A Hybrid Enforcement Model for Group-Centric Secure Information Sharing

Ram Krishnan
George Mason University
Email: rkrishna@gmu.edu

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

Abstract—**Group-Centric Secure Information Sharing (g-SIS) is motivated by the need to dynamically share information amongst a set of authorized users for a specific purpose. Authorized group users may read and contribute new objects to the group. An important usability objective in g-SIS is to allow users to access group objects offline without having to contact a server every time an access is requested. Thus a fundamental requirement for g-SIS is that protection needs to extend to clients. Henceforth we assume that a Trusted Reference Monitor (TRM) is present on the client platforms that can enforce the group policies in a trustworthy manner. In this paper, we discuss three different approaches for realizing a scalable and high-assurance g-SIS. In a Micro-Distribution (MD) architecture, objects are individually encrypted for each group user. Thus the server shares a unique key with each user. In a Super-Distribution (SD) architecture, a single key is shared amongst all group users and thus group objects are uniformly encrypted. SD promotes "protect once, access when authorized". We discuss the pros and cons of both MD and SD architecture and propose a novel split-key RSA based hybrid architecture. As we will see, this architecture incorporates the high-assurance aspect from MD and the usability and scalability advantages from SD approach respectively. We finally outline Trusted Platform Module based protocols to realize our hybrid g-SIS architecture. We collectively refer to the architecture and protocols as the Enforcement Model.**

I. 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 [2] and the feasibility to verify the trustworthiness of software running on trusted hardware (e.g., [20]) 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 [12].

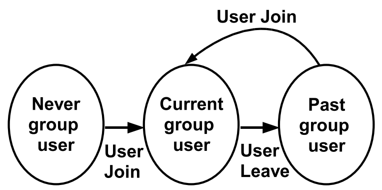

Fig. 1. User Membership States.

The g-SIS problem, introduced in [11], 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, users 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 relative membership state of user and object. For example, a user joining a group may only access objects that are added after join time. In another scenario, the user 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 [10], the authors discuss two distribution approaches, namely Super-Distribution (SD) and Micro-Distribution (MD), for g-SIS. 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 user may obtain a copy of the encrypted object, only authorized group users 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 Micro-Distribution approach for sharing. In MD, objects are typically custom encrypted individually for each authorized group user, thereby incurring scalability and performance costs. In this paper, we discuss the pros and cons of each approach and propose a new hybrid approach. As we will see later, both SD and MD suffers from many practical limitations and our hybrid approach combines the advantages of SD and MD approach and minimizes their disadvantages. We also outline TPM-based protocols for realizing the hybrid approach.

The remainder of this paper is organized as follows. In section II, we review the g-SIS architecture and discuss the SD and MD approach. In section III, we present the hybrid approach and compare the three approaches. In section IV, we present TPM-based protocols supporting the hybrid approach for the g-SIS architecture and in section V, we discuss related work and conclude.

II. g-SIS Architecture Review

In this section, we review the g-SIS architecture that was proposed in [10] and discuss the SD and MD approach for object access in g-SIS. In the next section, we discuss our hybrid approach that addresses the limitations of SD and MD.

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 where disconnected access attempts are likely. 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 user attributes be stored locally and used in a trustworthy manner. Nevertheless, authorization information needs to be periodically refreshed with the central authority. A motivation for offline access in the context of SD in P2P networks in mobile devices can be found in [15].

We follow the well-known architecture-mechanism separation principle as illustrated in [21]. In this section, we assume that the users 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(s) will only be available to the TRM in a trustworthy state. Thus the TRM can faithfully enforce group policies locally. Later, in sections IV we discuss TPM-based protocols for interaction between various entities in the system. It is beyond our current scope to discuss implementation model to realize the TRM. A number of work on building trustworthy and verifiable systems for enforcing mandatory policies using Trusted Computing Technology can be found in the literature (see [20], [19], [13] for example).

A. System Characterization

The g-SIS system consists of users and objects, trusted access machines (using which users access group objects), a Group Administrator (GA) who is responsible for updating user and object attributes and a Control Center (CC) that maintains the attributes. An access control policy is specified using user and object attributes. A user's access machine has a Trusted Reference Monitor (TRM) that maintains a local copy of user 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 user attributes. For example, a refresh could be triggered based on time-out or a count on the number of times the user 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 [19], [13]). Object attributes are embedded in the object itself. 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 g-SIS system can be characterized as follows:

User attributes	$\{\mathrm{id}, \mathrm{Join\_TS}, \mathrm{Leave\_TS}, \mathrm{ORL}, \mathrm{gKey}, \mathrm{N}\}$
Object attributes	$\{\mathrm{id}, \mathrm{Add\_TS}\}$
Access Policy	$\mathrm{Authz}(u, o, read) \rightarrow o \notin \mathrm{ORL}(u) \wedge \mathrm{Leave\_TS}(u) = \mathrm{NULL} \wedge \mathrm{Join\_TS}(u) \leq \mathrm{Add\_TS}(o)$

User attribute $\mathrm{Join\_TS}$ is join time of a user, $\mathrm{Leave\_TS}$ is the leave time of user (if the user has left the group, NULL otherwise), 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 time-stamps, gKey represents the group keys (symmetric or asymmetric key pair) using which objects can be encrypted and decrypted and N is the usage count that limits usage of gKey before a refresh of user 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 user and object. Any access policy (based on figure 1) can be specified using these sets of attributes. Authz here specifies that a user is allowed to read an object as long as both the user and object are current members of the group and the object was added after the time at which the user 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.

B. System Architecture

Figure 2 shows the 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 users and objects on behalf of the GA.

- *User 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 user's access machine contacts the GA and requests authorization to join a group. The GA authorizes the user join in step 1.2 (by setting AUTH to TRUE). The user 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-

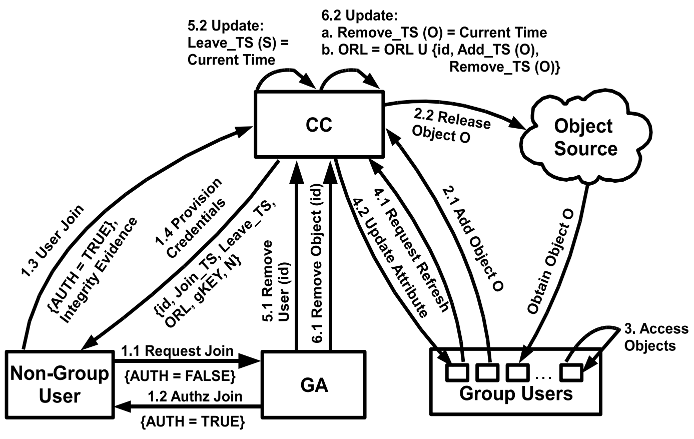

Fig. 2. g-SIS Architecture.

state is trustworthy. The CC remotely verifies GA's authorization, that the user's access machine is trustworthy (using the integrity evidence) and has a known TRM that is responsible for enforcing g-SIS 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 user's machine can access them.

- *Add Object (steps 2.1-2.2)*: From here on, the user is considered a group member. Objects may be added to the group by users 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 protected object to the Object Source (step 2.2). The Object Source acts as the object repository from which users may obtain objects. It is possible for the CC itself to act as the Object Source, but in practice, this may be an independent entity.

- *Access Objects (step 3)*: Users 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 may be obtained from Object Source and stored locally. Because of the presence of a TRM on user's access machines, these objects may be accessed offline conforming to the policy. Recall that various access policies are possible as discussed in section II-A.
- *Attribute Refresh (steps 4.1-4.2)*: The TRM refreshes user attributes with the CC periodically. More generally, g-SIS access policy, like the one discussed in section II-A, 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 user 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 user or object from the group. In step 5.1, the GA instructs the CC to remove a user. The CC in turn marks the user for removal by locally setting the user's Leave_TS attribute in step 5.2. This attribute update is communicated to the user's TRM during the refresh steps 4.1 and 4.2. In the case of object remove, an Object Revocation List or ORL is provisioned by the CC on the user's access machine. Thus for object removal (steps 6.1-6.2), the object's id, Add_TS and Remove_TS are added to the ORL.

[1]We treat this as an abstract step because verification of object depends on the specific application. In one case, this could simply be proof-reading while in another it could be a verification that the content is appropriate.

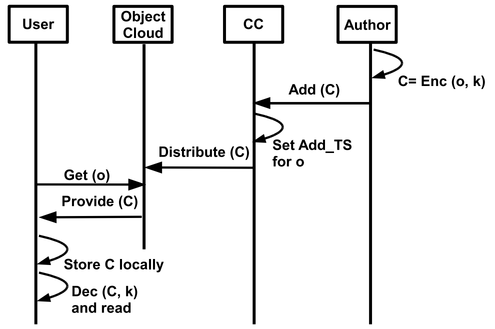

Fig. 3. Super-distribution in g-SIS.

C. Super Vs Micro-Distribution based g-SIS Architecture

We now discuss SD and MD approaches for object distribution and access in the g-SIS architecture. In figure 2, steps 2.1 and 2.2 are object distribution steps and step 3 is the offline object access step. We quickly compare and point out the pros and cons of each approach and present a superior hybrid approach in the following section. In this paper, *Super-Distribution* (SD) refers to widespread distribution of protected group objects while only authorized users may read them. Here all group users share a single group key using which objects are encrypted and decrypted. *Micro-Distribution* (MD) refers to custom encryption of objects for each authorized user. Figures 3 and 4 illustrate the difference between SD and MD in g-SIS. For simplicity, we assume that a symmetric key is used for encrypting and decrypting objects in both SD and MD. Our discussions below apply equally well to SD and MD approaches using asymmetric keys.

In SD based approach all group users share a single group key for encrypting and decrypting group objects. Thus in SD (figure 3), an Author (a group user) 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 (referred to as Object Cloud) through conventional networks such as WWW, Email, etc. or sneakernet such as USB flash drives. These are steps 2.1 and 2.2 in figure 2 for SD approach. Other group users can

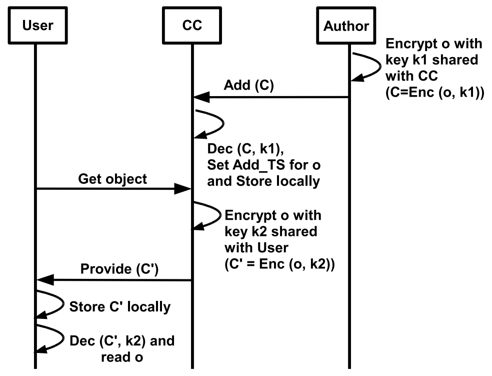
User
CC
Author
Encrypt o with key k1 shared with CC (C=Enc (o, k1))
Add (C)
Dec (C, k1), Set Add_TS for o and Store locally
Get object
Encrypt o with key k2 shared with User (C' = Enc (o, k2))
Provide (C')
Store C' locally
Dec (C', k2) and read o

Fig. 4. Micro-distribution in g-SIS.

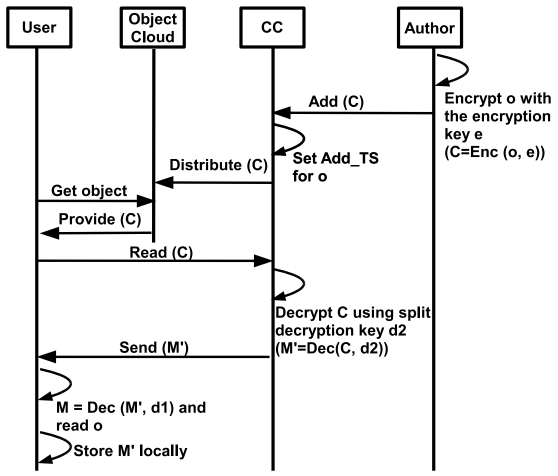
User
Object Cloud
CC
Author
Add (C)
Encrypt o with the encryption key e (C=Enc (o, e))
Distribute (C)
Set Add_TS for o
Get object
Provide (C)
Read (C)
Decrypt C using split decryption key d2 (M'=Dec(C, d2))
Send (M')
M = Dec (M', d1) and read o
Store M' locally

Fig. 5. Hybrid approach in g-SIS.

obtain such encrypted objects and store them locally in his/her access machine for later offline access. Note that group objects need not be obtained from CC or a specific Object Source in SD. Since the group key is shared with all group users, the TRM can decrypt the object and display to the user offline without involving the CC every time.

In MD (figure 4), the CC shares a unique key with each group user. In contrast, in SD the same key is shared with all users in the group. The Author (a group user) creates an object, encrypts the object using the author's key shared with 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. These are steps 2.1 and 2.2 in figure 2 for the MD based approach. When a User requests access to the object, the CC specifically encrypts the object with the key shared with that User. This is a critical difference from SD. Since objects need to be individually encrypted/prepared for each group user, the scalability of the system is gravely affected. Here scalability refers to performance in the context of number of objects shared amongst group users. Since a large number of objects may be shared within the group, the requirement that the CC decrypt and then custom encrypt each object for each group user affects the scalability of MD based enforcement model. In SD, the object is encrypted once and all authorized group users are able to access them without the intervention of the CC. We now discuss how MD differs from that of SD for g-SIS.

- User Join: In SD, the CC provisions the group key for each joining user. In MD, the CC needs to create and share a new key with each joining user. 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 user's key so that it is accessible to them. This suffers from scalability and performance issues.
- Object Access: In MD, the first time a user needs to access an object, it needs to be obtained from the CC where it is custom encrypted for that user. 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, User 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 user'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 user using that access machine will be compromised. Other group objects remain safe because they were encrypted with different keys. In summary, in SD a compromise can result in large-scale access violation while in MD the violation is limited. This is the trade-off between scalability/usability and assurance of each model.

Clearly, the SD based architecture has important scalability and performance benefits of simplified key management (single key per group) and the usability and convenience of offline access and "encrypt once and access where authorized" where group users can quickly share objects with other users. The MD based architecture on the other hand has important failsafe benefits such as limited damage in case of access machine compromise. Note that MD suffers from key management issues regardless of whether symmetric or asymmetric key cryptography is used. For the later, a public-private key pair needs to be shared by the CC with each group user. A more useful and practical architecture should combine the benefits of SD and MD based architecture (thereby minimizing the disadvantages of respective approaches). The hybrid architecture that we discuss in the following section attempts to achieve this by using split-key RSA.

TABLE I
COMPARISON OF SD, MD AND HYBRID APPROACH IN G-SIS ARCHITECTURE.

Aspect	SD	MD	Hybrid
Key type	Symmetric/Asymmetric	Symmetric/Asymmetric	Asymmetric
Number of encryption keys per group	One	One per group user shared with CC	One
Number of decryption keys per group	One	One per group user shared with CC	One split variant for the same RSA exponent per group user
Add object	Encrypt with group key	Encrypt with key shared with CC	Encrypt with private key
Read object	Decrypt with group key and read (encrypt once and access where authorized). Offline access enabled throughout.	First time, CC custom encrypts the requested object for the user (encrypt differently for each user). Subsequent reads can be carried out offline.	First time, CC decrypts the object with its split decryption key. Subsequent reads can be carried out offline (encrypt once and access where authorized).
Usability (with respect to users)	Very high (offline access, no CC participation).	Medium (To add object, need to encrypt with the key shared with the CC. The CC in turn decrypts and custom encrypts for each user.).	High (Encryption is performed with a uniform encryption key).
Performance (with respect to CC)	Very high (CC never participates in encryption/decryption).	Medium (CC participates in decrypting and custom encrypting each object for each group user).	High (CC performs a one time split key decryption operation per document).
Assurance	Low (compromising one user's access machine exposes group key thereby potentially exposing all group objects).	High (Only objects in the compromised access machine are exposed)	High (Only objects in the compromised access machine exposed).

III. HYBRID APPROACH USING SPLIT-KEY RSA

In this section, we present a hybrid approach that addresses the drawbacks of SD and MD based architecture thereby achieving the benefits of each approach.

A. Split-Key RSA

We provide a brief overview of split-key RSA (see [4], [7], [8]) in this section. A proof that the security of split-key RSA is equivalent to that of the classical RSA is given in [22]. In split-key RSA, the decryption key is comprised of multiple parts each held by various parties (or users) involved in the decryption process. Thus if e and d denote encryption (public) and decryption (private) keys respectively, d can be split into n parts—$d1$, $d2$, ..., dn and shared with possibly n different parties. Thus a message encrypted with e can be decrypted only if all n parties participate. Without the loss of generality, let us consider only two splits (and thus two parties) for our discussion below.

$$e * d = 1 \; mod \; \varphi(n) \tag{1}$$

$$d1 * d2 = d \; mod \; \varphi(n) \tag{2}$$

$$C = M^e \; mod \; n \tag{3}$$

$$\begin{aligned} (M)^{d1^{d2}} \; mod \; n &= \\ (M)^{d2^{d1}} \; mod \; n &= \\ (M)^{d1*d2} \; mod \; n &= \\ M^d \; mod \; n & \end{aligned} \tag{4}$$

In the classical RSA [18], the encryption (e) and decryption (d) keys for a given n are related by equation (1). In split-key RSA with two splits, d can be split into two portions as guided by equation (2). A message M is encrypted using a single operation as shown in equation (3). Finally, the fundamental operation of exponentiation in RSA is given by equation (4). Thus decrypting a message M using the split keys $d1$ and $d2$ in any order in equivalent to decrypting the message using d. Also, note that d can be split into two parts in any number of ways, there by yielding pairs of different splits such as $d1$ and $d2$, $d3$ and $d4$, etc. Thus a message encrypted with e can be decrypted using $d1$ and $d2$ or $d3$ and $d4$, etc.

B. Hybrid Approach in g-SIS

Figure 5 illustrates the hybrid approach. Split-RSA keys are created for each group. When a user joins a group, the CC creates a unique split decryption key pair ($d1$ and $d2$ for this user), keeps one split (d2) and shares the other decryption split key (d1) with the joining user. The CC also shares the same encryption key (e) with every joining user. Thus, in figure 5, the Author (a group user) adds an object by encrypting it with e and sending it to the CC. The CC approves the object, sets the add time-stamp and releases it into the object cloud similar to SD. The first time other group users need to access the object, they send a request to the CC. The CC performs the decryption on the object using its split decryption key d2 and sends it back to the user. The user then decrypts this blob using his/her split-key d1 (mediated by TRM) to get the final plain-text object. The blob from CC can be stored locally and future read accesses can be performed completely offline. Note that, in practice, a symmetric object key would be used and the split-key decryption operation will be carried out on the object key instead of the entire object to minimize performance penalty of asymmetric key operation. Table I summarizes and compares g-SIS architectures based on SD, MD and hybrid approaches. Clearly, the hybrid approach for g-SIS architecture combines the advantages of both SD and MD approach and minimizes their disadvantages.

IV. TPM-Based Protocols for Hybrid Approach

In [10], the authors discuss TPm-based g-SIS protocols for the SD based approach. In this paper, we specify protocols for some of the important steps in the architecture in figure 2 for the hybrid approach discussed earlier. The group credential refresh, user leave and object remove protocols are similar to that of the SD based protocols discussed in [10]. Due to space constraints, we do not discuss those protocols here. 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.

a) Mutual Authentication: A mutual authentication protocol (such as Authenticated Diffie-Hellman [5]) is required for any two-party communication. In g-SIS, CC and GA are identified using certificates $\mathrm{Cert_{CC}}$and $\mathrm{Cert_{GA}}$respectively, issued by a trusted Certificate Authority (CA). For simplicity, we assume that a user 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.

b) 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 MA(A:$\mathrm{id_A}$, B:$\mathrm{id_B}$) where $\mathrm{id_A}$ and $\mathrm{id_B}$ are the identities of A and B respectively. For example, MA(TRM:$\mathrm{AIK_{TRM}}$, GA:$\mathrm{Cert_{GA}}$) denotes a protocol between TRM and GA.

c) TPM Capabilities: It is beyond our current scope to explain the TPM capabilities in detail. A thorough discussion can be found in [2]. 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 in the sealed blob. Thus a seal operation allows an entity to specify the software environment in which a blob may be accessed by any entity.

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. 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.

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 approaches such as [19] and [13] is part of the TRM.

d) Join Protocol: Figure 6 shows the protocol for a new user join (steps 1.1-1.4 in figure 2). In the authorization step, the GA verifies that the user is not a current member and returns a signature on $\mathrm{AIK_{TRM}}$. "JoinAUTH" is simply a label that communicates the semantics that the user 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.

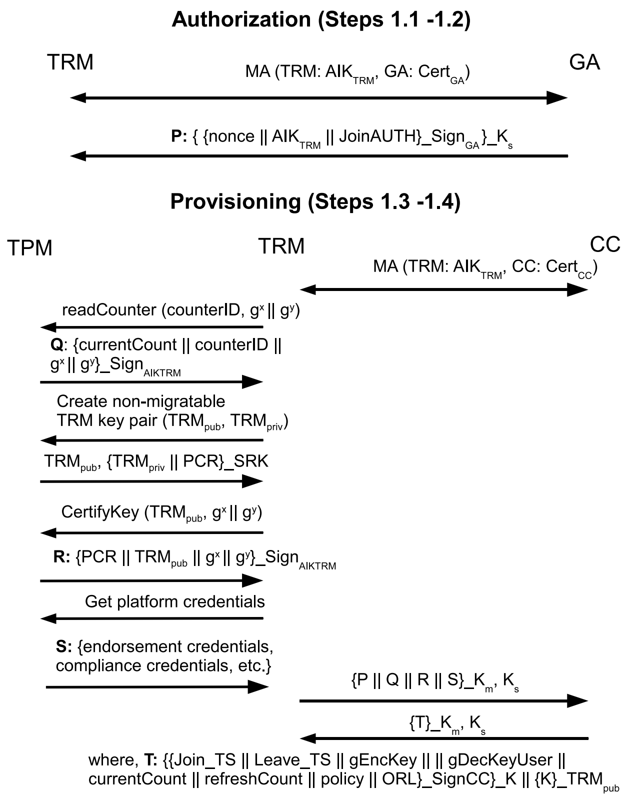
Authorization (Steps 1.1 -1.2)
TRM
GA
MA (TRM: AIK_TRM, GA: Cert_GA)
P: { {nonce || AIK_TRM || JoinAUTH}_Sign_GA }_K_s
Provisioning (Steps 1.3 -1.4)
TPM
TRM
CC
MA (TRM: AIK_TRM, CC: Cert_CC)
readCounter (counterID, g^x || g^y)
Q: {currentCount || counterID || g^x || g^y}_Sign_AIKTRM
Create non-migratable 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 || gEncKey || || gDecKeyUser || currentCount || refreshCount || policy || ORL}_SignCC}_K || {K}_TRM_pub

Fig. 6. Join (steps 1.1-1.4 in figure 2).

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. $\mathrm{gEncKey}$ and $\mathrm{gDecKeyUser}$) can be accessed only by using $\mathrm{TRM_{priv}}$. 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 credentials and access objects as per group policy.

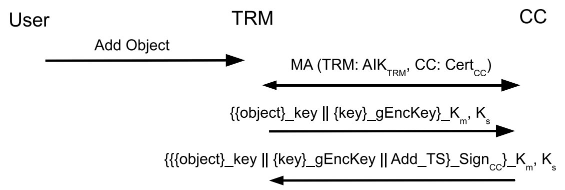
User
TRM
CC
Add Object
MA (TRM: AIK_TRM, CC: Cert_CC)
{{object}_key || {key}_gEncKey}_K_m, K_s
{{{object}_key || {key}_gEncKey || Add_TS}_Sign_CC}_K_m, K_s

Fig. 7. Add (steps 2.1-2.2 in figure 2).

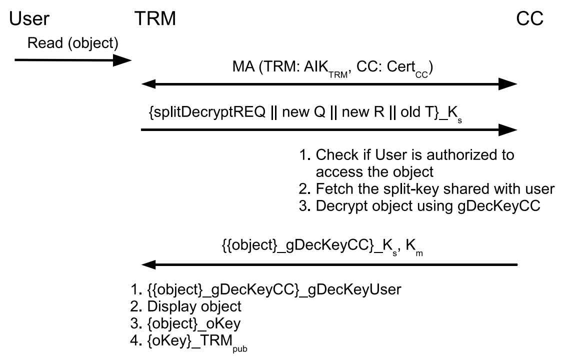
User
TRM
CC
Read (object)
MA (TRM: AIK_TRM, CC: Cert_CC)
{splitDecryptREQ || new Q || new R || old T}_K_s
1. Check if User is authorized to access the object
2. Fetch the split-key shared with user
3. Decrypt object using gDecKeyCC
{{object}_gDecKeyCC}_K_s, K_m
1. {{object}_gDecKeyCC}_gDecKeyUser
2. Display object
3. {object}_oKey
4. {oKey}_TRM_pub

Fig. 8. Object Read (first time).

e) Object Add and Offline Access Protocol: The object to be added is first sent to the CC for approval (figure 7). The CC sets the add time-stamp, signs it and the object is ready for sharing with other users.

We assume that the objects that need to be read (step 3 in figure 2) are available locally in the user's access machine via super-distribution. In the hybrid approach, the first time the user requests to access an object (figure 8), the TRM requests the CC to do a split key decryption operation on the object. The CC checks if the user is authorized to read the object, fetches the split decryption key shared with the requesting user, performs a decryption operation using its split key and returns the blob to the TRM. The TRM can then perform decryption using its portion of the split key and display the object to the user. The TRM may also encrypt the object using a symmetric object key and store locally for future accesses.

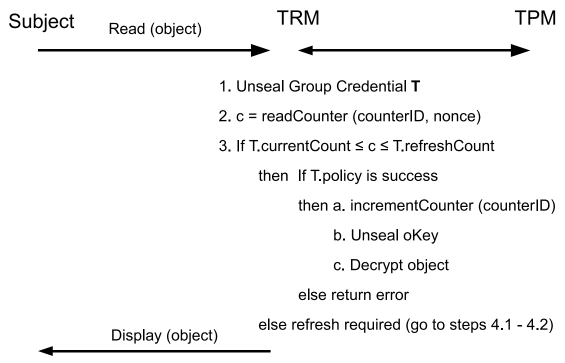
Subject
Read (object)
TRM
TPM
1. Unseal Group Credential T
2. c = readCounter (counterID, nonce)
3. If T.currentCount ≤ c ≤ T.refreshCount
then If T.policy is success
then a. incrementCounter (counterID)
b. Unseal oKey
c. Decrypt object
else return error
else refresh required (go to steps 4.1 - 4.2)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

Fig. 9. Object Read (subsequent accesses).

Subsequently, when a user requests access to an object (figure 9), 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 currentCount specified in the unsealed group credential. This check prevents a replay of old credentials that could be launched by the user or other malware. Since the counter will be incremented on every use of the group credential and the counter being monotonic, the 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 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 II-A), the TRM increments the counter, decrypts the object and allows the user to read the object. Note that the user or malware can never make or hijack copies of plaintext object. The TRM allows the user to read the object only under a protected memory section that it controls. If the usage count is exhausted (i.e., refreshCount reached), the group credential needs to be refreshed before any further access will be allowed.

V. Related Work and Conclusion

We presented a flexible g-SIS architecture that supports SD, MD and a hybrid approach for object distribution and offline access. We also presented TPM-based protocols for interaction between various entities in the g-SIS architecture using the hybrid approach. The g-SIS problem was introduced in [12], [11] and a discussion on SD and MD and TPM based protocols for the SD approach were presented in [10]. However, the SD and MD approach suffers from various practical issues. In this paper, we addressed the limitations of SD and MD by using a novel hybrid approach using split-key RSA and outlined TPM-based protocols for the hybrid approach.

Solution approaches for SD using specialized hardware or purely software-based architectures for Digital Rights Management of entertainment content are plentiful in literature ([1], [17], [23]). Mori et al [14] was the first to propose the concept of SD—although it was applied to widespread distribution of software, the usage of which needs to be authorized and monitored using specialized hardware. In [16], the authors classify security architectures for information dissemination based on three major factors: *virtual machine* that employs control functions on objects to be accessed, *control set* that specify the access rights and usage rules and *distribution style* which specifies whether the objects are pushed to users or the users obtain them from an external repository. In [6], Gallery et al discuss the application of Trusted Computing in the delivery of video content (broadcast services) to mobile receivers which use legacy Conditional Access Systems for protection. An SD infrastructure for secure storage, retrieval and use of copyrighted data in mobile devices can be found in [9]. The solution is based on secure multimedia card extensions for mobile phones. A classification and analysis of current content distribution networks and technologies (called Darknet) can be found in [3]. Our work is different in that it focuses on the g-SIS enforcement model for enabling SD, MD and Hybrid approach in open platforms using Trusted Computing Technology.

References

[1] Architecture of windows media rights management. *http://www.microsoft.com/windows/windowsmedia/drm/default.aspx*.

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

[3] P. Biddle, P. England, M. Peinado, and B. Willman. The Darknet and the future of content distribution. *Proc. of ACM Work. on DRM*, 2002.

[4] C. Boyd. Digital multisignatures. *In Cryptography and Coding*, pages 241–246, Oxford University Press, 1989.

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

[6] E. Gallery, A. Tomlinson, and R. Delicata. Application of trusted computing to secure video broadcasts to mobile receivers. *Tech Report RHUL-MA-2005-8, Dept. of Mathematics, Univ. of London*, 2005.

[7] R. Ganesan. Yaksha: Augmenting Kerberos with public key cryptography. In *Proc. of the Symp. on Network and Dist. Syst. Security*, 1995.

[8] R. Ganesan. *Yaksha: Towards reusable security infrastructures.* PhD Thesis. Johns Hopkins University, 1996.

[9] T. Hatakeyama, H. Maruyama, and T. Chiba. Web computing. Superdistribution and the security of music content. *Fujitsu*, 2001.

[10] R. Krishnan and R. Sandhu. Enforcement Architecture and Implementation Model for Group-Centric Information Sharing. *To appear in Proc. of 1st International Workshop on Security and Comm. Networks*, 2009.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] J. Park, R. Sandhu, and J. Schifalacqua. Security architectures for controlled digital information dissemination. *ACSAC*, 2000.

[17] B. Popescu, B. Crispo, A. Tanenbaum, and F. Kamperman. A DRM security architecture for home networks. *Proceedings of the 4th ACM workshop on Digital rights management*, pages 1–10, 2004.

[18] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[19] 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.

[20] 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.

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

[22] R. Sandhu, M. Bellare, and R. Ganesan. Password-enabled PKI: Virtual smartcards versus virtual soft tokens. In *Proceedings of the 1st Annual PKI Research Workshop*, 2002.

[23] S. Sovio, N. Asokan, and K. Nyberg. Defining authorization domains using virtual devices. *Symposium on Applications and the Internet Workshops*, pages 331–336, 2003.