Journal of High Speed Networks 15 (2006) 229–245
IOS Press

Client-side access control enforcement using trusted computing and PEI models *

Ravi Sandhu [a,**], Xinwen Zhang [b], Kumar Ranganathan [c] and Michael J. Covington [d]
[a] *George Mason University and TriCipher Inc., USA*
E-mail: sandhu@gmu.edu
[b] *George Mason University, Fairfax, VA, USA*
E-mail: xzhang6@gmu.edu
[c] *Intel System Research Center, Bangalore, India*
E-mail: kumar.ranganathan@intel.com
[d] *Intel Corporation, Hillsboro, OR, USA*
E-mail: michael.j.covington@intel.com

Abstract. It has been recognized for some time that software alone does not provide an adequate foundation for building a high-assurance trusted platform. The emergence of industry-standard trusted computing technologies promises a revolution in this respect by providing roots of trust upon which secure applications can be developed. These technologies offer a particularly attractive platform for security policy enforcement in general distributed systems. In this paper we propose a security framework to enforce access control policies with trusted computing, by following the recently proposed policy-enforcement-implementation (PEI) models. Our architecture is based on an abstract layer of trusted hardware which can be constructed with emerging trusted computing technologies. A trusted reference monitor (TRM) is introduced beyond the trusted hardware. By monitoring and verifying the integrity and properties of running applications in a platform using the functions of trusted computing, the TRM can enforce various policies on behalf of object owners. We further extend this platform-based architecture to support general user-based access control policies, cooperating with existing services for user identity and attributes, thus potentially supporting general access control models such as lattice-based, role-based, and usage-based access control policies.

Keywords: Access control, trusted computing, PEI models, security framework, client-side security enforcement

1. Introduction

The concept of "trust" is a complex one spanning technical, social, behavioral, legal and policy issues. Our focus is on a technical concept of trust. We adopt the definition from the Trusted Computing Group (TCG) [3]: that is, "Trust is the expectation that a device will behave in a particular manner for a specific purpose". A "device" can be a platform, or an application or service running on a platform. A platform can be a personal computer, personal digital assistant (PDA), smart phone, etc. In this paper, we consider a client to be a computing platform such that a user (object owner) can distribute some objects directly or indirectly to it, and other users can access the object on the platform. Generally a client can initiate communication with other clients to transfer or share data and resources such as digital documents, voice mail, and electronic currency of various kinds.

In traditional client-server architecture, the server is the focus of trust. A trusted server is typically protected by security mechanisms at multiple layers and across multiple aspects. Sensitive data and resources are protected on the server side. Once these are released to the client there is typically no further control. Even in absence of malice

** Corresponding author. Department of Information and Software Engineering, George Mason University, 4400 University Drive, MSN 4A4, Fairfax, VA 22030, USA. Tel.: (703)993-1668. Fax: (703)993-1638. E-mail: sandhu@gmu.edu.

on part of end users, information resident on the client becomes susceptible to software-based attacks from various forms of Trojan Horses and malware. Malicious software not only can illegally read or modify sensitive data in persistent storage, memory, and input and output buffers, but also change the request for information and actions sent to other computers. It is hardly necessary these days to talk about the prevalence of malicious software. All of us are suffering this pain as we deal with the endless cycle of security patches, new viruses and worms, and spyware. As an example, a recent study by Symantec [7] reports that an average of seven new vulnerabilities a day were announced in 2003, and that newly discovered vulnerabilities are increasingly severe. The attack path has evolved from sniffing and hijacking of authorized connections, to attacks on centralized servers with vulnerable services, and now increasingly to attacks on client platforms [9]. The lack of strong security mechanisms on client platforms in general, and the push for an open environment on computing devices, such as PCs, PDAs, smart phones, notebooks, etc., leave the client extremely vulnerable to software attacks.

It has been generally accepted for some time that software alone cannot provide an adequate foundation for building a high-assurance trusted platform. The quest for finding the "correct" set of hardware primitives for trusted computing was shaped in the mainframe era of the 1970s by the pioneering Multics system [27] followed by a number of research and commercial capability-based computers [1,20]. In the 1980s the US Department of Defense (DoD) pursued a major initiative to develop trusted computers wherein all the trust resided in a small security kernel which specifically controlled information flow based on military-style security labels [15]. The goal of taking trust out of the applications and putting it entirely in the kernel turned out to be fundamentally infeasible for a variety of reasons. Later the DoD attempted to extend trust into applications [16].

The most recent attempts to specify hardware trust primitives began in the late 1990s and continue to be pursued today under the name of trusted computing (TC). The Trusted Computing Group (TCG) defines a set of specifications aiming to provide hardware-based root of trust and a set of primitive functions to propagate trust to application software as well as across platforms. The root of trust in TCG is a hardware component on the motherboard of a platform called the Trusted Platform Module (TPM). TPM provides protected data (cryptographic secrets and arbitrary data) by never releasing a root key outside the TPM. Most important, a TPM provides mechanism of integrity measurement, storage, and reporting of a platform, from which strong protection capabilities and attestations can be achieved. Specifically, a TPM contains a set of Platform Configuration Registers (PCRs), which store the measurements of protected data or program code representing the properties and characteristics of the measured object, such as integrity, running states of a program, and configurations. Once a PCR state is set, it cannot be undone without a reset of the system hardware. An integrity report can be generated by a platform and provided to another platform through a challenge-response protocol called "attestation", by signing one or more PCR values with an attestation identity key (AIK) protected by the TPM, where the AIK is certified by a certificate authority. A significant enhancement of security with TPM is protecting sensitive data (i.e., secrets of an application or a user) with integrity measurement values through "sealed storage". In addition to applying a symmetric key to encrypt the data, one or more PCR values are stored during the encryption along with the protected object. A TPM releases a protected object only if the current PCR values match those stored with the protected object. Therefore, a protected object is available only when the platform is in a particular state.

The main contribution of this paper is to illustrate how TC technologies can support security policy enforcement in distributed environments. Specifically, we propose a framework for distributed access control to enforce an object owner's policy in a client platform by attesting the authenticity of the platform and the integrity and possible properties of the requesting application. We also show how to integrate user attributes such as a user's role into this architecture, as well as how to support a user's ability to roam between platforms by migrating subject identities and attribute certificates. This enables support of user-based security policies, which is not directly supported by TC technologies. Our framework follows the policy-model-implementation (PEI) models framework proposed by Sandhu et al. [29].

The rest of the paper is organized as follows. Section 2 provides the background of this work, including the motivations and the scope of the access control problem that is considered in this paper. Section 3 introduces the PEI models framework, and the individual models for client-side access control are explained in Section 4, 5, and 6, respectively. Section 7 presents some related work and Section 8 concludes this paper.

2. Motivations and challenges

2.1. Motivating applications

2.1.1. Decentralized dissemination control

In decentralized dissemination control (DCON), an object is distributed to a client platform, where the object owner may want to enforce some security policies to control the access, i.e., by trusting the subject that receives and views the object, and the platform and the state of applications. For example, health records of a patient may be transmitted from a primary physician to a consultant who can access them for some limited period of time without being able to retain the records indefinitely or transmit them to anyone else. Similarly, in a company, a manager might distribute a product specification to a team of consultants hired by his department, and the document can only be viewed for two days in the computers within the department. In a decentralized scheme, policy enforcement is performed on the receiving client. On one hand, the policy and secret (to encrypt and decrypt the object during distribution) have to be protected and only available to target platforms and applications. On the other hand, the platform and accessing application have to be trusted not to release the object illegally, either by incorrect configuration or compromised software. Trusted subject authentication may also be needed to enforce that only a valid user can access the object.

2.1.2. Peer-to-Peer Voice-over-IP applications

In recent Peer-to-Peer (P2P) Voice-over-IP (VoIP) applications such as Skype™ [1] [2], audio streams are routed and delivered in the global Internet through active peers, which is similar to that in traditional P2P file sharing systems. Beside the security considerations in routing and network connections, the realtime protection of audio data in a platform is a new issue. Realtime protection ensures that a conversation is not eavesdropped or illegally recorded in transit. Generally an audio stream is encrypted so that intermediate nodes cannot access it. To ensure that in an end platform an audio stream is not illegally accessed by other applications or processes, the initiator of the conversation needs to verify the integrity and state of the platform, including the P2P client application and the audio output channel between the sound card and the application.

Further, in voice mail applications, the owner of a voice object needs to make sure not only the object is not eavesdropped or recorded when playing, but also may need to control whether the receiver can forward the object, and to what kind of receivers (platforms and users) he/she can forward it.

2.2. Security enforcement challenges

The above applications imply the following technical challenges for security policy enforcement at general client side in distributed environments.

- *Change of trust relation*. In traditional security architecture, sensitive objects and policy enforcements are located at server side, and a client generally trusts a server. Once information is released to a client there is no further control. In recent and emerging computing models, a server needs to trust a client, with respect to both platform and user authentication. Note that a "server" here is a general platform that can distribute objects to other platforms.
- *Location of policy enforcement*. A significant requirement of the security architecture for these applications is that security policies are enforced on the client platform. Policy enforcement is dependent on trust between platforms.
- *Trust of platform and application*. In traditional security systems such as mandatory access control (MAC), role-based access control (RBAC), and usage control (UCON), security policies mainly consider the properties of subjects and objects, while the integrity and state of the platform and running software are not considered. It is simply assumed that the operating system and applications responsible for enforcing these

[1] Other names and brands may be claimed as the property of others.

policies are correctly loaded and unmodified, and they are not impacted by other software running in the platform. This may have been a reasonable approach in a time when the operating system and applications were relatively static. In modern open systems certainly the application and possibly the operating system are likely to be much more dynamic requiring mechanisms to guarantee their integrity.

- *Trusted user authentication and authorization in client platform*. Subject authentication is a prerequisite to successfully enforce a security policy. In traditional security systems, authentication mechanisms are provided by a centralized server or service in general. In distributed and decentralized systems, an object or policy owner needs to trust that the valid user is authenticated and authorized in a client platform before being allowed to access a protected object.
- *Trusted path from user to applications and vice versa*. Spoofing and "man-in-the-middle" eavesdropping or modification attacks are amongst the most insidious software attacks. A trusted enforcement environment must enable guaranteed input from the user to application software and, vice versa, guaranteed output from an application to the monitor.

In this paper we claim that the platform integrity and authentication aspects of TC, as well as the provided high-level TC functions such as integrity attestation, separation of execution, sealed storage, trusted channels and paths, etc., can enable a general mechanism to support these new security requirements. Specifically we show how to apply TC technologies to enforce access control policies on client-side platforms, by following the PEI models framework.

3. The PEI models framework

Designers of security systems have traditionally made a distinction between policy and mechanism [18,19]. The general goal has been to build flexible and robust mechanisms that can conveniently support a wide range of policies. Policy is concerned with "what" security needs to be enforced while mechanism is concerned with "how" the security is being enforced. Modern systems are distributed and have multiple trust and service dependencies. Trying to close the policy-mechanism gap in a single step is hardly viable in such a complex environment.

For the purpose of flexibly designing tradeoffs between fuzzy and informal security requirements (goals or objectives) and evolving implementation technologies, recently a 3-layer models framework has been proposed by Sandhu et al. [29] in order to close the overall gap from informal security goals to concrete code in multiple steps. As illustrated in Fig. 1, at the top there is a layer of informal security requirements (or objectives). This layer is necessarily informal. It is the layer at which business issues dominate. The bottom layer includes actual security technologies that are used to implement the objectives. For the purpose of this paper this bottom layer is assumed to be structured around TC technology.

PEI framework introduces three layers of policy, enforcement and implementation models to close the gap in multiple steps. The purpose of a policy model is to take informal high-level objectives and flesh out rigor and detail using a formal or quasi-formal notation. A well-known example of a successful policy model is the lattice model for mandatory access control [13] which captured the informal concept of enforcing information flow using the security labels that had been in use in the military and national security arenas in the paper world. While the policy models focus on the "what" aspect, the enforcement and implementation models address the "how" aspect of enforcing the desired policy. The enforcement models address the big picture of the "how" question, at the level of system block diagrams and protocol flows. The implementation models are focussed on specific issues identified in the enforcement models layer. These issues need to be resolved in sufficient detail to require descriptions at a pseudo-code level of detail and precision.

Note that this framework is absolutely not intended to be a top-down waterfall-style software engineering methodology. The main purpose of the framework is to enable work to proceed concurrently at all layers while keeping clear as to which decisions are being made at which layer. The relationship between models at adjacent layers is many-to-many. A single policy model can be enforced by multiple enforcement models (with possibly different trade-offs between security, trust, performance, cost, convenience, etc.). Likewise a single enforcement

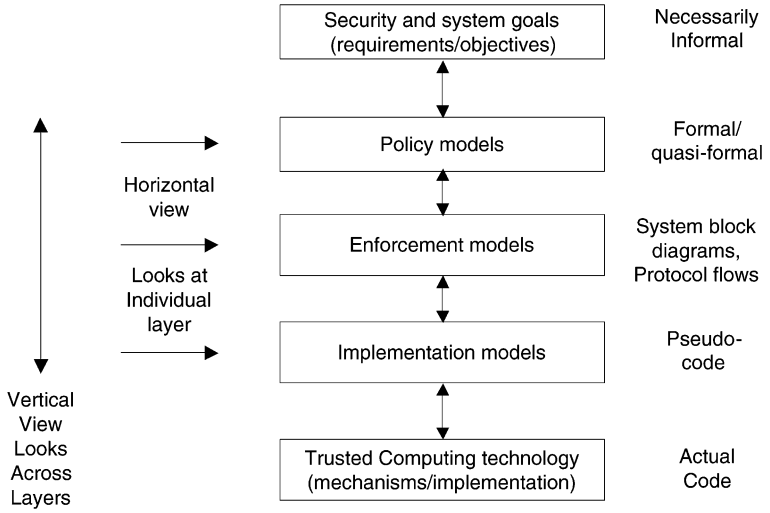

Fig. 1. The PEI models framework.

model may support multiple policy models. There is similarly a many-to-many relationship between enforcement models and implementation models. This many-to-many relationship between adjacent layers further indicates the futility of a top-down approach in this context.

In the rest of this paper we apply this PEI framework to the access control problem in distributed environments, as scoped out earlier, in the context of TC. We begin with a horizontal view at the policy models layer in Section 4, then we propose enforcement and implementation models in Section 5 and 6, respectively.

4. Policy model for client-side access control

The high-level view of distributed access control is to ensure the confidentiality and integrity of a protected object after sharing or dissemination. Within the protection scope considered in this paper we assume that an object is always encrypted for super-distribution, i.e., freely disseminated by public channels. An object is located and accessed in a general client platform, either pushed by the object owner or pulled by a requesting user (subject). Each object is logically associated with a protection policy, which is either physically embedded in the object, or its location (e.g., URL) is indicated by the object. The integrity of a protected object implies the integrity of its policy.

Based on the scalability and flexibility of policy specification, we consider three policy models: discretionary access control (DAC), role-based access control (RBAC), and usage-based access control (UCON), from simple and small-scale to complex and large-scale distributed applications. In this paper we assume that each object logically has a policy, which is defined by the object owner. Therefore, we do not consider mandatory access control model (MAC) in our framework.

- In DAC, an object owner specifies policies by restricting a subject's access to the object, where a subject is generally specified by its identity. Discretionary model can be used for small scale and person-to-person control, where identities of subjects and trust of platforms can be easily verified. For example, in a working group a manager distributes a product specification to all the members of the group and it only can be viewed on the platforms of the group.
- With RBAC model, policies can be defined for person-to-role or role-to-role approaches, where a role is a collection of permissions (to objects) and assigned to subjects. By leveraging the widely supported RBAC model [28], role-based policies can be easily implemented and enforced in existing system architectures.

- In UCON, authorization decisions can be determined by general subject and object attributes instead of pre-assigned permissions or roles. Besides authorizations, an access in UCON may also depend on subject obligations and system conditions. Also, an access can have side-effects of updating attribute values, which makes the model very expressive to support many dynamic policies. The UCON$_{ABC}$ model [22,30] comprehensively considers these aspects and fundamentally extends traditional access control models with properties of decision continuity and attribute mutability. Therefore usage-based access control not only provides comprehensive considerations of access control for an object owner, but also supports enriched constraints and usage process controls.

5. Enforcement model for client-side access control

5.1. Basic assumptions

The following are assumed in this paper.

- We do not include the access control policies of hardware administration, such as the permissions for a TPM owner or a general user [4].
- We assume that the hardware layer of TC is sufficiently tamper-resistent; that is, hardware attack is not a threat in our model.
- We assume a homogeneous environment in this paper. By "homogeneous" we mean that each platform is equipped uniformly with necessary TC hardware.
- For simplicity we do not explicitly consider the privacy problem in our approach, except that existing considerations in TC technologies are adopted, i.e., a platform AIK can be certified by a trusted privacy CA [12] or through direct anonymous attestation [14] between platforms.
- We only consider read-only and content-independent authorizations; that is, the access control mechanism can decide whether or not a specific user has access without rendering the protected object content (in encrypted form) in clear, and the user cannot modify the object.

5.2. A platform with trusted reference monitor

An abstract platform is shown in Fig. 2. The trusted components consist of trusted hardware including TPM, a secure kernel (SK), and a trusted reference monitor (TRM) in user space of the operating system. The hardware, cooperating with the kernel, provides necessary functions to the TRM, from basic cryptographic functions to platform and program attestation and sealed storage for sensitive data. The trustworthiness of the SK is provided by the hardware-based root of trust.

A prerequisite for application security is that the integrity of a process should be enforced not only when it is launched, but also during its runtime. This has two aspects. First, the memory space of an application is private, which is not accessible to other applications even for devices with direct memory access (DMA) capability. This implies that an application which can access an object cannot release it to other applications without explicitly opening a secure channel. Secondly, communications between applications through secure channels are protected and only available to the corresponding applications. TPM and TC-enhanced hardware technologies such as Intel's LaGrande Technology™ (LT) and AMD's Secure Execution Mode™ (SEM) [6] generally allocate isolated memory partitions to different applications to prevent software-based attacks from malicious applications, device drivers, worms, viruses, Trojan Horses, and even the underlying OS.

Another security requirement of a trusted platform is a trusted path between an application and graphics and I/O channels. For example, an application is authorized by the TRM to display a document to a user. The channel from the application to the graphic adaptor and driver must be protected such that no other process can intercept and sniff, or illegally modify the content. Also, inputs from keyboard and mouse must be protected. Again, the existing TC technologies support these functions.

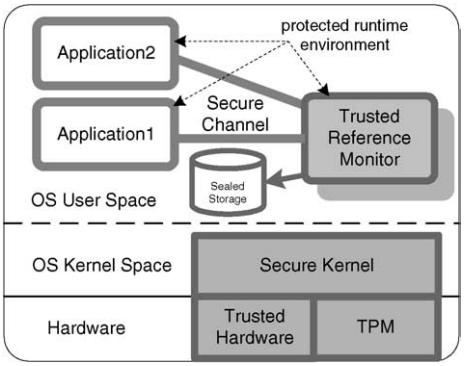

Fig. 2. An abstract platform architecture.

In this paper we abstract the underlying hardware and kernel structure by simply assuming that these requirements and necessary TC functions can be achieved by using existing technologies in a platform. For example, in a platform equipped with LT, a TPM v1.2 is applied in the hardware layer with extended CPU and chipset. A domain manager layer between the kernel and hardware supports separation of application domains. The states (integrity measurements) of the platform, including the booting system and SK are stored in specific PCRs in the TPM.

5.2.1. Trust model

After booting the platform, the integrity of the booting system and SK are measured by the TPM in a platform and stored in particular PCRs. SK has a public-private key pair where the public key is certified by an AIK of the TPM and the private key is protected by the TPM. Before starting an application, the SK measures the integrity of the application code and stores the hash value locally. SK also generates a public-private key pair for the TRM, where the public key is certified by SK (by signing with its private key) and the private key is protected by the TPM. This key pair is generated at the first time when the TRM is loaded in the platform. For attestation, the TPM signs the values in the PCRs with its AIK key, and SK signs the integrity value of the TRM with its private key. Both are sent to a remote challenger. The challenger verifies both signatures and the public key certificates of the TPM AIK and the SK. If all are valid and integrity values are as expected, the secure kernel and TRM are trusted.

TRM can seal sensitive data, including secrets and policies. A secret can be an encryption key for an object, which is originally from the object owner and distributed to the platform. A policy is also generated by an object owner and controls the access to this object in this platform. A typical policy specifies the integrity state of an application on a genuine platform where the object can be accessed. Security attributes of a subject may also be specified in a policy, such as security clearance or role name. Secrets and policies are sealed by a TRM such that they are only available to the TRM in a valid integrity state. This guarantees that the policies are correctly enforced by the TRM, and the secrets are not released outside of the TRM. The policy enforcement is performed by attestations explained later in this section.

When an application (i.e., a subject) is invoked to access an object, the TRM in the platform attests it and compares the application's integrity value with that defined in the object's policy. The object can be accessed only if the application is running in a trusted state. During the access (e.g., the object is being viewed by the application), all the I/O channels of this application must be protected with the functions of the underlying TC hardware.

5.2.2. Credentials

We presume the following set of credentials and the corresponding certificate authorities (CAs), if necessary, are available.

- TPM storage key(s) to protect SK and TRM's credentials and other sensitive data, such as secrets and policies. This key must be either the storage root key (SRK) of a TPM, or a key protected by the SRK.
- TPM AIK pair (PbK_{AIK}, PrK_{AIK}). An AIK is created by a TPM and used to sign PCR values and present to a challenger in an attestation protocol, or to sign a public key of an application running in the platform

for authenticity. Generally the private part of an AIK is protected by a TPM with the SRK and public key certificate is issued by a trusted third party such as a privacy CA.

- SK's asymmetric key pair (PbK_{SK}, PrK_{SK}). This key pair is generated by TPM when the platform is initialized. The public key is certified by the AIK of the TPM, and the private key is protected by the TPM.
- TRM's asymmetric key pair (PbK_{TRM}, PrK_{TRM}). Each TRM has this key pair for signature and encryption. The private key is protected by SK (e.g., sealed by SK), which is indirectly protected by the TPM in the platform such that only the TRM on this platform can use it (by checking the integrity value). The public key is certified by SK.
- Application asymmetric key pair (PbK_{APP}, PrK_{APP}). Similar to TRM, each application has an asymmetric key pair. The private key is protected by SK, and the public key is certified by SK.

5.2.3. Primitive functions of a TRM

By leveraging the capabilities of TPM and SK, a TRM has the following primitive functions.

- *TRM.Seal*($\mathcal{H}$(*TRM*), x). This function seals data x by a TRM which has integrity measurement of $\mathcal{H}$(*TRM*). The x can only be unsealed under this TRM when the corresponding PCR value is $\mathcal{H}$(*TRM*). The actual key used in the sealed storage is a TPM storage key. The integrity measurement of TRM is performed by SK when TRM is loaded and assigned to an isolated runtime environment.
- *TRM.UnSeal*($\mathcal{H}$(*TRM*), x). This function unseals x provided $\mathcal{H}$(*TRM*) is the value that was used to seal x.
- *TRM.GenerateKey*(k). This function generates a secret key k.
- *TRM.Attest*($\mathcal{H}$(*TRM*), PbK_{TRM}) = ($\{\mathcal{H}(TRM)\}_{PrK_{SK}}$, $\{PbK_{TRM}\}_{PrK_{SK}}$, $\{PbK_{SK}\}_{PrK_{AIK}}$). This function generates an attestation response by returning a certificate of the TRM's public key and its integrity value, both signed with SK's private key, along with the public key certificate of the SK.

5.3. Enforcement architecture

5.3.1. Policy and secret distribution

The first step to control access to an object in a client platform is the generation and distribution of the policy and the object encryption secret. Figure 3 shows a basic use case for our architecture wherein a server[2] (Alice's platform) attests a client (Bob's platform) and distributes policies and secrets to the client. A high-level protocol flow is explained as follows.[3]

1. The TRM of Bob's platform sends an access request message to the TRM of Alice's platform if it has not requested this object before. Originally this request may be from an application (*APP*) in Bob's platform, i.e., Bob invokes *APP* to access *OBJ*. The integrity measurement of *APP* signed by the SK in Bob's platform may be included in this message, which can be available to the TRM by an attestation challenge in the same platform. Also, an object identity (*OBJ_ID*) is included in this message.
2. The TRM of Alice's platform verifies the integrity of the requesting application. *APP* is trusted to preserve some properties, (e.g., *APP* will not save an object in plaintext to any persistent storage) that are required to enforce security policies. Alice's TRM sends attestation challenge message to Bob's TRM.
3. Bob's TRM calls *TRM.Attest*() function, which returns a certificate of Bob's TRM running hash and its public key, signed by the private key of his platform's SK, as well as the SK's public key certified by an AIK of the platform, and sends back to Alice.
4. Alice verifies the attestation. If Alice trusts the Bob's platform (that the platform has a genuine TPM and trusted booting), the SK running in the platform, and the integrity of TRM, Alice's TRM generates a secret key k_{OBJ}, then encrypts this key along with the policy information using the public key of Bob's TRM, and sends to Bob's TRM.

[2]Note that for comparison we use the term "server" here to denote the source platform of objects and policies. A specific platform can be a server platform or client platform at any time depending on the role it is playing in a given distributed interaction.

[3]This description is not intended to be a detailed cryptographic protocol, but rather the high-level architecture of what such a protocol needs to convey. A detailed protocol would require careful security consideration with respect to well-known attacks such as replay etc., and could be engineered around the high-level architecture using established cryptographic design principles.

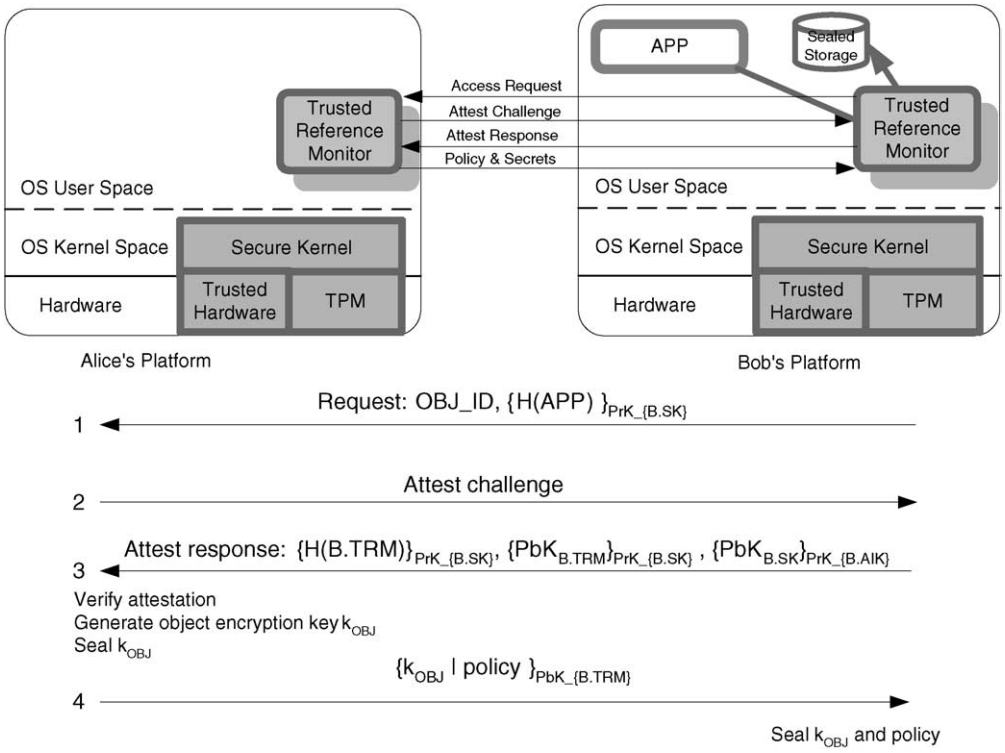

Fig. 3. Policy and secret distribution for client-side security enforcement.

Through the attestation challenge, Alice trusts that the TRM in Bob's platform can enforce the policy that Alice sent, which specifies the conditions whereby the object can be accessed in Bob's platform. For confidentiality of the policy and secret, the TRM in Bob's platform seals these items with its own integrity measurement value. Secrets and policy information never leave the TRM's application domain, which is isolated from other application domains and communications between them are protected as we show shortly. Therefore, the protection capability of a client's TRM can be considered as an extended function of the server's platform.

An assumption behind this architecture is that by verifying an application's integrity measurement, an entity knows particular expected properties are preserved by using this application. In an organization or company environment, this is applicable when all platforms and applications are administrated by the IT department. For global and open environments, we may rely on some trusted third party. For example, an independent CA certifies that software with a particular integrity value and patch version satisfies a particular property, and this CA is trusted by an attestation challenger (e.g., Alice's platform).

Secrets generated by a server platform and sent to a client platform are used to encrypt protected objects before distribution. For example, Alice encrypts the target *OBJ* with k_{OBJ} and distributes it to Bob's platform. An object can be distributed together with a policy, or separately. Since it is encrypted and the key is only available to a particular TRM, the security of the object is preserved during distribution.

An important property of secrets and policy is migratability. A *migratable* object can be re-distributed from a platform to another platform, e.g., Bob can migrate the key and policy from his office PC to home PC to view the object at home; while a non-migratable object cannot be re-distributed. Another option is whether after a re-distribution the key is retained on the original platform or deleted. If the key is deleted after migration, the object can only be accessible in single platform at any time. Both options should be determined by applications and specified in policies. For example, in a DCON application, a policy could specify that Bob can only read a product development document on his office desktop, while another policy could require that Bob can view a

product manual both on his office PC and home PC. Fine-grained policies can be defined to specify more complex situations, such as Bob can re-distribute an object to Charlie, who cannot re-distribute it to others; or Bob can read an object on a fixed set of platforms.

5.3.2. Policy enforcement

After secret and policy distribution, a subject on the client platform can generate an access request by invoking an application or process. The TRM in the platform checks the application's integrity state based on the policy information and makes an authorization decision. Figure 4 shows the enforcement of a policy when an access to OBJ is generated from application APP in Bob's platform. A high-level description is given below.

1. *APP* sends "view *OBJ*" request to the TRM, with the object encrypted by the secret k_{OBJ} distributed from the object owner.
2. The TRM sends attestation challenge message to *APP*.
3. *APP* responds with its running integrity value measured and signed by SK, along with its public key certified by SK.
4. The TRM compares the integrity measurement with a list of expected values according to the policy. If *APP* is trusted, the TRM generates a session key k_s and encrypts it with the public key of *APP*, then sends to *APP*. At the same time, the TRM unseals the k_{OBJ}, decrypts *OBJ* with k_{OBJ}, encrypts *OBJ* with k_s, and sends this back to *APP*. The TRM updates the policy if necessary, e.g., to update a usage count.

Mechanisms are needed to prevent replay attacks from malicious applications, e.g., including a nonce value in the attestation challenge and response messages. For simplicity we ignore this aspect in this paper. As we have mentioned, the applications (including TRM) in a trusted platform are running in isolated domains and memory spaces. There are many mechanisms to implement communication between two application, such as inter-process

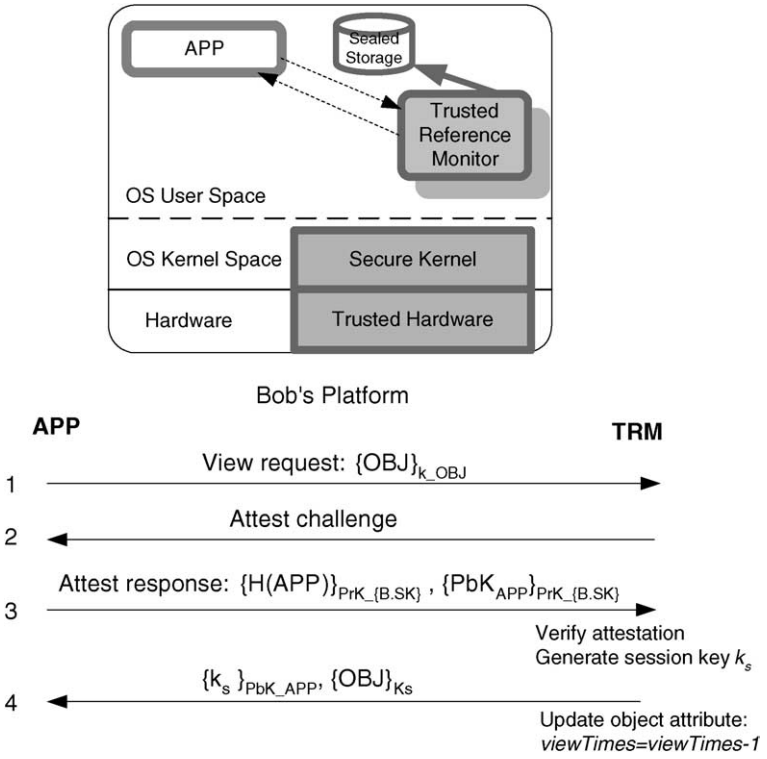

Fig. 4. Policy enforcement in a client platform.

communication (IPC). At the same time, a general-purpose TRM manages resources (secrets and policies) and supports multiple applications in a platform, and also accepts requests from other platforms. A TRM is responsible for ensuring that the resources that it protects do not leak into other applications. Also, a TRM can be dedicated to a particular application.

6. Implementation models for client-side access control

This section presents several aspects of implementation models for client-side access control. Note that a complete investigation of implementation models is beyond the scope of this paper. Particularly, we consider the policy specification and revocation, and supporting user attributes in access control policy enforcement.

6.1. Policy specification and revocation

A policy is created by an object owner (i.e., Alice in Fig. 3), distributed to a client platform, and enforced by the TRM in that platform. Generally a policy is sealed by a TRM. Logically we can assume that for each object there is a policy bundled with it.

6.1.1. Policy specification

Formally, a policy states the conditions under which an object can be accessed by a subject. A condition is one or more sets of platform configurations. A platform configuration consists of a trusted platform attestation identity name (represented by an AIK certificate) and an application's integrity measurement with expected properties. A policy may explicitly specify that a specific property of the accessing application must be satisfied. For example, for confidentiality an application cannot save an object in plaintext to any persistent storage, or cannot open any network socket. As we have mentioned, this can be certified by a trusted third party.

Figure 5 shows an XML specification of a policy. The "migratable" attribute with Boolean value of the <policy> entry states whether or not this policy can be migrated to another platform. A <subject> entry specifies attributes required for the accessing subject, such as identity certificate and role name. Multiple <subject> entries may appear

```
>?xml version="1.0" encoding="UTF-8"?> <policyns
xmlns="http://www.example.com/tc-policy"
 owner="example.com" filename="mypolicy">
<policy migratable="false">
  <object id type="object_type">object_ID</object_id>
  <subject type="cert"> Bob </subject>
    <subject type="role"> employee </subject>
  <right name="view">
    <param name="viewTimes" value=10/>
  </right>
  <condition type="TP">
    <platform\_id> Bob\_office </platform\_id>
    <environment>
      <title> APP</title>
      <version> 1.0</version>
      <DigestValue alg="shal"> 487A3D... </DigestValue>
      <X509Certificate> MAW09GF... <X509Sertificate>
    </environment>
  </condition>
</policy>
```

Fig. 5. Policy specification example.

in a policy, all of which have to be satisfied in an access. A <right> entry can have some parameters, such as, an object can be viewed 10 times (*viewTimes* attribute with value 10). A <condition> with type "TP" specifies a platform environment, including a running application's state, such as version, integrity hash, conformance certificate, etc. A conformance certificate is issued by the vendor of the application software and certifies a set of properties of the software.

More complex policies can be specified according to an object owner's requirements. For example, if a policy is migratable, then the possible platforms that a policy may be migrated to can be specified in the policy. Standard XML specification for certificates and digital signatures can be used in general policy specifications.

6.1.2. Policy revocation

A policy revocation can happen because of any of the following reasons.

- Trust revocation of a requesting applications, i.e., the integrity measurement of the application is not trusted by an object owner any more.
- Trust revocation of a TRM, i.e., a TRM software is updated or new patch is available, and the integrity value of the old version may be revoked.
- Trust revocation of a platform, i.e., a platform attestation identity is not trusted by an object owner any more, either because of the trust revocation of the platform itself, or because of the trust revocation of a privacy CA.

There are two approaches for policy revocation in our architecture: *push* and *pull*, based on the mechanism of policy update. With "push" approach, when any revocation happens, an object owner (Alice) pushes an updated policy message to the TRM of a client platform (Bob) where the object is located. A log of object distribution may be needed in a platform to record the location of an object. For "pull" approach, a TRM checks the source platform of the accessing object for policy updates, which can be performed periodically. Since there is no guarantee that both platforms are available at the same time after secret and policy distribution, both mechanisms may have some delay in updating policy. An instant policy revocation may require either both platforms are online, or there is an online policy service component such that each time a TRM tries to authorize an access, it checks the policy in the online component. Note that trust revocation is a different concept from the change of an integrity value, i.e., an application is illegally modified by a virus or malicious software such that its measurement value does not match that in a policy, and its access is denied.

6.2. Support user attributes-based policies

In the architecture described in Section 5.3, access control is based on the properties of platforms and applications only, which are directly supported by trusted computing technologies. In practice an object owner may want to control not only the platform and application, but also by which user an object can be accessed. In general a user is associated with one or more security attributes, such as a role name or a clearance level. As specified in Fig. 5, a subject type is an attribute name, such as "role", and the value is a role name. In this section we extend the basic architecture to support user-based access control policies, as a complement to platform-based policies. Specifically, we implement role-based access control [28] in our architecture with the mechanism of credential migration provided by TPM.

6.2.1. Identity credential of a user

To import user identity into a platform, we assume that in each platform there is a User Agent (UA). A UA can be an independent service, or a component of a service that manages user authentications and identities. Generally a UA is controlled and owned by a platform administrator. Note that a platform administrator may or may not be a TPM owner. Also, a platform user may or may not be a TPM user[4] [8].

Like TRM, the UA in a platform has a public-private key pair (PbK_{UA}, PrK_{UA}), where the private key is protected by SK and the public key is certified by SK. We further assume that each user has at least one identity key pair

[4] A TPM user is an entity that can load or use TPM objects such as keys, which is not necessarily a human, but also could be an application or service. A TPM user can be created either by the TPM owner or by other TPM users.

(PbK_u, PrK_u), which is a migratable object protected by a TPM, and wrapped with the UA's credential in the protected object hierarchy of the TPM. A migratable key can be created by the local TPM (or some other platforms) and can be securely migrated from one platform to another platform with the authorization of the key owner. For identity key, the key owner is the user.

Various authentication mechanisms may be provided by a platform, such as password, smart card, or biometric technology. Other online authentication mechanisms can also be used. Only a correctly authenticated user can be trusted to access his/her credentials and related services on a platform. For example, industry proposed Trusted Mobile Platform (TMP) specification [11] requires authentication between a user and a trusted mobile device. Authentication mechanisms can be integrated into the UA in a platform, or the UA can obtain authentication information from other components through secure protocols.

6.2.2. Identity and role certificates

Just like the privacy CA for certificate of a platform AIK, a user can be certified by a trusted third party (an identity CA). At the same time, a user is assigned a role, which is certified by a role server. A role certificate is based on a user's identity and contains the user's role name. There are several mechanisms to obtain identity and role certificates for a user. Figure 6 shows an approach based on using the UA in a platform, utilizing TC functions. The general process is described as follows.

1. The UA sends the public key of a user (PbK_u), its own public key (PbK_{UA}), and its integrity measurement ($\mathcal{H}(UA)$) to the identity CA. The PbK_u is signed by the UA, and both the UA's public key and integrity value are signed by the SK, which in turn, is signed by an AIK of the TPM.

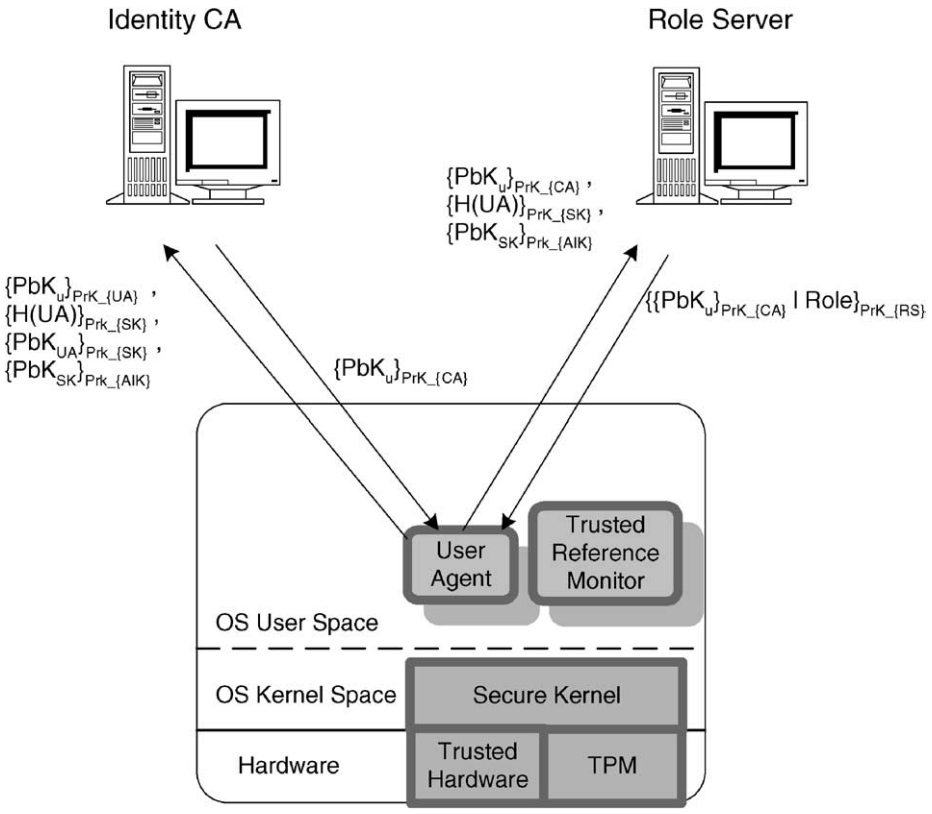

Fig. 6. Obtaining a role certificate with User Agent in a client platform.

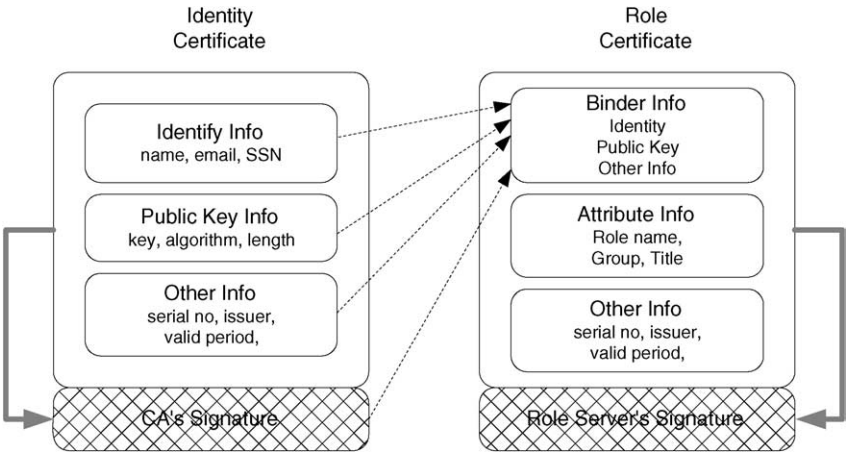

Fig. 7. Binding a role certificate with an identity certificate.

2. The identity CA verifies the public key certificates of UA and AIK, and some other related information of the user if necessary.[5] If all information is valid, identity CA issues a certificate for the identity public key of the user.
3. The UA sends the identity certificate along with the integrity value to the role server.
4. The role server verifies the certificate and other necessary information about the user, and issues a role certified by binding some information in the identity certificate, and sends back to the UA.

An implicit requirement behind this approach is that the identity CA trusts the privacy CA that certifies the AIK of the TPM. Furthermore, if a authentication service is applied in the platform, identity CA trusts this service and UA by attesting.

We apply the approach of Park and Sandhu [23] for the binding of a role certificate and an identity certificate, which requires trust between the identity CA and the role server. As shown in Fig. 7, a role certificate includes information about a user's role and some other information such as certificate serial number, certificate issuer name, etc. In addition, a binder block is included which selectively includes some information from the identity certificate, such as the subject name, public key information, identity certificate serial number, or the identity CA's signature in the certificate. The signature-based binding is tightly-coupled since each change of the identity certificate may require a re-issuing of the role certificate, while other bindings are loosely-coupled since a role certificate only includes some selected components of the identity certificate. If these components are not changed, the role certificate does not have to be re-issued. Which method is used depends on applications. For example, if an identity certificate is frequently renewed, and each time the user name is the same, then binding with the user name is a better choice. Additional details on certificate binding can be found in [23].

6.2.3. Role-based policy enforcement in TRM

To enforce role-based policy in a platform, a TRM first sends attestation challenge message to the UA in the local platform, and UA responds with attestation information. If the TRM trusts the running UA, it sends requesting message for role information of the user that invokes an application to access an object (the user context is provided to the TRM by the requesting application), which is either issued by a role server or migrated from another platform (discussed shortly). The UA sends back the role certificate of the user. Before this, the UA may also challenge the TRM to verify the TRM's integrity. On behalf of a user, the UA may submit the proof-of-possession for the corresponding private key of the identity public key, through an appropriate protocol. This process is similar to that described in Section 5.3.2, except that a role certificate can be sent to TRM directly, since it is not security sensitive.

[5] A registration authority may be available to accept certificate application, which requires some other information of a user (e.g., student ID, or social security number). For simplicity we ignore this component here.

6.2.4. Migration of user credentials

The concept of the identity of a user is similar to the attestation identity of a TPM. An attestation identity key is flagged as non-migratable when created, since it must be tightly bound to a single platform. In contrast a user can generally access a number of platforms, such as a desktop and a laptop in his company, and a home PC. Therefore a user identity key is a migratable object, and can be copied from one platform to another under the authorization of the user. In general the restriction on the set of the destination platforms that an identity key can be migrated to is determined by the identity owner (user). Some restrictions may be required by applications or global system policies. For example, a development engineer's identity credential can only be located in platforms within the department. Some critical applications may require that an identity credential is non-migratable, i.e., a payroll officer's identity key cannot leave a particular platform.

There are several approaches to migrate credentials between platforms. The proposed TPM specification provides a mechanism to copy a migratable key object protected by a TPM to another TPM. We apply this in our architecture. A simple example is shown in Fig. 8. It is "simple" because the identity key is copied from the source platform to the destination platform directly, which requires both platforms to be simultaneously available. For general case, intermediary entities may be needed, which is also supported by TPM specifications [10].[6]

The migration process is described as follows.

1. The UA of platform 1 (source) sends attestation challenge to the UA of platform 2 (destination).
2. The UA of platform 2 responds to the attestation challenge with the corresponding integrity value and its public key, signed by the SK.
3. The UA of platform 1 verifies the attestation. If platform 2 and its UA are trusted, the UA of platform 1 unseals the identity key and encrypts it with the public key of the UA in platform 2, and sends back to platform 2.
4. The UA in platform 2 decrypts the received identity key and seals it with a storage key of the local TPM.

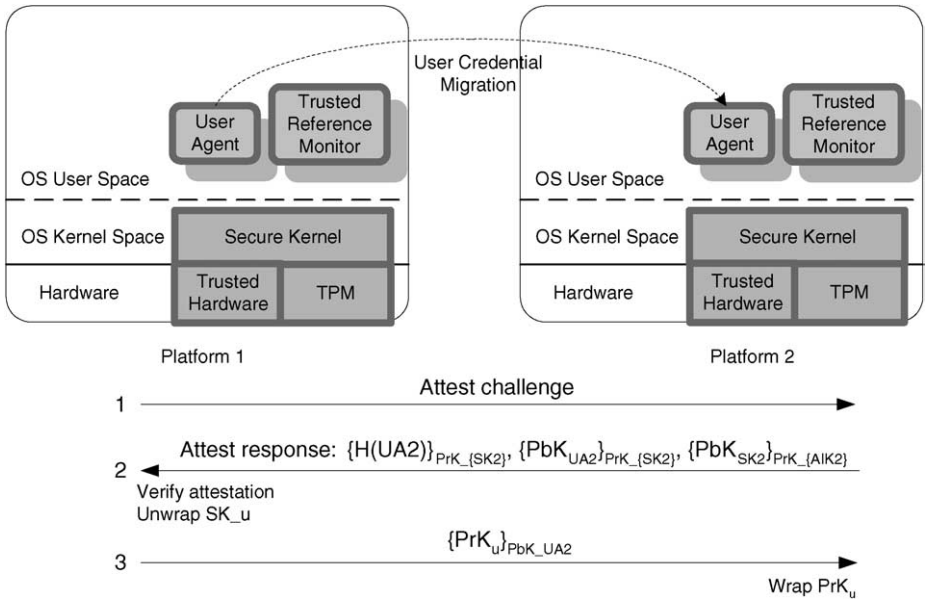

Fig. 8. Migration of user identity between platforms.

[6]Note that we focus on the migration of the private part of an identity key, since the public part is not security sensitive. But a public key and role certificate may be privacy sensitive, which needs protection. For simplicity we ignore this aspect in this paper.

7. Related work

In this paper we use an abstract platform beyond the underlying mechanisms of TC, including the hardware and kernel architecture, and the attestation mechanism. Our architecture focus on the application layer. Previous work has been presented to improve the primitive functions of TC. A trusted platform architecture is proposed in [24], in which PERSEUS kernel is applied on top of trusted hardware such as TPM. In [25], a property-based attestation mechanism is proposed, which extends the architecture of TCG trusted computing model and includes the property values of the remote side in an attestation. Similarly, in [17] a virtual machine based attestation is presented to capture the behaviors of remote entity. All of these can be applied as concrete underlying TC mechanisms in our approach.

Our approach is different from trusted operating systems, such as SELinux [21], Sun Microsystem's Trusted Solaris™, and the TrustedBSD [5] project. These projects enhance the security consideration in operating system layer by inserting authorization framework into the kernel to support access control modules, such as mandatory access control, multi-level security, etc. In this paper we enforce access control policies in application layer by leveraging underlying trusted computing functions.

An attestation-based architecture is presented in [26] to control access to cooperation server from remote clients. A policy agent is located in a client platform and verifies connection from client to server. The client system is attested with integrity measurement mechanism in Linux platform with TPM so trust is provided to the server that policies can be enforced correctly by the policy agent in the client side. The main difference between this and our work is that, in our approach, a policy is aiming to protect an object that is distributed to client platforms, and is enforced by a general-purpose trusted reference monitor. That is, our architecture is more flexible for highly distributed computing systems such as decentralized dissemination control and VoIP applications.

8. Conclusion

We present an architecture with trusted computing technology to support general client-side access control. Different from most traditional access control models and systems that focus on user property based policies, our approach considers the integrity and trust of platforms and applications that are used by a user to access an object, which is vulnerable from increasing software-based attacks in client platforms. By using proposed trusted computing technologies, a reference monitor in a platform can act as an agent of an object owner to enforce access control policies, which states that an object can only be accessed in a genuine platform with applications in valid states, such as integrity and configuration. General policies with user security attributes such as role-based access control can also be supported in our architecture by binding identity and attribute in a certificate and being protected by trusted hardware.

We do not include a centralized control component in our framework. As we have mentioned, for some functions, minimum online component such as policy service may be needed. These include instant policy revocation, object access log, synchronized policy update in different platforms, and centralized user identity storage. Our architecture can be extended to support these policies for client-side access control enforcement.

Acknowledgement

George Mason University gratefully acknowledges the collaborative and financial support of Intel Corporation in pursuing this research.

References

[1] Eros: The extremely reliable operating system, http://www.eros-os.org/eros.html.

[2] Skype, http://www.skype.com.

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

[4] Trusted platform module (TPM) security policy, http://www.trustedcomputinggroup.org.

[5] TrustedBSD: Adding trusted operating system features to FreeBSD, in: *Proceedings of the FREENIX Track: USENIX Annual Technical Conference*, Boston, MA, USA, 2001, pp. 15–28.

[6] AMD platform for trustworthy computing, Microsoft WinHEC, http://www.microsoft.com/whdc/winhec/pres03.mspx, 2003.

[7] Symantec Internet security threat report, Trends for July 1, 2003–December 31, 2003.

[8] *TCG Software Stack (TSS) Specification Version 1.10*, https://www.trustedcomputinggroup.org, 2003.

[9] TCPA resources, IBM Watson Research, http://www.research.ibm.com/gsal/tcpa, 2003.

[10] *TPM Main Part 1 Design Principles Specification Version 1.2*, https://www.trustedcomputinggroup.org, 2003.

[11] *Trusted Mobile Platform, Software Architecture Description*, http://www.trusted-mobile.org/, 2004.

[12] B. Balacheff, L. Chen, S. Pearson, D. Plaquin and G. Proudler, *Trusted Computing Platforms: TCPA Technology in Context*, Prentice Hall PTR, 2003.

[13] D.E. Bell and L.J. LaPadula, Secure computer systems: Unified exposition and Multics interpretation, Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford, MA, March 1975.

[14] E. Brickell, J. Camenisch and L. Chen, Direct anonymous attestation, in: *Proceedings of ACM Conference on Computer and Communication Security*, Washington, DC, USA, 2004, pp. 132–145.

[15] Department of Defense National Computer Security Center, *Department of Defense Trusted Computer Systems Evaluation Criteria*, December 1985, DoD 5200.28-STD.

[16] Department of Defense National Computer Security Center, *Trusted Database Interpretation of the Trusted Computer Systems Evaluation Criteria*, April 1991, NCSC-TG-021.

[17] V. Haldar, D. Chandra and M. Franz, Semantic remote attestation – a virtual machine directed approach to trusted computing, in: *Proceedings of the Third virtual Machine Research and Technology Symposium*, San Jose, CA, USA, 2004, USENIX, pp. 29–41.

[18] B. Lampson, Computer security in the real world, *IEEE Computer* (6) (2004), 37–46.

[19] R. Levin, E. Cohen, W. Corwin, F. Pollack and W. Wulf, Policy/mechanism separation in Hydra, in: *5th ACM Symposium on Operating Systems Principles*, 1975, pp. 132–140.

[20] H. Levy, Capability-based computer systems, Digital Press, 1984, Available at http://www.cs.washington.edu/homes/levy/capabook/index.html.

[21] P. Loscocco and S. Smalley, Integrating flexible support for security policies into the Linux operating system, in: *Proceedings of USENIX Annual Technical Conference*, 2001, pp. 29–42.

[22] J. Park and R. Sandhu, The UCON_{abc} usage control model, *ACM Transactions on Information and Systems Security* **7**(1) (2004), 128–174.

[23] J.S. Park and R. Sandhu, Binding identities and attributes using digitally signed certificates, in: *Proceedings Annual Computer Security Applications Conference*, New Orleans, LA, USA, 2000, pp. 120–127.

[24] A. Sadeghi and C. Stuble, Taming trusted platforms by operating system design, in: *Proceedings of the 4th International Workshop for Information Security Applications, LNCS 2908*, Berlin, Germany, 2003, pp. 286–302.

[25] A. Sadeghi and C. Stuble, Property-based attestation for computing platforms: Caring about properties, not mechanisms, in: *Proceedings of New Security Paradigms Workshop*, NS, Canada, 2004, pp. 67–77.

[26] R. Sailer, T. Jaeger, X. Zhang and L. van Doorn, Attestation-based policy enforcement for remote access, in: *Proceedings of ACM Conference on Computer and Communication Security*, Washington, DC, USA, 2004, pp. 308–317.

[27] J.H. Saltzer, Information protection and the control of sharing in the Multics system, *Communications of the ACM* **17**(7) (1974).

[28] R. Sandhu, E. Coyne, H. Feinstein and C. Youman, Role based access control models, *IEEE Computer* **29**(2) (1996).

[29] R. Sandhu, K. Ranganathan and X. Zhang, Secure information sharing enabled by trusted computing and PEI models, in: *Proceedings of ACM Symposium on Information, Computer, and Communication Security*, Taipei, Taiwan, 2006.

[30] X. Zhang, F. Parisi-Presicce, R. Sandhu and J. Park, Formal model and policy specification of usage control, *ACM Transactions on Information and Systems Security* **8**(4) (2005), 351–387.