Enforcement Architecture and Implementation Model for Group-Centric Information Sharing

Ram Krishnan
George Mason University
rkrishna@gmu.edu

Ravi Sandhu
University of Texas at San Antonio
ravi.sandhu@utsa.edu

Abstract

A fundamental requirement for Secure Information Sharing (SIS) is that protection needs to extend to clients. Trusted Computing Technology provides a hardware root of trust through the Trusted Platform Module. We present a super-distribution based enforcement architecture and implementation model for the group-centric SIS problem which is concerned with sharing information within a set of authorized users. With super-distribution, group subjects can encrypt objects once and distribute them via any means such as Email, USB flash drives, P2P, WWW, etc. Other authorized group subjects may access these objects without having to contact an authorization server to download the object. A Trusted Reference Monitor on client platforms faithfully enforces group policies.

1. Introduction

Secure Information Sharing (SIS) or sharing information *while* protecting it is one of the earliest problems to be recognized in computer security, and yet remains a challenging problem to solve. The central problem is that, in a distributed system, copies of digital information are easily made and controls on the original typically do not carry over to the copies. One main roadblock was the lack of hardware-rooted trust on client platforms. The advent of Trusted Computing Technology [1] and the feasibility to verify the trustworthiness of software running on trusted hardware (e.g., [10]) has made it possible to address SIS. Here, a Trusted Platform Module (TPM), a security chip that can store keys and provide trusted capabilities, is integrated in the system in an inseparable manner. With this hardware-root of trust, access control can now be trustworthily enforced on client platforms where the content is decrypted and displayed, so as to ensure that only authorized users get to see the content and that they are unable to make plaintext unprotected copies. There has been considerable interest in this approach, initially driven by the forces of digital rights management for entertainment content seeking to protect revenue but more generally seeking to protect content for its sensitivity. An overview of SIS motivation and solution approaches was presented in [4].

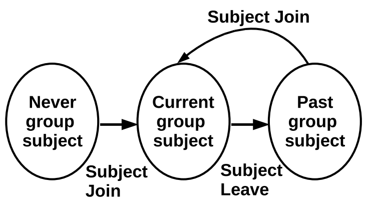


Figure 1. Subject Membership States.

We present an architecture and TPM-based implementation model for a group-centric SIS (g-SIS) problem that enables Super-Distribution [7]. The g-SIS problem [3] is motivated by the need to share sensitive information amongst a group of authorized users. For simplicity, we only consider read operations and addition of new objects to the group in this paper. In g-SIS, subjects may join, leave and re-join the group (figure 1). Similarly, objects may be added, removed and re-added to the group. Access decision is made based on the temporal relation of subject and object membership periods. For example, a subject joining a group may only access objects that are added after join time. In another scenario, the subject may also be allowed to access objects added prior to join time. The metaphor is that of a secure meeting room (although virtual) where members' access depend on their participation period. For example, in a program committee meeting, when discussing Alice's paper, she may be required to step out the room and thus cannot access discussions during her absence. In another scenario, the discussions during Alice's absence may be recorded on a whiteboard and she may access them on her return. Our architecture can support a wide variety of such policies.

In the context of g-SIS, Super-Distribution (SD) refers to widespread distribution of protected group objects. That is, a group object is encrypted once using a group key and released into the infospace through any available means such as Email, P2P, WWW, USB drives, etc. Although any subject may obtain a copy of the encrypted object, only authorized group subjects may read the object using access machines running trustworthy and verifiable software. This can be briefly summarized as "protect once and access where authorized". This is significantly different from the traditional approach to sharing, characterized as Micro-Distribution (MD) in this paper. In MD, objects are typically custom encrypted individually for each authorized group subject, thereby incurring scalability and performance costs. We discuss the differences between SD and MD architecture for g-SIS in more detail later.

The remainder of this paper is organized as follows. In section 2, we present our SD based architecture and differentiate from MD approach. In section 3, we present TPM-based protocols supporting the architecture. In section 4, we present an implementation model to realize the SD based g-SIS architecture. We conclude in section 5.

2. g-SIS Architecture

An important g-SIS objective that we are interested in is to enable offline access. We strongly believe that some degree of offline access is highly desirable in SIS in which disconnected access attempts are likely. A motivation for offline access in the context of SD in P2P networks in mobile devices can be found in [8]. Offline periods of access can be restricted by various measures such as time, usage, etc. Thus, we expect that sometimes access decisions will be made locally without contacting a central authority. This requires that authorization information such as subject attributes be stored locally and retrieved in a trustworthy manner. Nevertheless, authorization information needs to be periodically refreshed with the central authority.

We follow the well-known architecture-mechanism separation principle as illustrated in [11]. In this section, we assume that the subjects interact with other entities using access machines running a Trusted Reference Monitor (TRM) whose software configuration can be verified. Authorization information such as the group key will be available to the TRM only if the access machine is in a trustworthy state. Thus the TRM can faithfully enforce group policies locally. Later, in sections 3 and 4, we discuss realization of a TRM.

2.1. System Characterization

The g-SIS system consists of subjects and objects, trusted access machines (using which objects are accessed), a Group Administrator (GA) who is responsible for updating subject and object attributes and a Control Center (CC) that maintains the attributes. An access control policy is specified using subject and object attributes. A subject's access machine has a Trusted Reference Monitor (TRM) that maintains a local copy of subject attributes which are refreshed periodically with the CC so as to reflect changes, if any, in their values. There are many approaches to trigger a refresh of subject attributes. For example, a refresh could be triggered based on time-out or a count on the number of times the subject attributes (e.g. group key) may be used by the TRM to decrypt group objects. Offline access to secure clock is not feasible in TPMs today and so we take the usage count based approach for refresh (see for example [9, 5]). Object attributes are embedded in the object itself. Because of super-distribution, there are multiple copies of the same object and it is not practical to reflect attribute update in every copy of the object. For this reason, an object removed from the group is listed in the Object Revocation List (ORL) which is provided to the TRM as part of refresh. A TRM in a g-SIS system maintains the following:

Subject attributes	$\{\mathrm{id}, \mathrm{Join\_TS}, \mathrm{Leave\_TS}, \mathrm{ORL}, \mathrm{gKey}, \mathrm{N}\}$
Object attributes	$\{\mathrm{id}, \mathrm{Add\_TS}\}$
Access Policy	$\mathrm{Authz}(S, O, \mathit{Read}) \rightarrow O \notin \mathrm{ORL}(S) \wedge \mathrm{Leave\_TS}(S) = \mathrm{NULL} \wedge \mathrm{Join\_TS}(S) \leq \mathrm{Add\_TS}(O)$

Subject attribute $\mathrm{Join\_TS}$ is join time of a subject, $\mathrm{Leave\_TS}$ is the leave time of subject (if the subject has left the group), ORL is the Object Revocation List that identifies the list of objects that have been removed from the group along with their history of add and remove timestamps, gKey is the group key using which objects can be decrypted and N is the usage count that limits usage of gKey before a refresh of subject attributes is required with CC. Object attribute $\mathrm{Add\_TS}$ is the time at which an object was added to the group. Attribute id represents a unique identity for each subject and object. Any access policy (based on figure 1) can be specified using these sets of attributes. The policy here specifies that a subject is allowed to read an object as long as both the subject and object are current members of the group and the object was added after the time at which the subject joined the group. This policy is used as an example for our discussion below but in general the CC may instruct the TRM to enforce any such policy.

2.2. System Architecture

Figure 2 shows the SD based g-SIS architecture and illustrates the interaction between various components. The GA controls group membership and policies. The CC is responsible for maintaining authoritative attributes of group subjects and objects on behalf of the GA.

- *Subject Join (steps 1.1-1.4)*: Joining a group involves obtaining authorization from the GA followed by obtaining group credentials from the CC. In step 1.1, the TRM on subject's access machine contacts the GA and

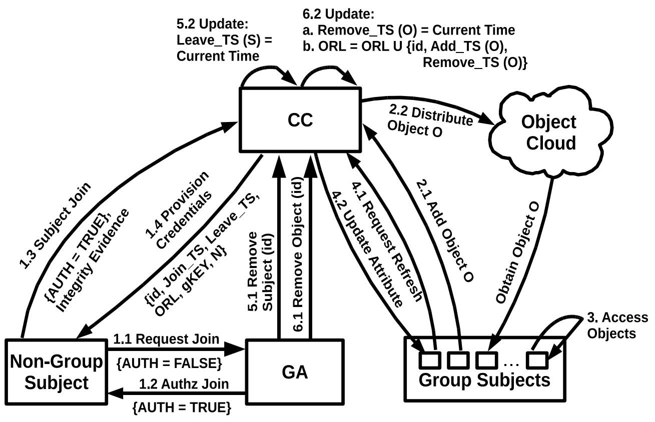


Figure 2. g-SIS Enforcement Model for SD.

requests authorization to join a group. The GA verifies that the subject is not already a member and authorizes the subject in step 1.2 (by setting AUTH to TRUE). The subject furnishes the authorization to join the group and the evidence that the access machine is in a good software state to the CC in step 1.3. The integrity evidence is a signed hash that proves that the chain of software loaded during the boot-up process including the system steady-state is trustworthy. The CC remotely verifies GA's authorization, that the subject's access machine is trustworthy (using the integrity evidence) and has a known TRM that is responsible for enforcing policies. In step 1.4, the CC provisions the attributes after setting them with appropriate values. We refer to these attributes as group credentials or simply credentials. Note that we assume that these credentials are provisioned in a manner such that only the TRM in the subject's machine can access them.

- *Add Object (steps 2.1-2.2)*: From here on, the subject is considered a group member. Objects may be added to the group by subjects after the CC sets the Add_TS (step 2.1). The CC verifies the object[1] and sets the Add_TS attribute. We assume object attributes are embedded in the object itself. The CC then releases the object into the Object Cloud (step 2.2). The Object Cloud represents super-distribution into the infospace.

- *Access Objects (step 3)*: Subjects may access group objects as per the group policy using the credentials obtained from the CC. This is locally mediated and enforced by the TRM. Note that the objects are available via super-distribution and because of the presence of a TRM on subject's access machines, objects may be accessed offline conforming to the policy. Recall that various access policies are possible and we discussed one example in section 2.1.

[1]Verification of object content is an important step for g-SIS. We treat this as an abstract step because verification depends on the specific application. In one case, this could simply be proof-reading while in another case it could be a check for duplicate content or verification that the content is appropriate. We abstract away from such details in the architecture.

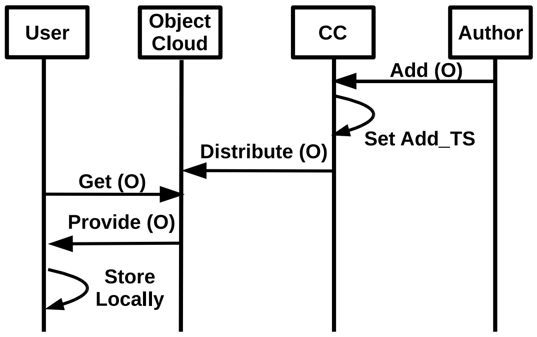


Figure 3. Super-distribution in g-SIS.

- *Attribute Refresh (steps 4.1-4.2)*: The TRM refreshes subject attributes with the CC periodically. More generally, g-SIS access policy, like the one discussed in section 2.1, may be updated or replaced in these steps. A usage count limits the number of times the credentials (e.g., gKey) may be used to access group objects (like consumable rights). Thus objects may be accessed until the usage count is exhausted and the TRM will be required to refresh subject attributes in steps 4.1 and 4.2 before any further access can be granted.

- *Administrative Actions (steps 5.1-5.2 and 6.1-6.2)*: The GA may have to remove a subject or object from the group. In step 5.1, the GA instructs the CC to remove a subject. The CC in turn marks the subject for removal by locally setting the subject's Leave_TS attribute in step 5.2. This attribute update is communicated to the subject's TRM during the refresh steps 4.1 and 4.2. In the case of object remove, due to super-distribution, the remove time-stamp for objects cannot be embedded in the object (since many protected copies of the object can be freely created). Instead, an Object Revocation List or ORL is provisioned for each subject on the access machine. Thus for object removal (steps 6.1-6.2), the ORL is updated with the object's id, Add_TS and Remove_TS.

2.3. Super Vs Micro-distribution g-SIS

In this paper, *Super-Distribution* (SD) refers to widespread distribution of protected group objects while only authorized subjects may read them. *Micro-Distribution* (MD) refers to custom encryption of objects for each authorized subject. Figures 3 and 4 illustrate the

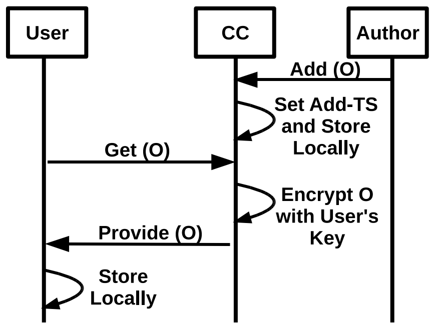


Figure 4. Micro-distribution in g-SIS.

difference between SD and MD in g-SIS. In SD, an Author (a group subject) creates an object, encrypts the object using the group key (mediated by TRM) and sends it to the CC for approval and distribution. The CC verifies the object, time-stamps object add and releases this protected object into the infospace (Object Cloud) through networks such as WWW, Email, P2P, etc. A User (another group subject) can obtain such encrypted objects and store them locally in his/her access machine for later offline access.

In MD, the Author creates an object, encrypts the object using the author's key given by the CC (mediated by TRM) and sends it to the CC for approval and distribution. The CC approves the object, time-stamps object add and saves it locally. When a User requests access to the object, the CC specifically encrypts the object with the key shared with that User. This is the critical difference from SD. Since objects need to be individually encrypted/prepared for each group subject, the scalability of the system is gravely affected. Here scalability refers to performance in the context of number of objects shared amongst group subjects. There may be a large number of objects distributed within the group and the task of custom encrypting each of these objects for each group subject affects the scalability of MD based enforcement model. In SD, the object is encrypted once and all authorized group subjects are able to access them without the intervention of the CC. We now discuss how MD architecture differs from that of SD for g-SIS.

Subject Join: In SD, the CC provisions the group key for each joining subject. In MD, the CC needs to create and share a new key with each joining subject. This requires maintenance of a large number of keys.

Object Add: In MD, the TRM would send the object to the CC. The CC needs to encrypt this object with every other group subject's key so that it is accessible to them. As mentioned earlier, this has scalability and performance implications.

Object Access: In MD, the first time a subject needs to access an object, it needs to be obtained from the CC where it is custom encrypted for that subject. It can thereafter be accessed offline whenever authorized. Note that every group object needs to be initially obtained from CC in MD. In contrast, the objects could be obtained through any means in SD since all objects are encrypted with the group key.

Attribute Refresh, Subject and Object Remove: There is no difference between SD and MD approach for g-SIS.

Assurance and Recourse: In SD, if any one of the group subject's access machine is compromised, the group key can be exposed and all group objects can be read in plaintext. A new group key can be provisioned after recovering from the compromise—although this can only guarantee secrecy of new objects that will be created. Due to this single point of failure problem, the sensitivity of information that be can be distributed using SD model may be limited. In MD, if any one access machine is compromised, only objects that were encrypted for the specific subject using that access machine will be compromised. Other group objects remain safe because they were encrypted with different keys. Note however that the attacker can still compromise confidentiality of all group object using a single machine even in MD architecture. In summary, in SD a compromise can result in large-scale access violation while in MD the violation is fragmented. This is the trade-off between scalability/usability and assurance of each model. A hybrid g-SIS architecture may be employed in large-scale distributed systems in which a big group may be divided into multiple sub-groups. MD can be used for each sub-group and SD may be used within each such sub-group.

3. Protocols

We specify protocols for each step in the architecture in figure 2. We intentionally omit some system level details in these protocols for clarity. For example, we assume that the messages in the presented protocols all carry a MAC (Message Authentication Code) and are protected against replay appropriately using well-known techniques. The focus here is on how the TPM and TRM play a role in enforcing access policies offline by preventing or detecting tamper of group credentials by a malicious user.

Mutual Authentication A mutual authentication protocol (such as Authenticated Diffie-Hellman [2]) is required for any two-party communication. In g-SIS, CC and GA are identified using certificates $\mathrm{Cert}_{\mathrm{CC}}$and $\mathrm{Cert}_{\mathrm{GA}}$respectively, issued by a trusted Certificate Authority (CA). For simplicity, we assume that a subject is tied to an access machine and hence is identified using the id of the TRM. A TRM is identified by an Attestation Identity Key (AIK) certificate. At the end of a mutual authentication protocol, the parties should have authenticated each other and share two session keys $\mathrm{K_s}$ and $\mathrm{K_m}$ used for encryption and MAC respectively.

Notations $X||Y$ refers to item X concatenated with item Y. Key operations are represented using an underscore and multiple items are enclosed within curly braces. Thus $\{X||Y\}\_\mathrm{K_s}$ means that item $X||Y$ is encrypted using $\mathrm{K_s}$. If AK is an asymmetric key, then $\{X\}\_AK$ represents encryption of X using the public part of AK and $\{X\}\_Sign_{AK}$ represents a signature on X using the private part of AK. Finally, $\{X\}\_\mathrm{K_m}, \mathrm{K_s}$ means that item X has been MAC'ed and encrypted appropriately using keys $\mathrm{K_m}$ and $\mathrm{K_s}$ respectively. We use labels to refer to long cryptographic items for convenience. For example, $P : \{X||Y||Z\}$ and a subsequent usage of P such as $\{P\}\_\mathrm{K_s}$ denotes $\{X||Y||Z\}\_\mathrm{K_s}$. A Mutual Authentication (MA) protocol run between entities A and B is denoted $\mathrm{MA(A{:}id_A, B{:}id_B)}$ where $\mathrm{id_A}$ and $\mathrm{id_B}$ are the identities of A and B respectively. For example, $\mathrm{MA(TRM{:}AIK_{TRM}}$, $\mathrm{GA{:}Cert_{GA})}$ denotes a protocol between TRM and GA.

TPM Capabilities It is beyond our current scope to explain the TPM capabilities in detail. A thorough discussion can be found in [1]. A TPM has a Storage Root Key (SRK), the private part of which never leaves the TPM. SRK can be used to encrypt data (keys or arbitrary blob) that can later be decrypted only with the same TPM. A chain of keys can be created (keys in the leaves encrypted with the parent and so on and so forth) where the root key is SRK. A TPM has many Platform Configuration Registers (PCR). A PCR is a register that is capable of holding 160-bit SHA1 hash values. The idea is that as a machine boots up, all the software that are loaded is measured in sequence thus resulting in a final 160-bit SHA1 hash value that reflects the specific boot sequence of software loaded in the main memory of the machine. This value is registered in the PCR and an entity can read this value and understand the trustworthiness of the machine by comparing with a well-known PCR value.

Seal is a protected capability exposed by the TPM. In the simplest case, a seal operation takes a key or a data blob and appends it to a PCR value and encrypts it using the SRK. This secret can later be unsealed only if the current PCR value in the TPM is same as the value mentioned the sealed blob. Thus a seal operation allows an entity to specify the software environment under which a blob may be accessed by any entity in a platform.

CertifyKey is another protected capability where a public part of a key-pair and the PCR value under which the private counterpart may be accessed, is collectively signed using the AIK of the TPM. If (P_{priv}, P_{pub}) is an asymmetric key pair, then a certify key operation, $\{P_{pub}||PCR\}\_Sign_{AIK}$, means that a) P_{priv} is protected using the SRK b) P_{priv} is non-migratable, i.e., it cannot be used in any other platform other than the TPM that created it and c) P_{priv} is sealed to a software state of PCR. Thus any external entity can encrypt a secret using P_{pub} with the assurance that the secret can be decrypted only under a trustworthy platform software state reflected by PCR.

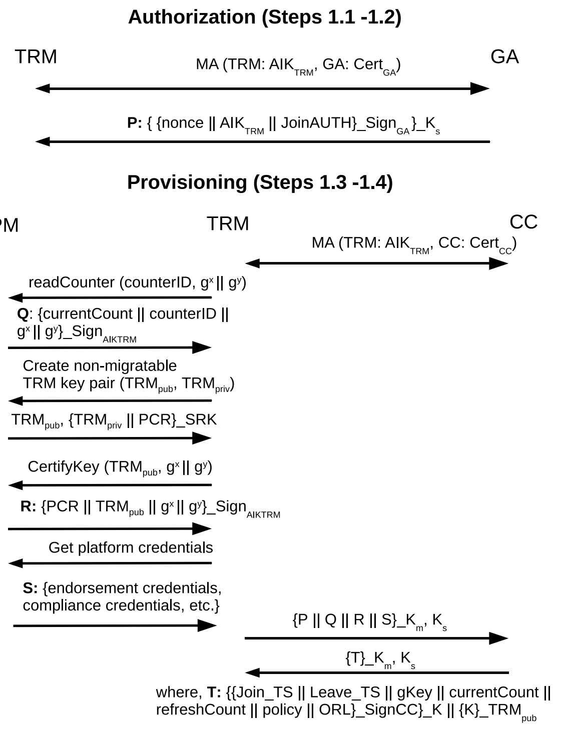

Authorization (Steps 1.1 -1.2)
TRM
GA
MA (TRM: AIK_TRM, GA: Cert_GA)
P: { {nonce || AIK_TRM || JoinAUTH}_Sign_GA }_K_s
Provisioning (Steps 1.3 -1.4)
TPM
TRM
CC
MA (TRM: AIK_TRM, CC: Cert_CC)
readCounter (counterID, g^x || g^y)
Q: {currentCount || counterID || g^x || g^y}_Sign_AIKTRM
Create non-migratable TRM key pair (TRM_pub, TRM_priv)
TRM_pub, {TRM_priv || PCR}_SRK
CertifyKey (TRM_pub, g^x || g^y)
R: {PCR || TRM_pub || g^x || g^y}_Sign_AIKTRM
Get platform credentials
S: {endorsement credentials, compliance credentials, etc.}
{P || Q || R || S}_K_m, K_s
{T}_K_m, K_s
where, T: {{Join_TS || Leave_TS || gKey || currentCount || refreshCount || policy || ORL}_SignCC}_K || {K}_TRM_pub

Figure 5. Join (steps 1.1-1.4 in figure 2).

The TPM also features a monotonic counter, a hardware counter, intended to reflect freshness of any value. For the purpose of this paper, we assume that a running version of the module presented in many approaches in the literature (see for example [9, 5]) is part of the TRM.

Join Protocol Figure 5 shows the protocol for a new subject join (steps 1.1-1.4 in figure 2). In the authorization step, the GA verifies that the subject is not a current member and returns a signature on $\mathrm{AIK_{TRM}}$. "JoinAUTH" is simply a label that communicates the semantics that the subject with the id $\mathrm{AIK_{TRM}}$ is allowed to join the group. In the provisioning step, the TRM contacts the CC to obtain the group credentials. The TRM needs to attest its platform software and hardware state to the CC before the credentials can be provisioned. First, the TRM obtains the current virtual monotonic counter value from its counter module. The nonces used in the mutual authentication can be reused as a nonce for this operation ($g^x||g^y$ represents Diffie-Hellman style exponents used as nonces). Next, the TRM requests the TPM to create a non-migratable key-pair that will be owned by the TRM. The TPM, in response, returns $\mathrm{TRM_{pub}}, \{\mathrm{TRM_{priv}}||PCR\}\_SRK$. As discussed earlier, this message implies that the private part of the created key-pair, $\mathrm{TRM_{priv}}$, is sealed to a software state of PCR. Thus $\mathrm{TRM_{priv}}$ can be unsealed in the future by the TRM only if the software state at unseal time matches the one specified in the PCR at seal time. If the seal-time PCR represents a trustworthy software state, $\mathrm{TRM_{priv}}$ will be accessible to TRM whenever the platform is in the same trustworthy state in the future.

These semantics can be communicated to a challenger (CC in this case) using the TPM's CertifyKey capability. The CertifyKey command takes the $\mathrm{TRM_{pub}}$ key and the private part's associated PCR and a nonce and signs them using the $\mathrm{AIK_{TRM}}$. Again, note that the nonce used here is the same as the ones used for mutual authentication which reflects the freshness of the certified key blob. The TPM will sign $\mathrm{TRM_{pub}}$ using $\mathrm{AIK_{TRM}}$ (which is a key of type AIK) only if $\mathrm{TRM_{pub}}$ is non-migratable. That is, the TRM key-pair can never be accessed using any TPM other than the TPM that created the key-pair. Thus the CC can get the following assurance by looking at the certified key (message labeled as R). The private counter-part of $\mathrm{TRM_{pub}}$ can be accessed only if the software state of the platform is as represented by PCR. If the CC knows the hash-value of a well-known trusted platform state, it can verify this value against the reported PCR and decide to trust the TRM or not. The TRM further gathers the platform credentials (which reflects the trustworthiness of the hardware). Finally, the TRM sends the GA's join authorization (P), current virtual monotonic counter value (Q), the certified TRM key (R) and the platform credentials (S) to the CC. The CC verifies these values and returns the group credentials encrypted with a symmetric key K which in turn is encrypted with $\mathrm{TRM_{pub}}$. As one can see, the group credentials (e.g. gKey) can be accessed only with $\mathrm{TRM_{priv}}$. But $\mathrm{TRM_{priv}}$ is accessible to the TRM only if the platform is in the same software state as specified at seal-time and only in the same platform it was created in. If all is well, the TRM can access the group key and decrypt objects as per group policy.

Object Add and Offline Access Protocol The object to be added is first sent to the CC for approval (figure 6). The CC sets the add time-stamp, signs it and the object is ready for sharing with other subjects.

We assume that the objects that need to be read (step 3 in figure 2) are available locally in the subject's access machine via super-distribution. When a subject requests access to an object (figure 7), the TRM sends a request to unseal the group credentials to the TPM. Recall that the group credentials were sealed to a trusted platform state at join time. Hence the TPM will unseal it only if the current platform state is still trustworthy. The TRM then reads the current counter value and verifies that it is greater than or equal to $\mathrm{currentCount}$ specified in the unsealed group credential. This check prevents a replay of old credentials that could be launched by the subject or other malware. Since the counter will be incremented on every use of the group credential and the counter being monotonic, the $\mathrm{currentCount}$ value in the older group credentials will be less than the value that was read. The TRM also checks that the current counter value is lesser than or equal to the $\mathrm{refreshCount}$ specified in the unsealed credential. This check verifies that the usage count on the group credential has not yet been exhausted. Thus if the policy allows access to the object (see section 2.1), the TRM increments the counter, decrypts the object and allows the subject to read the object. Note that the subject or malware can never make or hijack copies of plaintext object. The TRM allows the subject to read the object only under a protected memory section that it controls. If the usage count is exhausted (i.e., $\mathrm{refreshCount}$ reached), the group credential needs to be refreshed before any further access will be allowed.

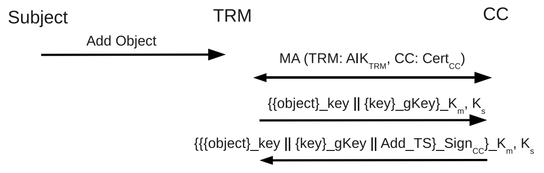

Subject
TRM
CC
Add Object
MA (TRM: AIK$_{TRM}$, CC: Cert$_{CC}$)
{{object}_key || {key}_gKey}_K$_m$, K$_s$
{{{object}_key || {key}_gKey || Add_TS}_Sign$_{CC}$}_K$_m$, K$_s$

Figure 6. Add (steps 2.1-2.2 in figure 2).

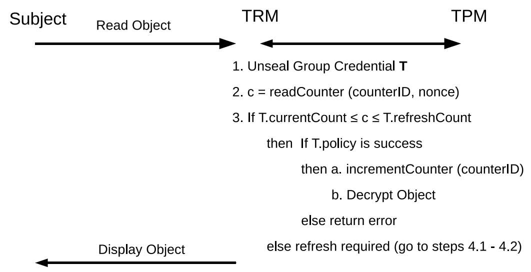

Subject
Read Object
TRM
TPM
1. Unseal Group Credential **T**
2. c = readCounter (counterID, nonce)
3. If T.currentCount ≤ c ≤ T.refreshCount
then If T.policy is success
then a. incrementCounter (counterID)
b. Decrypt Object
else return error
else refresh required (go to steps 4.1 - 4.2)
Display Object

Figure 7. Object Read (step 3 in figure 2).

Refresh Protocol To refresh (figure 8), the TRM sends the current counter value (Q) and a fresh certified TRM key (R) along with refresh request (refreshREQ) to the CC (similar to join protocol). A new credential (T) is created with updated $\mathrm{refreshCount}$. If the subject or object were removed, corresponding attributes are updated in new T.

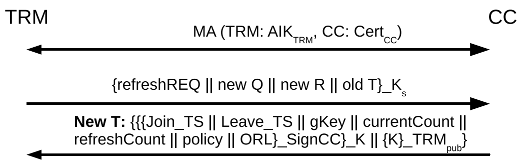

TRM
CC
MA (TRM: AIK_TRM, CC: Cert_CC)
{refreshREQ || new Q || new R || old T}_K_s
New T: {{{Join_TS || Leave_TS || gKey || currentCount || refreshCount || policy || ORL}_SignCC}_K || {K}_TRM_pub}

Figure 8. Refresh (Steps 4.1-4.2 in figure 2).

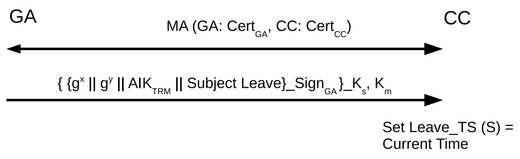

GA
CC
MA (GA: Cert_GA, CC: Cert_CC)
{ {g^x || g^y || AIK_TRM || Subject Leave}_Sign_GA}_K_s, K_m
Set Leave_TS (S) = Current Time

Figure 9. Leave (steps 5.1-5.2 in figure 2).

Leave and Remove Protocols In figure 9, GA instructs the CC to remove a subject ($\mathrm{AIK}_{\mathrm{TRM}}$). The CC sets the subject's Leave_TS to the current time that its maintains. Similarly, the ORL is updated in the case of object remove (figure 10). Note that the object id could be its hash value and the updates are propagated to the TRM during refresh.

4. Implementation Model

We now provide an overview of the g-SIS Implementation Model and a supporting access control framework.

4.1. g-SIS Trusted Execution Environment

The TRM requires a trusted execution environment to run and hence a Trusted Computing Base (TCB) needs to be established on the access machines. Many approaches have been discussed in the literature and figure 11 shows one such approach using the L4 microkernel [6]. L4 is a light-weight, message-based and secure microkernel. L4 ensures that each service running in its context is isolated from each other. A Service Manager (SM) is responsible for managing the L4 services. The TRM and TPM support services run as L4 services each with its own protected address space. The SM owns the TPM device and hence only it can access the TPM primitives. The grayed blocks form the TCB. We assume that the BIOS acts as the Core Root of Trust for Measurement (CRTM). The CRTM is an entity that can be trusted to measure the first piece of software in the boot chain. CRTM initially measures TrustedGRUB [5], an extension of the GRUB bootloader, that can measure the integrity of various files and store it in TPM's PCR during the boot process. After the boot process is complete, the TPM PCR value reflects the integrity of the entire system. This PCR value is reported to remote entities such as the CC in order to attest to the platform's integrity.

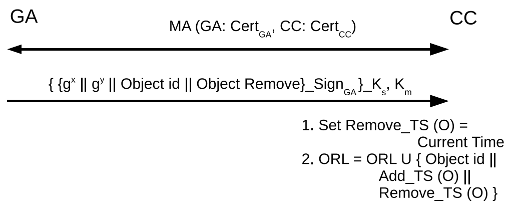

GA
CC
MA (GA: Cert_GA, CC: Cert_CC)
{ {g^x || g^y || Object id || Object Remove}_Sign_GA}_K_s, K_m
1. Set Remove_TS (O) = Current Time
2. ORL = ORL U { Object id || Add_TS (O) || Remove_TS (O) }

Figure 10. Remove (steps 6.1-6.2 in figure 2).

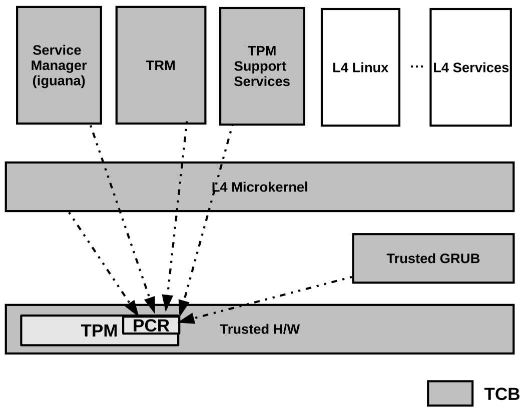


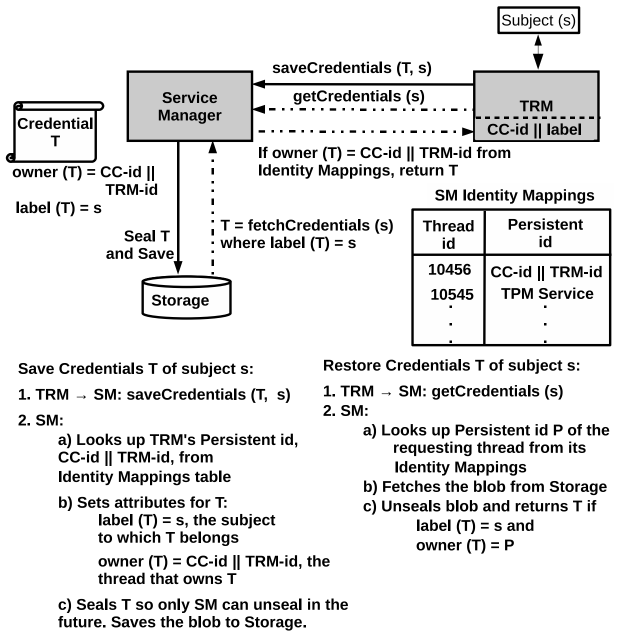
Figure 11. g-SIS implementation model.

4.2. Access Control For Group Credentials

Although the group credentials are sealed, the TPM cannot determine which software can unseal the credentials. It is critical that L4 ensures only the TRM and not any other service gets access to the group credentials like gKey. Our approach is to maintain a local identity infrastructure for L4 threads such as the TRM and ensure that the identity will be preserved across boot cycles. That is, the TRM should be uniquely identified across boot cycles.

Figure 12 shows the access control framework necessary for this purpose. When the TRM is initially provisioned, a persistent local id is stored as an attribute of the TRM's ELF (Executable and Linkable Format) file. At boot time, when SM loads TRM, it reads the persistent id from the ELF and maps it to the TRM's assigned thread id for that session. The SM maintains such mappings for all threads in an Identity Mapping table. This allows SM to identify the TRM even if the thread id changes across boot cycles. Note that in figure 12 the persistent id includes the identity of the CC which issued the credentials.

When SM receives a request to save a subject's credentials T from TRM (solid lines in figure), it uses the Identity Mappings table to look up the TRM's persistent id corresponding to the requesting thread id. Note that the saveCredentials method indicates the identity of the subject (s) to which the credentials belongs—in the event multiple sub-



Save Credentials T of subject s:

1. TRM → SM: saveCredentials (T, s)
2. SM:
   a) Looks up TRM's Persistent id, CC-id || TRM-id, from Identity Mappings table
   b) Sets attributes for T:
      label (T) = s, the subject to which T belongs
      owner (T) = CC-id || TRM-id, the thread that owns T
   c) Seals T so only SM can unseal in the future. Saves the blob to Storage.

Restore Credentials T of subject s:

1. TRM → SM: getCredentials (s)
2. SM:
   a) Looks up Persistent id P of the requesting thread from its Identity Mappings
   b) Fetches the blob from Storage
   c) Unseals blob and returns T if
      label (T) = s and
      owner (T) = P

Figure 12. g-SIS Credentials Access Control.

jects use a single machine. Next, it sets two attributes of T for later retrieval. The 'label' attribute is set to the subject to which the credentials belongs and the 'owner' attribute is set to the TRM's persistent id. SM then seals T so that only it can unseal T in the future and saves the sealed blob.

When SM receives a request to retrieve the credential of s from the TRM (dotted lines in figure), it looks up the requesting thread's persistent id from the Identity Mappings table. Next, it fetches the blob from storage that belongs to the subject s using the 'label' attribute. It then unseals and verifies that the set 'owner' attribute matches the persistent id that it looked up earlier. The TRM can now use the credentials to serve subject requests.

5. Conclusions

We presented architecture, protocols and implementation model for the g-SS problem. We argued that super-distribution is a scalable and usable approach for g-SIS than micro-distribution. In the SD architecture, group subjects were able to encrypt objects once and share with other authorized group subjects. The ability to host a Trusted Reference Monitor on client platforms enabled client-side access control on group objects without always involving an authorization server. We outlined an L4 microkernel based implementation model along with an access control framework for the credentials. A full implementation, investigating architectures to support multiple groups and formal protocol verification are part of our future work.

6. Acknowledgments

Ravi Sandhu is partially supported by NSF grant IIS-0814027, AFOSR MURI award FA9550-08-1-0265 and grants from the State of Texas Emerging Technology Fund and Intel Corporation. Ram Krishnan is partially supported by a grant from Intel Corporation.

References

[1] TCG Specification Architecture Overview. *http://www.trustedcomputinggroup.org*.

[2] W. Diffie, P. Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[3] R. Krishnan, R. Sandhu, J. Niu, and W. Winsborough. A conceptual framework for group-centric secure information sharing. *Proc. of 4th ACM Symposium on Information, Computer and Comm. Security*, 2009.

[4] R. Krishnan, R. Sandhu, and K. Ranganathan. PEI models towards scalable, usable and high-assurance information sharing. *Proc. of 12th ACM Symposium on Access Control Models and Technologies*, 2007.

[5] U. Kühn, M. Selhorst, and C. Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In *Proc. of ACM workshop on Scalable trusted computing*, 2007.

[6] J. Liedtke. Toward real microkernels. *Communications of the ACM*, 39(9):70–77, 1996.

[7] R. Mori and M. Kawahara. Superdistribution: The concept and the architecture. *The Transactions of the IEICE*, 73(7):1133–1146, 1990.

[8] A. Osterhues, A. R. Sadeghi, M. Wolf, C. Stüble, and N. Asokan. Securing Peer-to-peer Distributions for Mobile Devices. In *4th Information Security Practice and Experience Conference*, 2008.

[9] A. Sadeghi, M. Scheibel, C. Stüble, and M. Wolf. Play it once again, Sam-Enforcing stateful licenses on open platforms. In *2nd Workshop on Advances in Trusted Computing*, 2006.

[10] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. *Proc. of the 13th USENIX Security Symposium*, 13:16–16, 2004.

[11] R. Sandhu. Engineering authority and trust in cyberspace: the OM-AM and RBAC way. In *Proc. of 5th ACM workshop on Role-based access control*, 2000.