Peer-to-Peer Access Control Architecture Using Trusted Computing Technology

Ravi Sandhu and Xinwen Zhang
George Mason University
{sandhu, xzhang6}@gmu.edu

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 in peer-to-peer environments. In this paper we propose a trusted computing architecture to enforce access control policies in such applications. 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 user-based control policies, cooperating with existing services for user identity and attributes. This architecture and its refinements can be extended in future work to support general access control models such as lattice-based access control, role-based access control, and usage control.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Security

Keywords

access control, trusted computing, security architecture, policy 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) [2]: 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 (including the owner) 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 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. Just as an example a recent study by Symantec [5] 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 [7]. The lack of strong security mechanisms on client platforms in general, and the push for an open environment on computing devices, such as PDAs, smart phones, notebooks, etcetera, 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 [23] followed by a number of research and commercial capability-based computers [17]. 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 [14]. 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 [15].

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). We will review these in the next section. There are several distinguishing characteristics of modern TC primitives relative to the earlier efforts cited above. First of all they are designed for a distributed and dynamic, open environment wherein "trusted" application software can be executed and protected from interference from other software on the same platform. Thus the trust mechanisms provide for greater security for software execution within a single platform. Secondly there is direct support for platform-to-platform propagation of trust. TC technologies seek to protect data in creation, processing, storage, and transfer primarily by exposing the cryptographic secrets required to access the data only to software which has a verifiable chain of trust, be it on a single computer or across multiple computers. Reliance on appropriate application software to actually enforce the security policy is an integral part of this approach. TC primitives include cryptographic operations and trusted storage of root keys as a foundation for security. This is a sharp departure from previous approaches to trusted computing. Notably, modern TC technology is a product of industry initiatives with little direct input from the academic and research communities. As a consequence there is hardly any exposure of this important technology in the research literature. Likewise there is little guidance to industry regarding the use of this technology in support of traditional and new access control objectives.

The main contribution of this paper is to illustrate how TC technologies can support access control architectures and mechanisms between platforms and users. Specifically, we propose an architecture 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.

The rest of the paper is organized as follows. Section 2 provides background and review of trusted computing as well as the motivation for new security requirements from emerging applications. Section 3 describes our access-control architecture with client-side trusted reference monitor and policy specifications. Applications of our approach are discussed in Section 4. Some related problems and work are discussed in Section 5 and 6, respectively. Section 7 gives our conclusions.

2. BACKGROUND AND MOTIVATION

In this section we first briefly introduce the emerging TC technologies, then present some applications that can benefit from these technologies. TC focuses on the assurance that an entity does behave in the expected manner through mechanisms of integrity measurement, storage, and reporting.

2.1 Trusted Computing Technologies

2.1.1 TCG and LT

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. In addition, TPM provides some primitive cryptographic functions, such as random number generation, RSA key generation and RSA asymmetric key algorithms. 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. A brief introduction is given below.

A TPM contains a set of Platform Configuration Registers (PCRs). A measurement of protected data or program code represents the properties and characteristics of the measured object, such as integrity, running states of a program, and configurations. The specific PCR's value is updated by applying SHA-1 operation on its current value concatenated with a new measured value. Therefore PCR values can record the integrity and state of a running platform from booting to loading operating system to loading applications.

With the integrity measurement and storage, an integrity report can be generated by a platform and provided to another platform through a challenge-response protocol called "attestation". During attestation, a platform (challenger) sends attestation challenge message to another platform (attestor). One or more PCR values are signed with an attestation identity key protected by the TPM of the attestor and provided to the challenger. The challenger verifies this attestation by comparing the signed values with expected values. Attestation provides the authenticity of a platform's current integrity, state, or configuration. Within a single platform, a running application can send attestation challenge message to another running application to verify its integrity or running state.

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.

A key is protected either by storing it in a TPM without releasing it, or encrypting it with a key that is protected by the TPM. This forms a key hierarchy where the leaves are protected secrets and arbitrary data, and the intermediate nodes are storage keys and identity keys. Each TPM has a storage root key (SRK) that is protected inside the hardware and is never released. A key can be a symmetric key, or an asymmetric key pair where the private part is the protected object. Each key has a flag with value *migratable* or *non-migratable*. A non-migratable key is created by a TPM and never leaves the platform, therefore it is guaranteed to be known only by the TPM that creates it. Entities who trust the TPM can thereby trust information protected by non-migratable keys. A migratable key can move from one platform to another. Trust in a migratable key goes back to the entity that creates that key, such as a certificate authority (CA).

In addition to a TCG compliant TPM, the LaGrande Technology (LT) of Intel [4] includes an extended CPU enabling software domain separation and protection, and extended chipset enabling protected graphics and basic I/O devices (i.e., keyboard/mouse), which enable trusted channels between application software and these devices. Beyond the hardware layer there is a domain manager supporting protected execution environments by domain separation, including separation of processes, memory pages, and device drivers. It is anticipated that these capabilities will be used in Mircosoft's next-generation operating system.

2.1.2 Trusted Mobile and Embedded Platforms

In addition to the PC platform-based approaches, there are emerging trusted computing technologies for mobile and embedded systems, such as the ARM TrustZone [10] and the Trusted Mobile Platform (TMP) [9]. TrustZone provides trusted computing in embedded systems by allowing trusted programs to run separately from others in the operating system. TrustZone provides basic trusted computing functions such as storing platform private key or master key in on-chip non-volatile or one-time programmable memory, platform identification, code integrity check, etc. Very recently, IBM, Intel and NTT DoCoMo published the TMP specification to bind TC with mobile wireless platforms. TMP is based on TPM, and the class 3 devices defined in TMP support the core functions of TC such as trusted boot, integrity measurement and reporting, and hardware-based domain separation and protected storage. With increasing transactions relying on mobile platforms such as PDAs and functionality enhanced cellphones [12], the trust between platforms becomes fundamental and opens new arena for security research.

2.2 Motivating Applications

The evolution of computing and business models in recent information systems motivates new security requirements. We illustrate this by considering some emerging applications.

2.2.1 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. Both of these goals can be supported by the emerging TC technologies. Trusted subject authentication may also be needed to enforce that only a valid user can access the object. We illustrate an example of this application with our proposed architecture in Section 4.

2.2.2 P2P Voice-over-IP

In recent P2P VoIP applications such as Skype, 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. We discuss this application in Section 4 with our proposed architecture.

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.

2.2.3 M-commerce

There are a number of emerging m-commerce applications which involve exchange or transfer of some form of electronic currency between peer platforms. The currency could be usage minutes on a VoIP phone card or some other service. This is an intriguing class of applications for TC. It is beyond the scope of this paper to investigate these applications.

2.3 Motivations for New Security Model and Architecture

We summarize the new requirements of the security model and architecture beyond traditional approaches as followings.

- *Change of trust relation*. In traditional security architecture, sensitive objects and policy enforcements are located on 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. In P2P systems, a platform can be both a server and a client, since it can distribute and receive objects.

- *Location of policy enforcement*. A significant requirement of the security architecture for emerging 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 policies are unmodified and correctly loaded. 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. Trusted path technology enables 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 architecture to support these new security requirements. Specifically we show how to apply TC technologies to enforce access control policies on client-side platforms.

3. TC-BASED ACCESS CONTROL ARCHITECTURE AND POLICY

In this section we first develop the main architecture of our approach based on some basic assumptions. Then we discuss the policy specifications within this architecture. Finally we show how to combine user attributes with the proposed architecture.

3.1 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 [3].
- We assume that the hardware layer of TC is tamper resistent or that hardware attack is not a threat.
- 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 identity attestation can be achieved from a trusted privacy CA [11] or through direct anonymous attestation [13] between platforms.

3.2 A Platform with Trusted Reference Monitor

An abstract platform is shown in Figure 1. The trusted components include trusted hardware with a TPM, a secure kernel, 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, sealed storage for sensitive data, and protected running environment. The sensitive data of TRM includes the 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.

An important security requirement is the integrity and confidentiality of the protected runtime environment for each application (including the TRM). 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 explicit opening a secure channel. Secondly, communications between applications through secure channels are protected and only available to the corresponding applications. Existing TC technologies such as LT and TrustZone support curtained memory space and process isolation and secure channels between processes. Also, separation of application domains has been specified for trusted mobile platforms [9].

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.

The secure kernel provides separation of execution between upper layer applications, and related services such as TCG Core Services (TCS) and TCG Service Providers (TSP). The secure kernel can be separated from the main kernel of the OS, such as the Nexus in Microsoft's Next-Generation Secure Computing Base (NGSCB) [1], or a special component of the main OS, such as the Trusted Zone access driver. A micro kernel architecture is proposed in [20].

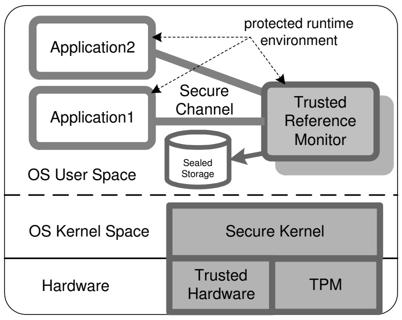


Figure 1: A platform architecture

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. As an example, a LT-based platform is shown in Figure 2. In 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 state (integrity measurement) of the platform, including secure kernel and running applications is stored in specific PCRs in the TPM.

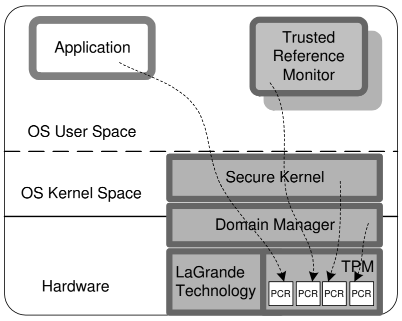


Figure 2: A LT-based platform

3.2.1 *Credentials*

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

- TPM attestation identity key (AIK) pair $(PK_{TPM.AIK}, SK_{TPM.AIK})$. 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 storage root key (SRK), and the public key certificate is issued by a trusted third party such as a privacy CA. A TRM can have a number of AIKs with certificates from different CAs.

- TRM asymmetric key pair (PK_{TRM}, SK_{TRM}). Each TRM has this key pair for signature and encryption. The private key is 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 in a certificate format signed by an AIK of the TPM.

- Application asymmetric key pair (PK_{APP}, SK_{APP}). Similar to TRM, each application has an asymmetric key pair. The private key is protected by the TPM, and the public key is in form of certificate signed by an AIK of the platform.

- TPM storage key(s) to protect TRM's credential and other sensitive data with sealed storage, such as secrets and policies. This key must be either the SRK of a TPM, or a key protected by the SRK.

3.2.2 *Primitive Functions of a TRM*

With the capabilities of TPM, 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.

- $TRM.UnSeal(\mathcal{H}(TRM), x)$. 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.

- $TRM.Attest(\mathcal{H}(TRM), PK_{TRM}) = \{\mathcal{H}(TRM) |PK_{TRM}\}_{SK_{TPM.AIK}}$. This function generates an attestation response by returning a certificate of the TRM's public key concatenated with its integrity value, signed with an AIK private key of the TPM.

3.3 Architecture

3.3.1 *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[1] (Alice's platform) attests a client (Bob's platform) and distributes policies and secrets to the client. The general process is explained as follows.[2]

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_B) in Bob's platform, i.e., Bob invokes APP_B to access OBJ. The integrity measurement $\mathcal{H}(APP_B)$ signed by one AIK of the TPM 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) may be included in this message.

2. The TRM of Alice's platform verifies the integrity of the requesting application. If Alice trusts APP_B to enforce some basic policies (e.g., APP_B will not save an object in plaintext to any persistent storage) as well as additional object-specific policies that Alice may specify (see next message), Alice's TRM sends attestation challenge message to Bob's TRM.

3. Bob's TRM calls $TRM.Attest(\mathcal{H}(TRM), PK_{TRM})$ function, which returns a certificate of Bob's TRM running hash and its public key, signed by the private key of his platform's AIK, and sends back to Alice.

4. Alice verifies the attestation. If Alice trusts the platform (that the platform has a genuine TPM and trusted booting) and the running hash 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.

Through the attestation challenge, Alice trusts that the TRM in Bob's platform can enforce the policy that Alice sent, which specifies the conditions that 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 environment, this may rely on some trusted third party. For example, an independent CA certifies that software with 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 during 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

[1]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 P2P interaction.

[2]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.

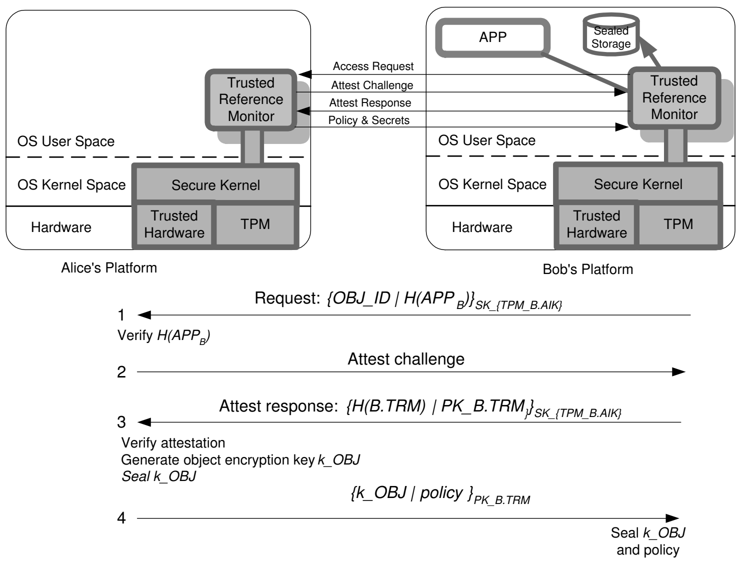


Figure 3: Basic architecture for client-side policy enforcement

available to a 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.

3.3.2 Policy Enforcement

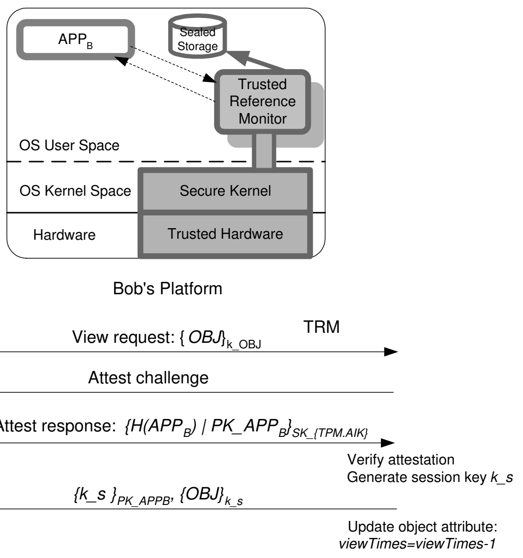
After secret and policy distribution, a subject on the client platform can generate an access request by invoking an application or process. The TRM in the platform checks the application's integrity state based on the policy information and makes an authorization decision. Figure 4 shows the enforcement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

1. APP_B 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_B.

3. APP_B responds with its running integrity measurement (one or more PCR values) and its public key signed by the TPM's AIK.

4. The TRM compares the integrity measurement with a list of expected values according to the policy. If APP_B is trusted, the TRM generates a session key k_s and encrypts it with the public key of APP_B, then sends to APP_B. 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_B. The TRM updates the policy if necessary, e.g., to update a usage count.



APP_B
Sealed Storage
Trusted Reference Monitor
OS User Space
OS Kernel Space
Secure Kernel
Hardware
Trusted Hardware
Bob's Platform
APP_B
TRM
View request: { OBJ}_k_OBJ
1
Attest challenge
2
Attest response: {H(APP_B) | PK_APP_B}_SK_{TPM.AIK}
3
Verify attestation
Generate session key k_s
{k_s }_PK_APPB, {OBJ}_k_s
4
Update object attribute: viewTimes=viewTimes-1

Figure 4: Policy enforcement within a platform

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 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

```
<?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_B</title>
      <version> 1.0</version>
      <integrity alg="sha1"> 0x487A3D... </integrity>
      <certificate> 0xA48ED...<certificate>
    </environment>
  </condition>
</policy>
```

Figure 5: A policy example

ensuring that the resources that it protects do not leak into other applications. Also, a TRM can be dedicated to a particular application.

3.4 Policy

A policy is created by an object owner (i.e., Alice in Figure 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.

3.4.1 *Policy Specification*

Formally, a policy states that an object can be accessed by a subject under what kind of conditions. 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 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.

3.4.2 *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) that 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.

3.5 Support for Policies with User Attributes

In the architecture described in 3.3, 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 under 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 Figure 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 [25] in our architecture with the mechanism of credential migration provided by TPM.

3.5.1 *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. A platform administrator may or may not be a TPM owner. Also, a platform user may or may not be a TPM user[3] [6].

Like a TRM, a UA in a platform has a credential (PK_{UA}, SK_{UA}). We assume that each user has at least one identity key pair (PK_u, SK_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

[3]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.

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, recently proposed TMP specification [9] 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.

3.5.2 *Identity and Role Certificates*

Just like the privacy CA for certificate of a platform attestation identity key, 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 also in form of a certificate 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 by 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 (PK_u), its own public key (PK_{UA}) signed with an AIK of the TPM, and its integrity measurement ($\mathcal{H}(UA)$) to the identity CA, signed with its private key. Note that the public key of AIK is supposed to be certified by a privacy CA.

2. The identity CA verifies the public key certificates of UA and AIK, and some other related information of the user.[4] 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, signed by the UA's private key, along with the integrity value and public key of UA, to the role server.

4. The role server verifies the certificate and other necessary information about the user, and issues a role certificate by binding some information in the identity certificate, and sends back to the UA.

A 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 [19] 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 Figure 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

[4]A registration authority may be available to accept an 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.

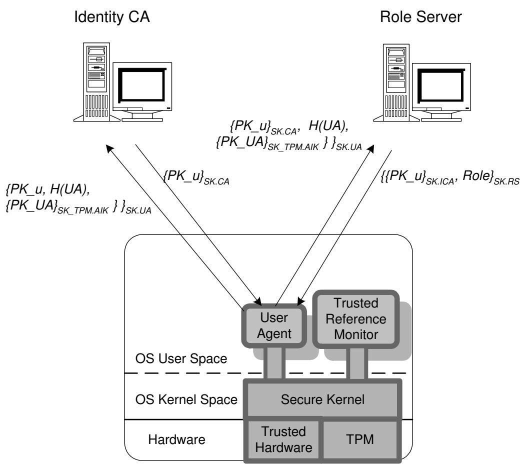

Identity CA
Role Server
$\{PK\_u\}_{SK.CA}$, $H(UA)$, $\{PK\_UA\}_{SK\_TPM.AIK}$ $\}$ $\}_{SK.UA}$
$\{PK\_u\}_{SK.CA}$
$\{PK\_u$, $H(UA)$, $\{PK\_UA\}_{SK\_TPM.AIK}$ $\}$ $\}_{SK.UA}$
$\{\{PK\_u\}_{SK.ICA}$, $Role\}_{SK.RS}$
User Agent
Trusted Reference Monitor
OS User Space
OS Kernel Space
Secure Kernel
Hardware
Trusted Hardware
TPM

Figure 6: Obtain 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 [19].

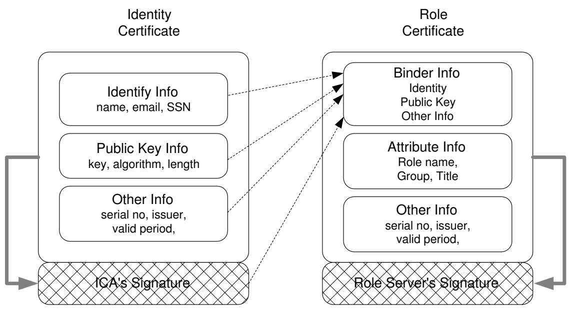


Figure 7: Binding a role certificate with an identity certificate

3.5.3 *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 3.3.2, except that a role certificate can be sent to TRM directly, since it is not security sensitive.

3.5.4 *Migration of User Credentials*

The concept of an 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 sin-

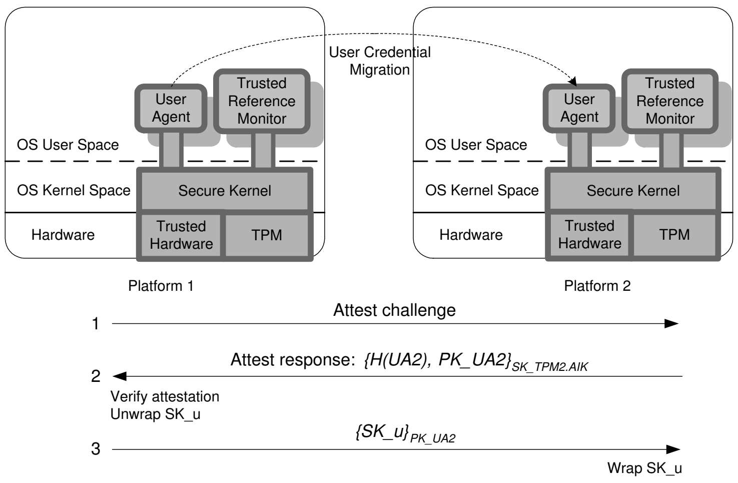


Figure 8: Migration of user identity between platforms

gle 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 Figure 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 [8].[5]

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 PCR value and its public key, signed with the AIK of the TPM in platform 2.

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.

[5]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.

4. APPLICATIONS

In this section we show how to apply the architecture introduced in Section 3 in various applications. First we show a document dissemination control application between platforms and users; then we illustrate how to protect end-to-end communication in a P2P VoIP application.

4.1 DCON

Dissemination control is a fundamental problem in access control. Our trusted access control architecture can be used to enforce some DCON policies such as "Alice gives OBJ to Bob's office desktop, where it can be viewed 10 times" (P1). Further we can extend it to include user role attribute in the policy, such as "Alice gives OBJ to Bob's office desktop, where it can be viewed 10 times only by employees of the company" (P2). Finally we consider the platform roaming, such as "Alice gives OBJ to Bob, whereby it can be totally viewed 10 times on office desktop and laptop" (P3).

4.1.1 *A Simple DCON Policy*

Policy P1 is platform-based only and can be enforced with the basic architecture of Figure 3 with policy specified in Figure 5 except that there is no <subject> entry. A parameter $viewTimes$ is defined for <right> to record the available times that the object can be viewed on that particular platform,. The value of this parameter is updated by the TRM in Bob's platform whenever an authorized access takes place. The policy is non-migratable since Bob cannot distribute it to other platforms.

4.1.2 *Including Role in Policy*

Policy P2 includes a role attribute of a user such that every "employee" in the company can view the document on Bob's platform. The policy is similar to that shown in Figure 5 except there is a single <subject> entry with type "role" and value "employee". The architecture proposed in 3.5 can be used to enforce this policy. Also, the policy is non-migratable.

4.1.3 *Platform Roaming*

A user can access an object from different platforms, which is called platform roaming. In the very simple case, both platforms are available at the same time, and the roaming can be done with direct migration of the key and policy from one platform to the other provided both of them are "migratable". The mechanism is similar to identity migration in Section 3.5. For more general cases, if two platforms are not available at the same time, one or more intermediaries may be involved, such as a centralized online service. In both cases, since Alice is supposed to be the object's owner, the TRM in Bob's office PC needs to get authorization of Alice before migration. For example, Alice can specify a set of platforms (i.e., expected integrity values of TRM signed by the AIK of platforms) that can receive the object specified in the policy, and the source TRM is trusted to enforce this policy by checking the destination platform. Also, an online policy service component can be introduced in the architecture to support platform roaming. This component should be trusted by object owner to store and update policies.

A problem of platform roaming is to maintain the consistency of possible policy updates in different platforms for a single object. For example, in policy P3, the policy specifies Bob can view the object in two platforms with a total usage of 10 times. If Bob reads once in his laptop by migrating the policy and secret from his desktop, the policy is updated ($viewTimes$ decreases by one). Then when Bob returns to view OBJ in the desktop, the policy should be synchronized to count the access in his laptop. A simple solution is that each time when a platform roaming happens, a policy migration is required. This is inconvenient for a user. Another solution is an online trusted policy service component, and each update is performed on this component.

4.1.4 *Secure redistribution*

All the above DCON policies are for one-step user-to-user dissemination. Our architecture can support multi-step dissemination control by specifying each step's information in the policy. For example, Alice distributes OBJ to Bob and allows Bob to distribute OBJ to Charlie. Alice puts both Bob's and Charlie's information (platform, running environment, and user attributes) in the policy and distributes to Bob. The TRM in Bob's platform can enforce the policy by checking Charlie's platform and user attributes upon requests. Again, a trusted online policy component can be another option to enforce multi-step dissemination control.

4.2 P2P Voice-over-IP Applications

While considerable work has been done in secure routing and network connections in P2P, we focus on the trusted end-to-end communication between platforms. Since each voice stream is encrypted by the initial platform, we ignore the security problem in intermediate nodes. Hence, the main security concern lies on the end platforms. A general requirement is the realtime protection for a VoIP conversation in a platform. Further, secure storage and forwarding of a voice object is becoming important with applications such as voice mail.

4.2.1 *Realtime Protection of Conversation*

Realtime protection requires that voice streams in a platform are not eavesdropped or illegally recorded by other processes during a conversation. Figure 9 shows an architecture for an end-to-end conversation. In addition to domain and process isolation in runtime space, another requirement for VoIP is the secure channel between the client application and sound card driver. Specifically, the TRM has to make sure that the communication between the client application and sound card is not compromised. TC technologies such as LT support a trusted channel between an application and input or output devices [4].

For VoIP applications, the distribution of voice stream encryption key and policy between TRMs is similar to that in Section 3.3. In addition to the client application, the integrity of the loaded sound card driver is also attested and verified by the TRM in the sender's[6] (Alice) platform. Mutual attestations may be needed if the receiver (Bob) wants to trust the sender's platform and application, since the audio conversation is bidirectional. For simplicity we only explain one-way object flow. In the sender platform, the VoIP client application accepts audio streams from the sound card through a secure channel with the device driver. An audio stream is encrypted by the TRM with the object encryption key (k_{OBJ}), and sent to the client application on the receiver side. The client application on the receiver side receives the encrypted audio object and sends to TRM for decryption, then sends to the sound card through the secure channel. The details of the policy enforcement in the receiver side is shown in Figure 10 and described as follows.

1. The TRM challenges and gets the attestation response from the client application.

2. If the integrity verification is valid according to corresponding policy, the TRM generates a secret key k_s and sends it to the client application.

3. The TRM challenges and gets the attestation response from the sound card device service (driver).

4. If the integrity verification is valid according to corresponding policy, the TRM sends secret k_s to the sound card service. After this, the client application and sound card service share the secret key k_s, which is used for secure stream flow between them.

5. If subject information is specified in the policy, the TRM sends attestation challenge message to the UA, along with the expected information of the user that invokes the VoIP client application.

6. The UA returns an attestation response message and related authentication information (attributes) of the user. If the TRM trusts the UA, and the user attributes satisfy the corresponding policy, the conversation is authorized, and the client application can receive and send voice streams now.

7. Upon receiving an encrypted voice stream, the client application sends it to the TRM, the TRM decrypts it with the secret key distributed from the sender's TRM, encrypts with k_s, and sends back to the client application. The client application re-orders (due to different delays of streams from the network) the receiving streams and sends to the sound card service, which sends to the hardware for playing. (This step is shown by the dotted lines in Figure 10.) The process of sending streams to the other platform is similar. Specifically, the client applications generates the streams, encrypts with the session key k_s, and sends to the TRM. The TRM decrypts it, re-encrypts with the object secrets shared with the TRM of the other platform (k_{OBJ}), and sends back to the client application, which deliveries to the network in turn.

Different approaches can be implemented for user-based authentication and authorization. For example, in instant message applications like Skype and MSN Messenger, the UA is integrated with the client application, through which a user can login with id and

[6]We user "sender" to refer the side that initializes a conversation, and "receiver" the other side. Actually during a conversation both sides send and receive voice streams.

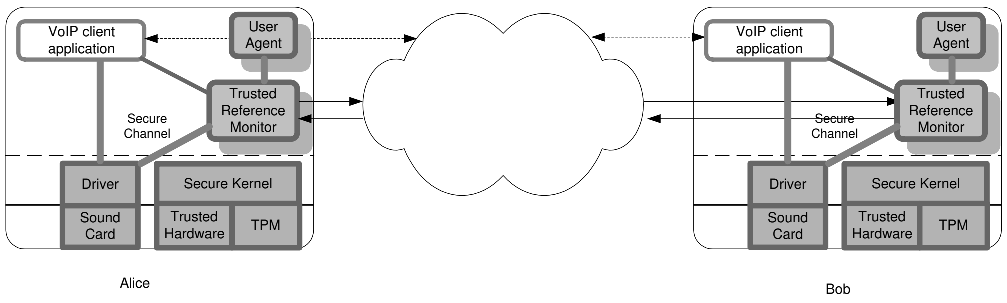


Figure 9: Secure VoIP architecture

password by communicating with centralized login server for authentication. For a cellphone platform, SIM card may be applied for user authentication.

Note that the shared secret k_s among the TRM, client application, and device driver is an one-time key; that is, it is negotiated for each conversation. The object secret (k_{OBJ}) negotiated between two TRMs can be one-time if there is no storage and re-play of the audio data after a conversation, otherwise it should be sealed by the TRM along with the policy for re-play. If it is one-time, k_{OBJ} can be sent to the client application by the TRM for better performance, so the client application does not need to send received streams to the TRM for decryption and encryption. But the attestation challenge of a client application by a TRM is necessary before allowing a conversation.

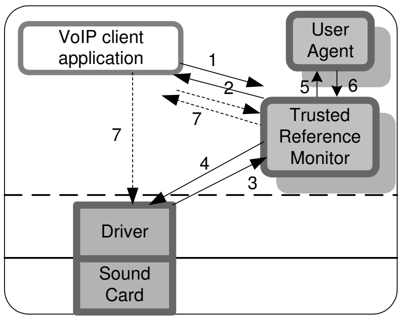


Figure 10: VoIP policy enforcement in a platform

4.2.2 *Secure Storage and Forward of Voice Mail*

An encrypted voice mail object can be saved and only available to an authenticated entity controlled by a TRM according to corresponding policy. The secret and policy are generated and distributed from original platform to a client platform, which is similar to the DCON architecture we have illustrated. As mentioned above, the additional requirement of a TRM to authorize playing a voice object is that the integrity of sound card driver has to be attested, and secure channel is constructed from sound card to the application, such as client mail process.

Secure forward of voice mail is similar to multi-step DCON. Not only a platform and TRM are attested before a secret and policy are distributed, the policy should include the expected integrity measurement value or properties that a client application and sound card driver. Fine-grained control policy for secure forward can be defined by an object owner.

5. DISCUSSION

The policy update in a TRM is the result of an access action, e.g., the available number of times that an object can be viewed in a platform is decreased by one after each authorized use. Update rules should be specified by a policy or object owner, which is referred as the attribute updates in UCON policies [27]. For simplicity we ignore this aspect in this paper. Another type of policy update concerns the change of authorization requirement, which is similar to the policy revocation and requires the availability of the policy owner. All these are related to a policy administration model and architecture, which is out of the scope of this paper.

We do not include a centralized control component in our architecture. 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 just provides a core part for client-side access control.

In the context of OM-AM framework for security engineering [24], we focus on the architecture layer in this paper. Some supporting components and mechanisms, such as policy administration and the user-role assignments for role-based access control policies, are out of the scope of this paper.

6. 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 [20], in which PERSEUS kernel is applied on top of trusted hardware such as TPM. Access control mechanisms are applied in a resource management layer beyond the kernel for system-wide policies. In [21], 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 [16] 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 the SELinux [18], the Trusted Solaris, and the TrustedBSD projects. 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 [22] 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.

An application of TC to protect pirates of entertainment products in P2P content distribution network is presented in [26]. The main idea is to use remote attestation to control a peer's joining the network and publishing contents. Only certified platforms and application that are trusted by the peers in the network can use the resource of a P2P system. This work and ours complement each other, since we focus on the enforcement of security policies in the platform of a peer, while their work aims to control a peer's joining or using resources in the system. Particularly, when a genuine peer joins the network and use the resources, the usage of an object in that platform may need further protection, such as realtime protection during playing media data, and re-distribution control for sensitive documents.

7. CONCLUSION

We present an architecture with trusted computing technology to support peer-to-peer based 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. Applications of the architecture in various domains show flexibility of deployment and enhancement of overall security in client platforms.

Acknowledgement

The authors thank Kumar Ranganathan and Carlos Rozas of Intel Corporation for valuable discussion and input which have considerably improved the quality and technical accuracy of this paper.

8. REFERENCES

[1] Next-generation secure computing base. http://www.microsoft.com/resources/ngscb.

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

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

[4] LaGrande technology architecture. Intel Developer Forum, 2003.

[5] Symantec internet security threat report, Trends for July 1, 2003-December 31, 2003.

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

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

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

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

[10] TrustZone: Integrated hardware and software security: Enabling trusted computing in embedded systesm. http://www.arm.com/products/CPUs/arch-trustzone.html, 2004.

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

[12] M. Baron. Bulverde and Marathon turn cellphones into PCs. *Microprocessor Report*, July 2004.

[13] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *Proc. of ACM CCS*, 2004.

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

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

[16] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing. In *Proc. of the Third virtual Machine Research and Technology Symposium*. USENIX, 2004.

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

[18] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proc. of USENIX Annual Technical Conference, June 25-30*.

[19] J. S. Park and R. Sandhu. Binding identities and attributes using digitally signed certificates. In *Proc. ACSAC*, 2000.

[20] A. Sadeghi and C. Stuble. Taming trusted platforms by operating system design. In *Information Security Applications, 4th International Workshop, LNCS 2908*, 2003.

[21] A. Sadeghi and C. Stuble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *Proc. of NSPW*, 2004.

[22] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proc. 11th ACM CCS*, 2004.

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

[24] R. Sandhu. Engineering authority and trust in cyberspace: The OM-AM and RBAC way. In *Proc. of Fifth ACM Workshop on Role-based Access Control*, 2000.

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

[26] S. Schechter, R. Greenstadt, and M. Smith. Trusted computing, peer-topeer distribution, and the economics of pirated entertainment. In *the Second International Workshop on Economics and Information Security*, 2003.

[27] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *Proc. of 9th ACM Symp. on Access Control Models and Tech.*, 2004.