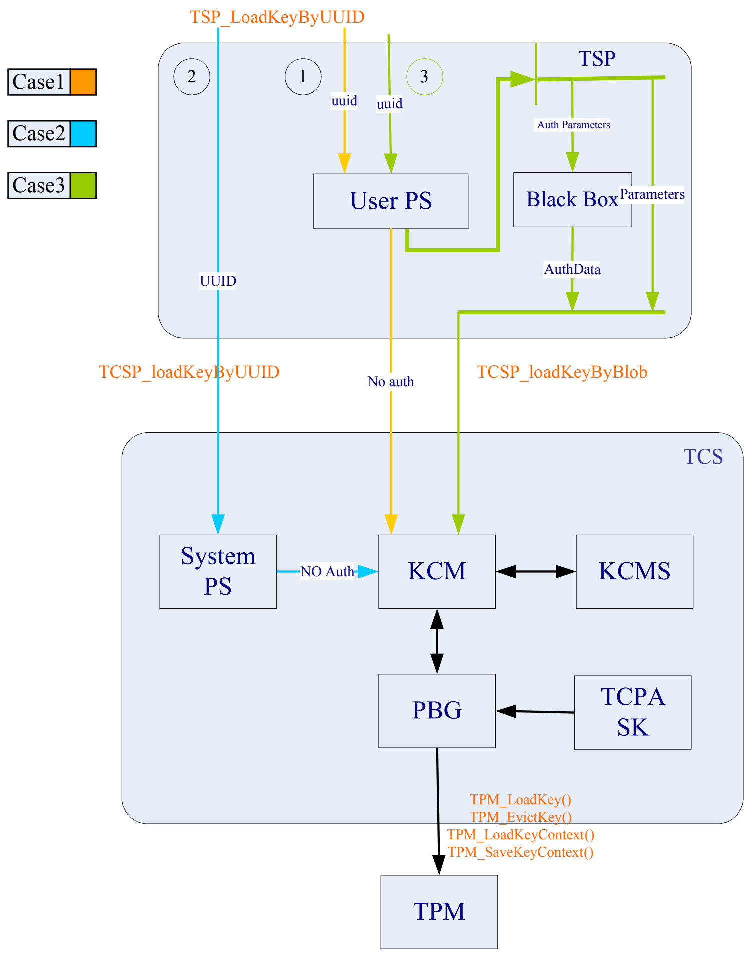

Figure 5-5. Load Key Flow

5.8 Key/Credential Migration

We now describe the necessary services of key/certificate migration.

5.8.1 Background

In grid system, user's keys/credentials can be transmitted from one platform to another. For example, if a grid user wants to log into a grid system in different sites, he should input his credential into the application system at according site for the authentication. This kind of operation is supported by TPM that is used to improve the security of grid. We call this mechanism enable by TPM the Key/Credential Migration.

Grid is a large-scale dynamic distributed system. Users and resource change dynamically. Because users may want to log in grid from different sites, the authentication for a user should be processed on different sites. In order to improve the security in grid, we design to use TPM to enable the grid system. So, the authentication of user can be processed in different sites with TPM. Since the credential validation is necessary for authentication, the user's credentials should be able to migrate from one site to another both with TPM. In our design, even though user sometimes wants to log in grid from a site without TPM, the user can process the authentication by accessing another site with TPM and using the TPM in that site for the credential validation. Then the migration is also needed in this condition.

5.8.2 Solution

In order to improve the security in grid system, we design some security mechanisms by using TPM in grid. Considering the Key/Credential Migration, we also manage to use the TPM enabled implementation.

Firstly, we describe the key migration in the following. The TCG specification defines some mechanism for the key migration. Migratable keys may be exchanged between TPM devices. This enables the key pair to follow the grid user around regardless of the device he uses. Messages exchanged between entities remain accessible even though the computing platform changes in grid environment. Storing data outside the TPM has the additional advantages of enabling easier migration of confidential data from one platform to another and enabling recovery of confidential data in the event of platform failure. These capabilities also are designed to avoid the need for the TPM to manage the confidential data that is stored outside the TPM.

In the key hierarchy in PS, the migration tree is directly below a “migration root” key that is directly below the SRK. The key hierarchy in PS is described in the section of Key Management. Each node in a tree provides confidentiality for the nodes immediately below it. Obviously, all intermediate nodes in the trees must be encryption keys. Any migratable key can be migrated by anyone that owns any of its migratable ancestors. As a result, in order to be sure that a migratable key cannot be migrated by anyone but the owner of that key, the owner can always create the migratable key and store it with a non-migratable storage key, thus guaranteeing the user has unique authority to authorize migration of that key.

An encryption (storage) key or a signing key, which is migratable, can be stored in PS. If a key is for encryption, it must not be used for signing, and visa versa. Encryption keys are used only to provide confidentiality for blobs. Signature keys are used for signing arbitrary data submitted by the entity authorized to use that key. Migratory data may be copied to an arbitrary number of platforms.

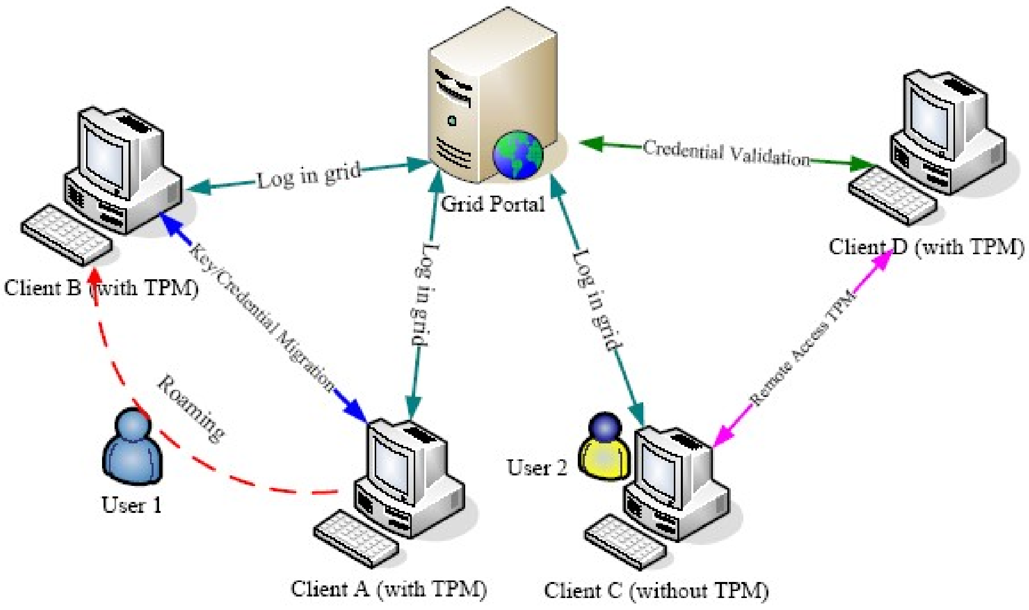

Figure 5-6. The Key/Credential Migration in Grid Environment

Secondly, we describe the Credential Migration in Grid as in Figure 5-6. Sometimes, if a grid user wants to log into grid from different site, his credential should be input into the application system on the site where he log in grid. In our design, we use the TPM to improve the security in grid. The client system is designed to work in a platform enabled with TPM. So, when a user (User 1 in Figure 5-6) hopes transmitter his credential to another platform, he could use the transmission mechanism enabled with TPM to process this operation. As described above, the TPM enable the key migration. Then the credential migration is also can be done with this mechanism. User1 firstly migrate his credential from client A to client B. Then he roams to client B and log in grid from client B. In the TSS layer of our design, we provide software module to interact with TPM to fulfill the key/credential migration. In some other condition, people hope to log in grid at a site without TPM. The user (User 2 in Figure 5-6) can firstly migrate his credential to a site with TPM (Client D in Figure 5-6). Later, user2 could access remotely client D from client C and fulfill the credential validation by using the TPM in client D. Then he logs in grid from client C without TPM. The TSS provides a TSP context in client to support these operations.

6. File Protection Management

In this section we detail the security service for file protection management.

6.1 Background

Grid is an open dynamic large-scale distributed system. There are many dynamic resources and users that may interact with other entities. In this kind of environment, some critical files of grid system may be destroyed or leaked by illegal user or cracker. Then the confidentiality, integrity and reality may be breached, and the grid system will be not an available system for users. For example, if the GridMap file in grid is leaked to cracker, they may know the important authority information and even alert them to obtain illegal usage for the grid resource. So, the protection for critical files should be provided in any grid system. The confidentiality and integrity of them is the basic factors to be considered.

The function of file protection is to protect the critical files in grid systems. In our design, these files to be protected are stored in specific protected storage (PS), such as persistent storage spaces in hard disk. These protected files are managed by particular program modules enabled with TPM in the grid system.

6.2 PCR

Clearly, our systems must be prevented from tampering. It will promote system's security that we protect our systems building on hardware. As we know, trust in a platform is built bottom-up, starting at the base with Trusted Platform Module (TPM) hardware bound to the platform's motherboard. So we can use TPM's secure characteristic to store important messages.

Thus, for system's integrity, we can use TPM's PCRs to store platform integrity measurements in a way that prevents misrepresentation.

PCR values can play important role in: auditing operations, verifying integrity of logs, resuming of protecting files.

We can hash some audit events' data using SHA-1, regard it as measured data, and calculate it using the formula: PCR[new] = SHA-1 (PCR[old] + measured data), and then TPM can securely store the new PCR value in PCR within the TPM. When we need to use the external audit events' data, for verify measurement events, we can hash it and compare it's hash value with PCR value, the external audit events' data cannot be used unless a PCR value is the same as it's hash value. If the external data had been tampered, its hash value will different from PCR value and it will not be adopted, because the PCR value, which stores in TPM, is secure. So we can verify external data's integrity using PCR.

The TPM contains a set of registers, called Platform Configuration Registers (PCR) containing measurement digests. These registers are big enough to contain a Hash (currently only SHA-1).

6.3 Protect Critical Files by PCR

The TCG defined some protected storage mechanisms rooted in hardware. Those mechanisms can then be used to protect keys, secrets and hash values. In order to protect those critical files in grid environment, we manage to use the function of TPM, such as the encryption, signature and verification of integrity. The TCG defined some protected storage mechanisms rooted in hardware. Those mechanisms can then be used to protect keys, secrets and hash values. We design some software modules to implement these functions for file protection. The functions of data encryption and integrity verification are provided by TPM. These functions are fulfilled mainly in file protection modules in the TSS layer, which interacts with TPM. The file to be protected is taken as data stream and sent down to TPM by TSS. The data is encrypted in TPM and sent back to TSS. Then these encrypted files are stored in PS and can only be accessed by authorized entity. This process provides the confidentiality protection for critical files. In order to fulfill the integrity protection, we use the PCR in TPM. The PCR is a group of registers, called Platform Configuration Registers (PCR) containing measurement digests. The actual GridMap file is stored in PS. User can operate on the mirror of GridMap file directly. The sequence of operations is logged and processed by PCR. The final operation for actual GridMap file will be compared with the PCR value. Then correctness of the operation can be judged and the integrity of the actual GridMap file will be achieved. The detail descript of PCR's functions for file protection is described in the later section of PCR.

6.4 Intensive Solution

In order to explain the function of PCR, we can give an example of its usage for protecting files.

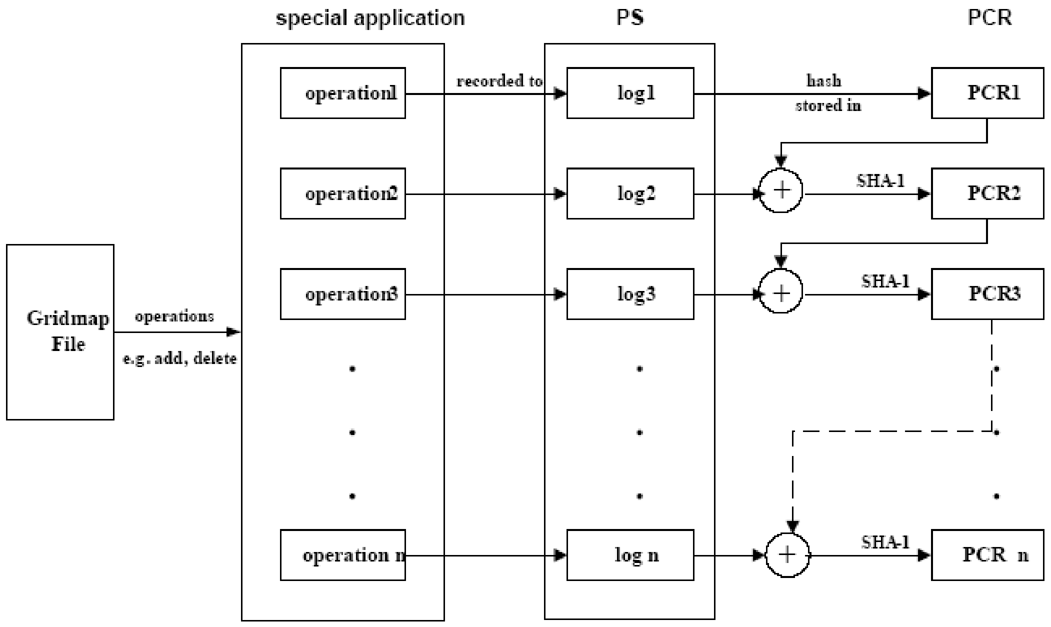

Figure 6-1. The Function of PCR for Protecting Files

For example, when the authorized users/application want to modify the GridMap file, the operation is did on the mirror of GridMap file. The mirror file is plain text that can be read and operated by users directly. The actual GridMap file is stored securely in PS. The file protection module will record a user's operation sequence for the mirror of GridMap file, and modify the actual GridMap file in PS in the end. As in Figure 6-1, when the Gridmap file needs to be changed, we should use a special application to handle the operation (e.g. add, delete). These metrics are stored in logs (Stored Measurement Logs), and digests (hashes) of them are put into

PCRs. When an operation is to be done on the mirror of GridMap file, a log that records the operation will be created. We store the logs in PS (Persistent Storage), synchronously, hash it and then store its hash value in PCR. When next log comes into being, we calculate a new PCR value using PCR [new] = SHA-1 (PCR [old] + new log data), and replace the old PCR value using the new one. The PCR is in TPM and can't be tampered by illegal user or Hacker. Only operations authorized legally are recorded by PCR. The application system will record all the operations, including potential illegal operations, on the mirror of GridMap file. Later, as in Figure 6-2, when the operations for GridMap file in PS are to be done finally, we can compare the final operation sequence, which have been recorded by application system, with the values in PCR. If they are same, then the operations to be done is legal, otherwise there are illegal operations in the operation sequence. So, we can know whether the final operations on actual GridMap file in PS is right. If they are right, we can use them affirmatively. Thus it can be seen, PCR-related have two hash properties:

• Order is important - not commutative => (A || B) != (B || A)
• One way only – it's infeasible to determine input from a resulting digest

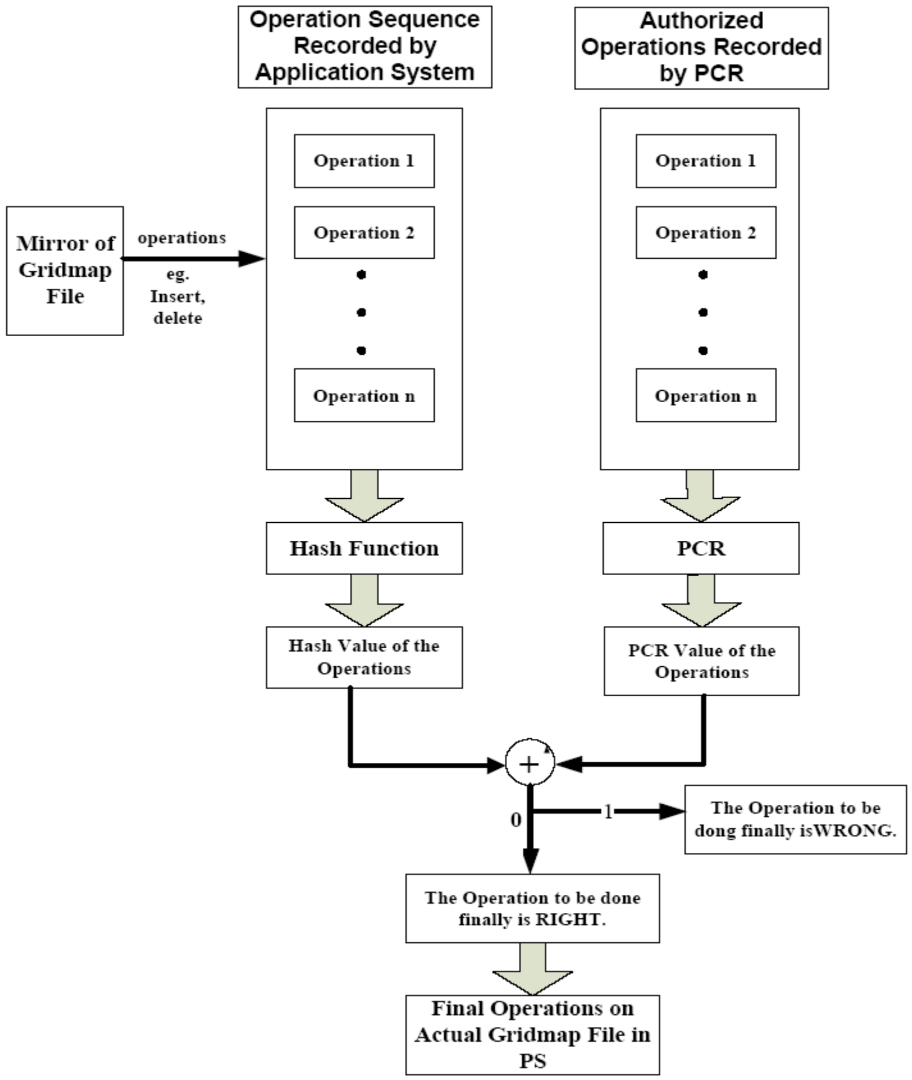
Operation Sequence Recorded by Application System
Authorized Operations Recorded by PCR
Mirror of Gridmap File
operations
eg. Insert, delete
Operation 1
Operation 2
Operation n
Operation 1
Operation 2
Operation n
Hash Function
PCR
Hash Value of the Operations
PCR Value of the Operations
+
0
1
The Operation to be dong finally isWRONG.
The Operation to be done finally is RIGHT.
Final Operations on Actual Gridmap File in PS

Figure 6-2. The compare between PCR values and operations recorded in application system.

7. TPM Management

To implement the management of TPM from top to bottom, we need TPM object as the representatives of applications which use TPM functions, TCS Parameter Block Generator (TCS PBG) to interpret TCS functions used by TSP interface into available form for TPM, and TCG Device Driver Library (TDDL) as a user mode interface between the TCS and TPM Device Driver (TDD).

7.1 TPM class

The TSP Interface defines the following seven classes:

- Context class
- Policy class
- TPM class
- Key class
- Encrypted Data class (sealed or bound data)
- PCR Composite class
- Hash class

TPM class is mainly used to act on behalf of the owner of a TCG subsystem (TPM), which just likes the administrator in a PC environment.

As the owner of a TPM would be the only one to take charge of the TCG subsystem and execution of owner-privileged operations, TPM class should automatically be set with a default policy to deal with owner verification, provide only one instance per context and basic control/reporting functionality accordingly.

There are three aspects that need to be clarified: the owner, identity and credentials of a TPM.

The owner of TPM can execute some privilege commands such as taking ownership of the TPM, which need shared secret and use the authorization protocols (Object Specific Authorization Protocol or Object Independent Authorization Protocol) to prove knowledge of that secret.

One TPM can have many different identities (i.e., an identity used by TSS), and a Privacy CA or several Privacy CAs must attest each of them separately for validation.

The procedure of establishing a TSS identity can be set for three phrases: creating of a new identity, contacting the Privacy CA, and activating the new identity. Endorsement credential, platform credential, conformance credential, as well as the public key of the Privacy CA are required in the meantime.

7.1.1 Definition of Basic Structures

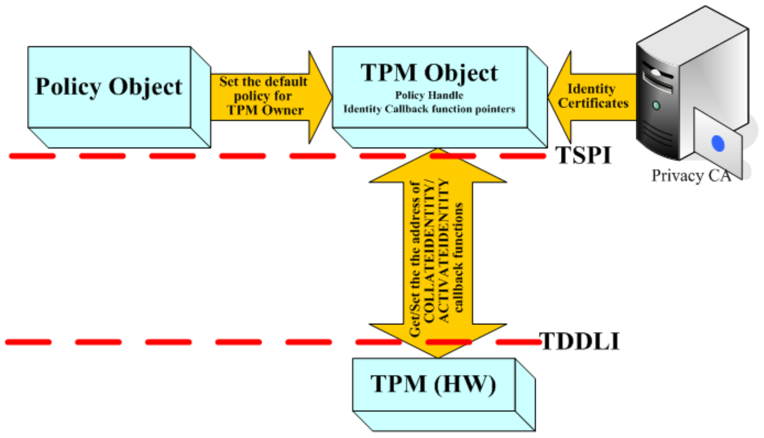
Policy Object
Set the default policy for TPM Owner
TPM Object
Policy Handle
Identity Callback function pointers
Identity Certificates
Privacy CA
TSPI
Get/Set the the address of COLLATEIDENTITY/ ACTIVATEIDENTITY callback functions
TDDLI
TPM (HW)

Figure 7-1. How a TPM Object interacts with others

TPM Object

TPM object is a data structure containing all the necessary resources to act on behalf of the TPM owner. TPM object needs a default policy to describe the TPM user's authorization state, as well as the callback function pointers to deliver the TPM identity information to calling applications or interact with the Privacy CA which would issue the proper identity certificates to the trusted platform. The following diagram illustrates how a TPM object interacts with other resources.

TPM List

TPM List is a one-way list data structure which contains all TPM objects used in TSS architecture. The following picture

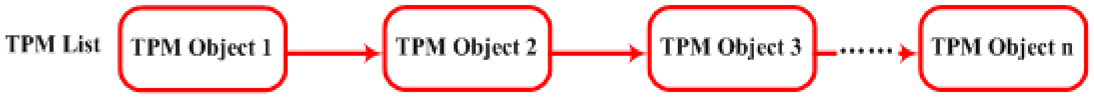

Figure 7-2. TPM List

7.1.2 Management of TPM Object

The management of TPM objects can be classified as the following seven situations:

- Adding a TPM Object
- Judging a TPM Object Handle
- Assigning an existing policy to a TPM object
- Getting the assigned policy to a TPM object
- Getting the corresponding tcsContext for a TPM Object
- Getting the corresponding tspContext for a TPM Object
- Searching a specific TPM Object from TPM Object List

7.1.3 TPM class Specific Methods in TSPI

TPM class specific methods in TSPI consist of two kinds of methods. One is common method (see also Design of Context), which set or get TPM object's attributes, while the other is TPM object related method, which interacts with a TPM object.

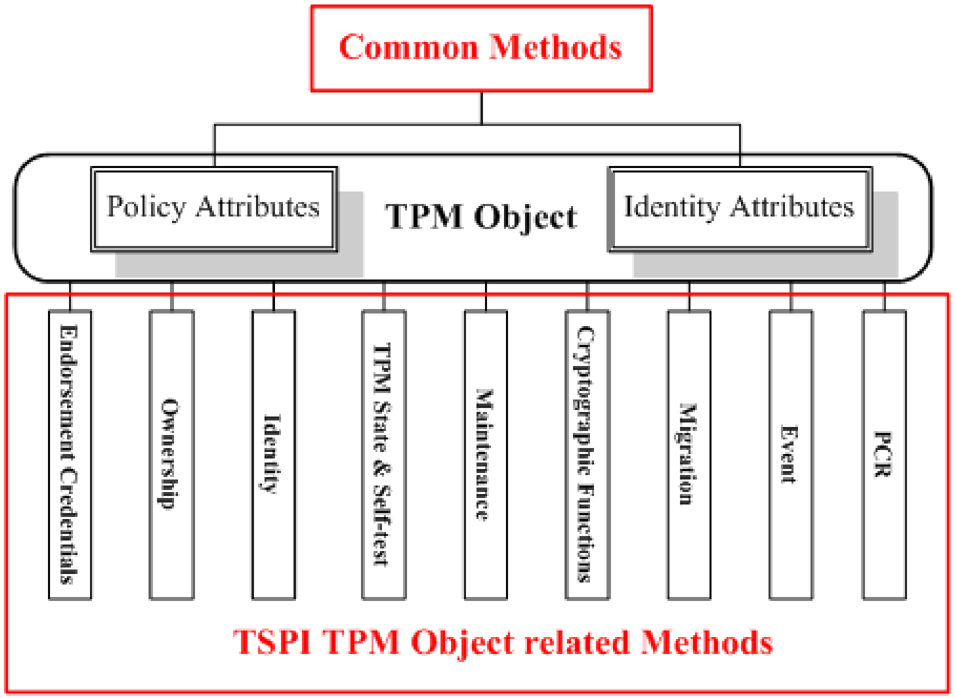

Figure 7-3. Classification on TPM class Specific Methods in TSPI

7.2 TCS PBG Functions

TCS consists of TCS PBG and other function modules. PBG is the only direct access to TPM device, and builds byte streams to input to the TPM. Besides, it is used to serialize, synchronize, and process TPM commands.

7.2.1 TCS PBG working flow

TCS PBG's working can be divided into three parts. The first part verify TCS context with TCS context manager, then PBG can deal with authorization. It may wrap the inputs before commands got to TPM and unwrap the outputs back to calling functions.

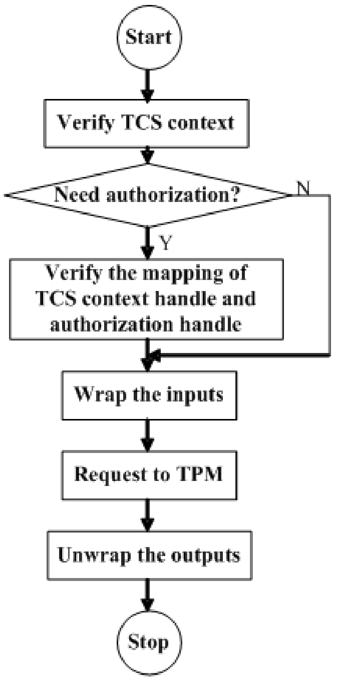

Figure 7-4. TCS PBG working flow chart

7.2.2 TCS PBG Interface

Since TCS PBG controls the only access to TPM, it has to interact with many TSP layer resources, such as TPM object, policy object, data object and so forth. TCS PBG can be divided into twelve sets according to the function, which are demonstrated in the following diagram.

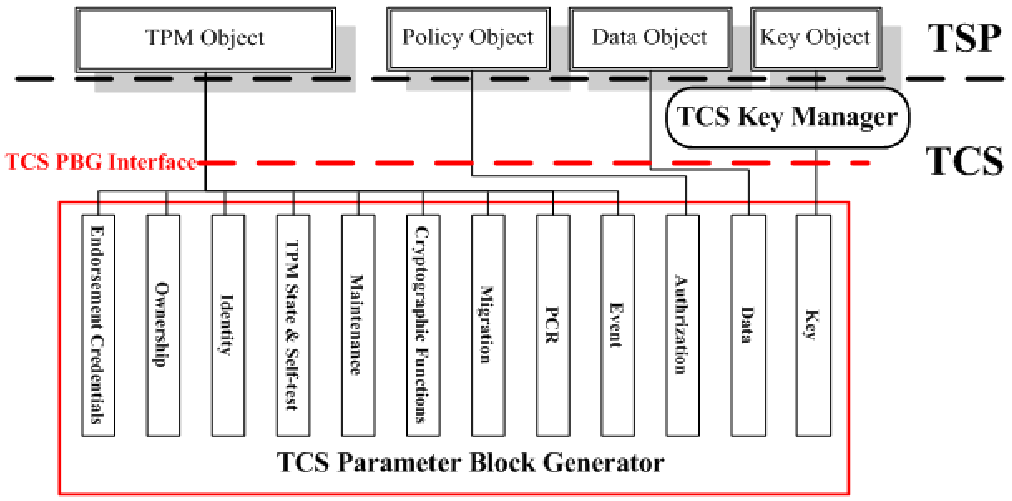

Figure 7-5. TCS PBG Functions and its Interaction with other Components

7.3 TCG Device Driver Library (TDDL)

The TCG Device Driver Library (TDDL) resides in the user mode, and provide the only connection to the TCS.

TDDL Interface
TDDL Interface contains three most important functions which are Tddli_Open, Tddli_Close,

Tddli_TransmitData. These functions can be mapped into open, close and read/write functions in TPM Device Driver under Linux kernel mode.

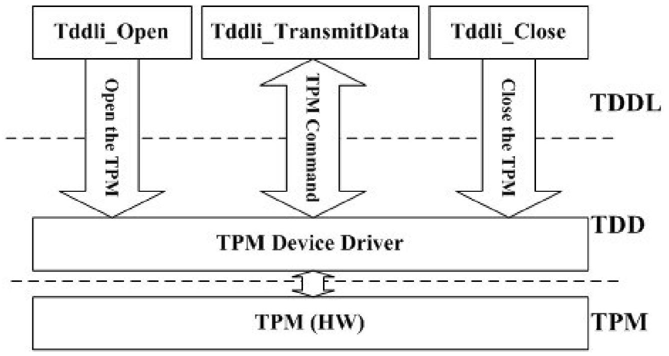

Figure 7-6. How TDDL interact with TPM Hardware

8. Session Authorization Management

In this section we detail the security services for session authorization management.

8.1 Motivation

As we known, our privacy or reveal platform secrets must be authorized. Authorization means the caller must supply a secret as part of command invocation.

For preventing attackers form using our privacy data, we should have authorized operation when we use some important private information. For example, some TPM commands require authorization; and owner-related commands normally require authorization based on knowledge of the owner authorization 160-bit secret. Similarly, the use of keys may require authorization based on the key's authorization secret. Normally, this is done in the form of a hash of password, or PIN, applied to the key when it is created.

The TPM supports two protocols for this authorization: Object Independent Authorization Protocol (OIAP) and Object Specific Authorization Protocol (OSAP). The two protocols authorize the use of entities without revealing the authorization data on the network or the connection to the TPM. In both cases, the protocol exchanges nonce-data so that both sides of the transaction can compute a hash using shared secrets and nonce-data. Each side generates the hash value and can compare to the value transmitted. Network listeners cannot directly infer the authorization data from the hashed objects sent over the network.

The OIAP allows the exchange of nonce-data with a specific TPM. Once an OI-AP session is established, its nonces can be used to authorize the use any entity managed by the TPM. The session can live indefinitely until either party request the session termination.
The OSAP allows establishment of an authentication session for a single entity. The session creates nonces that can authorize multiple commands without additional session-establishment overhead, but is bound to a specific entity. In a word, OIAP is used to create long-term sessions and can be used across multiple objects within a session. OSAP is used to create sessions that only deal with a single object.

8.2 Solution

There is no requirement for the application to initialize any OIAP or OSAP authorization session. The application is simply needed to offer its operational objects to the TSP. So the TSP hides the management of TCG related authorization sessions from the calling application.

The TSP initializes a required authorization session, based on objects which application has offered, and handles all internal data of that session. In detail, for authorization session (OIAP or OSAP), the TSP must obtain some necessary information from related objects' policy settings, which contain specialized secrets handling for the authorization. And then, the TSP transmits the information to the TCS. The TCS will add TPM command (TPM_OIAP or TPM_OSAP) to the information, and then transmit the information in the form of data-stream to TPM. When TPM receives the data from TCS, it will start authorization sessions via the TPM command. Concrete flow is described as follows (An example of TCS connecting TPM with OIAP session):

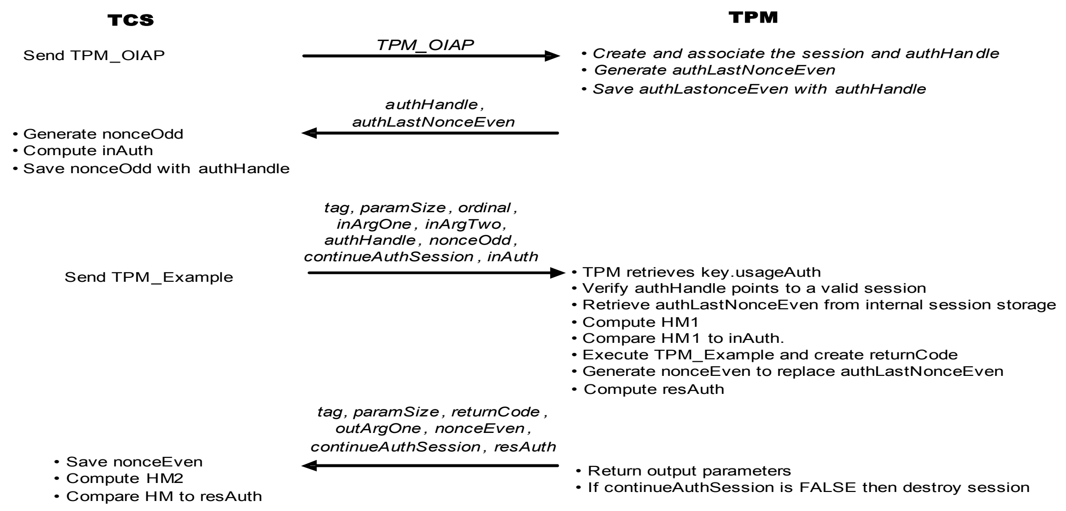
TCS
TPM
Send TPM_OIAP
TPM_OIAP
• Create and associate the session and authHandle
• Generate authLastNonceEven
• Save authLastonceEven with authHandle
authHandle, authLastNonceEven
• Generate nonceOdd
• Compute inAuth
• Save nonceOdd with authHandle
tag, paramSize, ordinal, inArgOne, inArgTwo, authHandle, nonceOdd, continueAuthSession, inAuth
Send TPM_Example
• TPM retrieves key.usageAuth
• Verify authHandle points to a valid session
• Retrieve authLastNonceEven from internal session storage
• Compute HM1
• Compare HM1 to inAuth.
• Execute TPM_Example and create returnCode
• Generate nonceEven to replace authLastNonceEven
• Compute resAuth
tag, paramSize, returnCode, outArgOne, nonceEven, continueAuthSession, resAuth
• Save nonceEven
• Compute HM2
• Compare HM to resAuth
• Return output parameters
• If continueAuthSession is FALSE then destroy session

Figure 8-1. TCS connects TPM with OIAP Session

In the above flow, we need to use the following formulas:

inAuth = HMAC (key.usageAuth, inParamDigest, inAuthSetupParams)
resAuth = HMAC(key.usageAuth, outParamDigest, outAuthSetupParams)
HM1 = HMAC (key.usageAuth, inParamDigest, inAuthSetupParams)
HM2 = HMAC(key.usageAuth, outParamDigest, outAuthSetupParams)

Above description is about authorization between the TCS and TPM. Similarly, we can solve the secure problem of communication between the TSP and TCS using some protocols like OIAP and OSAP.

When a remote TSP calls a local TCS, we can exchange nonce-data between them. They both generate the hash value. Via comparing the value transmitted, system judge whether the communication is secure. Attackers cannot directly infer the authentication data from the hashed objects sent over the network. Communication between TSP and TCS is described in TCSD part.

9. Security Considerations and Analysis

This is a REQUIRED section (to be added in a near future).

Author Information

The Daonity Team:
HP Labs China, Beijing 100022, China
Wuhan University, Wuhan, China
Huazhong University of Science and Technology, Wuhan, China

Contact Author: Wenbo Mao, Principal Engineer, Hewlett-Packard Laboratories, China, HP Building, 112 JianGuo Road, Beijing 100022, China.
wenbo.mao@hp.com
TC-Grid@chinagrid.edu.cn

Glossary

(To be added in the final version).

Intellectual Property Statement

Full Copyright Notice

References

(To be added in the final version).