On the Applicability of Trusted Computing in Distributed Authorization using Web Services

Aarthi Nagarajan[1], Vijay Varadharajan[1], Michael Hitchens[1], and Saurabh Arora[2]

[1] Macquarie University, Sydney, Australia
{aarthi,vijay,michaelh}@ics.mq.edu.au
[2] The Royal Institute of Technology, Stockholm, Sweden
arora@kth.se

Abstract. Distributed authorization provides the ability to control access to resources spread over the Internet. Typical authorization systems consider a range of security information like user identities, role identities or even temporal, spatial and contextual information associated with the access requestor. However, the ability to include computing platform related information has been quite limited due to constraints in identification and validation of platforms when distributed. Trusted computing is an exciting technology that can provide new ways to bridge this gap. In this paper, we provide the first steps necessary to achieving distributed authorization using trusted computing platforms. We introduce the notion of a Property Manifest that can be used in the specification of authorization policies. We provide an overview of our authorization architecture, its components and functions. We then illustrate the applicability of our system by implementing it in a Web service oriented architecture.

1 Introduction

Distributed computing can be described generally as a collection of individual computers communicating with one another. Recent advances in networking, end node processing power and software technologies have enabled distributed computing to be widely deployed. Distributed systems can be used to share resources as simple as printers or files to anything as complex as large business functions across an organization. When resources are spread across the Internet, controlling access to their usage becomes an important concern. Different resources have different access restrictions based on how important the resource is, who is requesting access, what actions on the resource have been requested and other environmental factors as time and place of request. This makes access control a challenging area for research.

Traditional access control mechanisms like maintaining access control lists cannot sufficiently express all these requirements. Such mechanisms normally suit systems with a centralized authority that administers access control policies where access requestors are known in advance. When systems are decentralized

in nature, it is possible that both the access requestor and the authorizer are strangers. The authorizer has to rely on third parties for gathering information about the requestors. When access is obtained, entities can further delegate their rights to other parties that they know. Absence of a central authority, reliance on third parties, rich access control requirements and issues like delegation make traditional access control systems unsuitable for distributed systems.

Trust management systems were introduced to address some of these issues. The term 'Trust Management' was first given by Blaze et al [1] when they introduced the PolicyMaker system. It was described as an unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions. Since then, there have been many implementations of such systems like KeyNote [2], Binder [3], REFEREE [4] and IBM's trust management framework [5]. A trust management system provides a flexible mechanism usually in the form of a policy language to specify the authorization requirements of a system. The heart of a trust management system is the authorization engine that evaluates whether an access request can be granted or not based on a number of conditions. Authorization credentials are loosely coupled to permissions and are usually created, distributed and managed by the trust management system. Furthermore, the framework can itself be extended to support features like delegation and trust negotiation. Trust management systems, thus move the notion of authorization from a closed-centralized approach to a more open and distributed approach.

In this paper we provide the first steps necessary to build a trust management framework using trusted computing platforms. The structure of the paper is as follows. Section 2 discusses about trusted platforms and attestation. In section 3, we motivate the need for a trust management framework based on trusted platforms. Section 4 introduces the notion of a Property Manifest. In sections 5,6 and 7 we define our authorization system, its components and working. Section 8 looks at an application of the proposed system using Web services. Section 9 discusses about some issues and challenges in using trusted platforms for distributed authorization and we conclude in section 10.

2 Trusted Computing Platforms

In the recent years, computers have become complex with large number of software applications running on them. When these computers get connected to the Internet, they risk data exposure and compromise due to software attacks. Computers have also become mobile and are at constant risk of physical theft or loss. As these risks escalate, it has led to the realization that security mechanisms using software alone are in-sufficient. The use of hardware based security is becoming an important approach to protecting information. Trusted computing technology developed by the Trusted Computing Group (TCG) is an effort that aims to provide techniques for achieving security using hardware in computing platforms.

The core of the trusted computing technology is a Trusted Platform Module (TPM) [6] chip that is embedded in the motherboard at the time of manufacture. A TPM chip is similar to a secure co-processor. It performs certain cryptographic functions and provides secure storage for secrets and data. When hardware and software components are manufactured for a trusted platform, they are supported with information regarding the provenance of the component by its manufacturer. The manufacturer provides a 160-bit binary measurement value that indicates a component's good working state. These values are called the reference values and are represented using the TCG Reference Manifest (RM) [2] structure. A Reference Manifest contains information regarding the identity, version and manufacturer of the component along with the measurement. The TPM also creates a public-private key pair called the Attestation Identity Key or AIK. While the public AIK is used to identify a trusted platform associated with an user, the private AIK is used by the TPM for signing purposes.

Perhaps the most important feature of a trusted platform is its integrity measurement mechanism. When a trusted platform boots, all processes starting from the boot measure the next process to be loaded. All measurements are in the form of a 160 bit hash that are stored inside special registers called the Platform Configuration Registers (PCR) within the TPM. As the number of measurements outnumber the available PCRs (usually 16 for a PC), a hash of the concatenation of the new measurement with the old measurement is stored in the PCR. A log of the measurement history is also stored outside the TPM in local storage.

2.1 Attestation in Trusted Platforms

When a communicating host wishes to learn the state of a trusted platform, it initiates a process known as 'attestation'. During attestation, TPM creates the 'Quote' blob by collating the requested PCR values and by signing them using the private Attestation Identity Key. A Platform Trust Service (PTS) then generates an Integrity Report using the TCG Integrity Report [3] structure. The report includes the Quote information, references that point to the Reference Manifests and the measurement log. When the host receives the report, it validates each individual measurement inside the log against the corresponding Reference Manifest value, recomputes the PCR values using the log and matches them against the Quote values. If all the values match, it believes that the trusted platform is integrity proof.

Attestation mechanism which is strongly founded on binary measurements has certain limitations. Hash measurements change every time there is a 'minor' modification in the implementation. Security updates, version updates and patches applied can continuously change the expected measurement of a component. Measurements are also not human understandable as they are stored as binary inside the TPM. These reasons limit the usability of measurement values as authorization parameters in security policies. Another argument is that measurements relate only to the code or logic of a component. This can move the focus to implementations rather than properties of systems favoring certain

vendors and their products. Recent efforts like Property based attestation [7–9], an extension of the attestation mechanism try to address these issues by combining binary measurements with security properties of systems. Property based attestation aims to prove that the availability of a certain measurement guarantees the availability of a certain security property thereby abstracting low level binary values to more meaningful attributes.

WS-Attestation [10] proposed by Yoshihama et. al. extends the attestation architecture on the Web services framework. To include the attestation architecture for Web services, it extends the bootstrapping process. The root of trust is a trusted BIOS which begins the measurement process and measures all the components up to and including the middleware layer. The middleware then measures all data it loads or uses in the platform. This way the transitive chain of trust is built from the trusted boot all the way up to a Web service application loaded on the platform.

3 Authorization using Trusted Platforms

In a distributed system like Web services, there are Service Providers (SP) who provide services and there are Service Requestors (SR) who receive services. When SP receives a service request, it has to answer at least two questions. Is SR the one it claims to be and does it have the necessary privileges for the requested service. These two basic questions relate to the issues of authentication and authorization. The authorization requirements in distributed applications are much richer than the authentication both in terms of the types of privileges required and the nature and degree of interactions between participating entities.

Authorization systems have usually been able to define policies from a user's context and not based on the user's computing environment. Users here we mean human beings who wish to have access to a certain service. Of course, one can think of simple policies like those based on the network address of a requestor or the application (e.g the browser) from which a request has been made. But the ability to include useful information like security properties or behaviour of platforms has been very limited. This is because it is difficult to remotely identify a platform and validate its claims. Software can either be manipulated to produce false claims or the validation technique itself can be manipulated to prove non-existing claims. Therefore, it is common practice to assume that the underlying platform from which a request is made is sufficiently 'secure'.

With the introduction of trusted computing, it is possible to address such limitations. Trusted computing provides mechanisms to both identify platforms and validate claims made about a platform. All users receive Attestation Identity Key credentials that identify them with respect to that platform. AIK keys could be used to identify a platform on the Internet rather than using identities like MAC addresses and IP addresses. Attestation keys which are created and stored inside the TPM may not be as easily spoofed as MAC or IP addresses. Secondly, trusted platforms support attestation which is founded on hardware based trust. Attestation provides a mechanism to validate actual measurements

of components against the reference values. When combined with property based attestation, a platform can guarantee the existence of certain security properties.

In this paper we extend the notion of traditional trust management systems from an user-only approach to an user and platform based approach. We provide the necessary first steps towards achieving platform and property based authorization. Firstly, we believe defining suitable credentials for property based attestation is important. We introduce the notion of a Property Manifest that has been discussed in detail in section 4. An overview of the authorization system and its components is available in section 5. Section 6 discusses about the interactions between the entities of the system using the push, pull and delegation models. Section 7 outlines our extensions to XACML to support platform based authorization.

4 Property Manifests

In this section, we introduce the notion of a Property Manifest. Property Manifest (PM) is the representation of a platform's security properties. It is created and issued by a Certification Authority (CA) which can be a trusted third party, e.g manufacturer of a component. The purpose of a Property Manifest is to support the mapping of a component to its security properties. Each Property Manifest may describe a trusted platform as a whole or a component of the platform. However, there may be sub-components each of which may have corresponding Property Manifests. A Property Manifest is represented in an XML based Property Manifest structure. It contains information such as the component identity, manufacturer, model or version number, and others. In order for the Property Manifest to be useful with in a given context, the Reference Manifest data must be made available.

Security properties are closely bound to components that they belong to. Therefore, with a given security property, it might actually be possible to detect some information about the component. This is specially possible when certain properties are unique to components. This defeats the purpose of property based attestation in the first place because property based attestation tries to hide the implementation details of a component. Revealing security properties of components can therefore pose some privacy concerns for a trusted platform. For this reason, we define properties at three different levels of granularity (of course, more levels of granularity are possible). At Step 1 of the pyramid or S1, properties are very coarse. S1 properties are helpful to prove that some security property is available in the component without revealing the implementation details. For example, a Service Provider application guarantees 'confidentiality' and 'privacy' of a Service Requestor's data without revealing how this is actually achieved. At Step 2 or S2, properties reveal more detail. Service provider guarantees 'confidentiality by encryption' and 'privacy by data deletion' pushes the properties to the next level of granularity. At step 3 of the pyramid or S3, properties reveal implementation details of a component. For example, Service

Provider guarantees 'confidentiality by encryption using DES' and 'privacy of data by deleting it on the 7th day after transaction' are very fine grained policies.

The Property Manifest schema consists of the following elements. A 'Property' element of complex type that includes property related information PropertyID, Name, Value and Type. 'PropertyID' is the unique identifier of a property and 'Name' is the simple name given to a property. 'Value' is the element that indicates if the value of the property has been evaluated as true or false or undetermined. 'Type' is the element that includes property granularity information like S1, S2 or S3. Property Manifest schema also includes the ManifestID, ComponentID and the Certification Authority elements. 'ManifestID' is the unique identity (e.g. UDDI) of a Property Manifest. 'ComponentID' is a set of attributes accommodating a wide range of change management schemes that when combined uniquely identifies a particular version of a component. It is drawn from the Core Integrity Schema [11] which is also used in the Reference Manifests. The 'Certification Authority' element contains attributes of a CA like name, identity and signature details. More details of the core schema itself can be found at [11].

5 Overview of the System Architecture

In this section, we provide an overview of the system architecture for distributed authorization using Web services. The architecture consists of three main entities, the Service Requestor or SR, the Service Provider or SP and a trusted third party called the Validation Service or VS. The Service Provider hosts and publishes one or more Web services. It has several access control requirements for each of the services it offers. Requirements differ based on the type of service offered and the type of requestor requesting the service. Requirements may also depend on other factors such as time of the day, place of request or other environmental attributes. A Service Requestor is an entity that discovers the services offered by Service Provider and makes a request for one or more of these services. A Validation Service VS is a trusted third party that performs one or more functions on behalf of AP or AR. Its main function includes the verification of the Integrity Report. However, it can also be used for the verification of authorization policies. This is especially useful if many entities have shared policies and trust the VS to do policy verification on their behalf. Reference Manifest repositories and Property Manifest repositories store the Reference Manifests (RM) and the Property Manifests (PM) respectively. There is also a Policy Repository (PR) that stores the authorization policies of the Service Provider. We choose XACML policy language [12] to define authorization policies as it is well suited for Web services. All entities with in the system communicate using Web services.

5.1 System Components

We now define each component and its sub-component that have been implemented in the system.

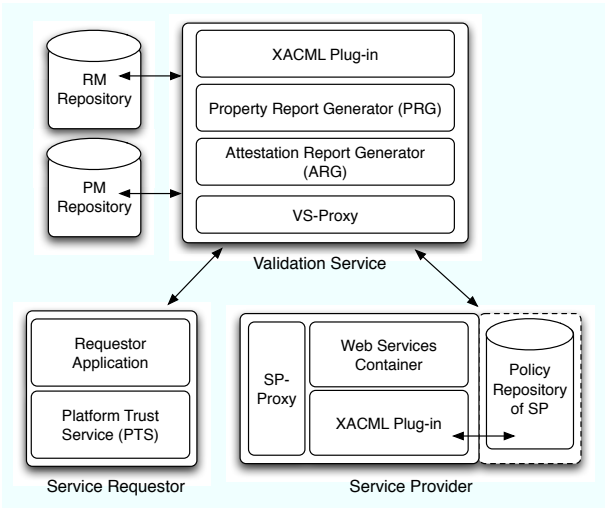

Fig. 1. System Architecture

1) Service Provider (SP) is a hardware platform that hosts different Web services. SP can be any type of hardware platform and not necessarily a trusted platform. Its main functions include receiving a request, checking if access can be allowed, and servicing or denying the request. It consists of the sub-components Proxy, WS Container, XACML plugin API.

SP-Proxy is a sub-component of the Service Provider. Proxy is also the first point of communication to an SP hosted Web service. It can be thought of as an abstraction of a Web service application which performs certain functions on its behalf. Its main functions include acting as the central point of communication between the different SP components, and communicating with the Service Requestor and Validation Service.

Web Service Container (WSC) is a collection of Web services offered by the Service Provider each of which can be discovered and invoked by a Service Requestor.

XACML Plugin provides an API for the inclusion of a standard XACML engine inside the provider. An XACML engine is the core component of the XACML access control model and is complex in its functionality. In short, it accepts requests for policy evaluation. It collects the necessary policies from the repositories. It evaluates the policies and resolves conflicts to arrive at a decision. It passes its decision on to the requester for necessary action to be taken. We do not discuss the XACML working model in detail and the specification [12] can be referred for more information.

Policy Repository (PR) is a repository that is used to store all the authorization policies of a Service Provider. This repository can be located anywhere in the Internet especially if the Service Provider is distributed in nature. As

the repository is administered by the Service Provider, this association is shown using dashed lines in the diagram.

2) Service Requestor (SR) is the entity that requests access to a service from SP. It is a trusted computing hardware. It is assumed that all the components of the platform have the corresponding Reference Manifests and Property Manifests in some repository. It the Manifests are not available, then it is assumed that they can be obtained on demand. It consists of the Platform Trust Service and Requestor Application.

Platform Trust Service (PTS) on a trusted platform performs the function of generating an Integrity Report (IR) as defined in section 2.1. At the time of attestation, PTS collates the TPM signed PCR values, Reference Manifests or their pointers, Property Manifests or their pointers and the measurement log information into an Integrity Report. This report known as the Integrity Report is used during attestation.

Requestor Application is the component that invokes a request on a Service Provider. It is ideally a Web browser. It can also communicate with the Validation Service.

3) Validation Service (VS) is a third party trusted by both the Service Provider and the Service Requestor. It can be located any where in the Internet, can be any type of platform and should be accessible by the Service Requestor and the Service Provider. It consists of the VS-Proxy, Attestation Report Generator, Property Report Generator, and an XACML plug-in API. In the proposed system, a VS primarily serves three different purposes. It is used for attestation verification, Property Report generation and policy verification.

VS-Proxy is the first point of communication to a VS. SP and SR communicate with the VS-Proxy to start the attestation verification process. It aids communication between the different components of VS like PRG and ARG. It can also invoke other Web services on behalf of VS. For example, it communicates with PTS to request for an Integrity Report.

Attestation Report Generator (ARG) is responsible for the verification of the Integrity Report. ARG is presented with an Integrity Report, the Reference Manifests and the measurement log as inputs. Using these, it verifies the Integrity Report as defined in 2.1. After verification, it generates an Attestation Report that includes the result of the verification process.

Property Report Generator (PRG) performs the functions of a Property Report generation. When VS receives an Integrity Report, it verifies whether the binary values are validated using the Reference Manifests. If the measurements are valid and the component is integrity protected, it looks up the Property Manifest to check if that component satisfies any property (a property satisfied by a component becomes invalid if the component is not in its measured state). It picks up the properties satisfied by that component and collates them into a Property Report. A Property Report is heavily drawn from the Core Integrity Schema [11]. It includes details of components, their properties and property type information along with the signature of the Validation Service. A Property

Report can be thought of as a summary sheet of all the properties satisfied by a trusted platform.

XACML-plugin is available in the Validation Service also. It provides an API for the inclusion of a standard XACML policy engine with in the Validation Service. This enables a Validation Service to not only generate attestation credentials and Property Reports but also to validate policies on request. In such cases, it also has access to SP's policy repository for policy evaluation.

4) Reference Manifest (RM) Repository is responsible for storing the Reference Manifests of different components of a trusted platform. Each component manufacturer can stand-up its own Reference Manifest database for all its products, or such a database can be made available by a third party. RM Repositories can be located anywhere in the Internet and be accessed by SP and VS when required.

5) Property Manifest (PM) Repository The PM repository is responsible for storing the Property Manifests of different components of the platform. Each manufacturer can stand-up its own Property Manifest database for all its products. A trusted third party can also host a Property Manifest repository for all the components that it has evaluated.

6 Authorization of Web Services

In this section we provide an overview of the authorization mechanism of a Web service. This system supports three different authorization models which are push, pull and delegation. We provide a brief description of each of the models.

The Pull Model - The pull model authorization is initiated when the Requestor Application of SR makes a service request to the provider's Proxy. Like in standard authorization systems, the Integrity Report is appended to the request in order to prove the possession of necessary privileges to access a service. If the Integrity Report is not available during request, then SP-Proxy can initiate a report request and obtain it from the PTS. Once the Integrity Report is received, SP-Proxy invokes the ARG service of VS using the VS-Proxy. The Integrity Report is first validated by ARG using the Reference Manifests from the RM repository. Then the request is passed on to PRG with the Attestation Report. For all the components whose binary measurements have been validated, the property information is looked up in the Property Manifests available in the PM repository. The Property Report is then generated, signed and sent back to the SP-Proxy. The SP-Proxy then sends the request from the requestor and the Property Report from VS to its XACML engine. The XACML engine verifies the request against the access control policies available in the policy repository. It arrives at a decision and sends its decision as allow or deny back to the SP-Proxy. The Proxy then forwards the decision to the Web service in the WSC. Depending on the decision, the service request is either accepted or rejected.

This model has certain design issues to be considered. Firstly, it is assumed that the Validation Service is trusted by SP and more so by SR. This is because, VS is chosen by SP in this model and SR should trust that VS will generate a

correct Property Report about its platform. There can be an initial negotiation phase where SP and SR agree on the Validation Service that will be used. Secondly, the provider has to wait until the Integrity Report has been verified and a Property Report is generated. If the provider trusts VS enough, then it can make its policies also available along with the Integrity Report. VS can now not only generate the Property Report, but also validate the policies on behalf of the provider and pass its decision to SP. Thirdly, when the Property Report is being generated, if certain components exists in the platform whose measurements do not validate against the reference values, then the Property Report can include a list of such component identities. This could be useful for the requestor to subsequently take the necessary actions (as re-installation of components) in order to ensure that those properties are made available. Also, when the access control policies are being verified against the Property Report, if certain properties are missing in the report that may be required by the policy, the response from the XACML engine could include a list of missing properties that are required.

The Push Model - In the push model, Property Report generation is initiated by the Requestor Application of SR. The Requestor Application first invokes VS-Proxy by providing its Integrity Report. ARG of VS generates the Attestation Report which is then passed on to the PRG. PRG generates the Property Report using the Property Manifests and sends it to the Requestor Application. The Requestor Application now invokes the Web service of SP with the Property Report. The SP-Proxy receives the Property Report which it forwards to the XACML engine. The engine verifies policies as usual and determines if the properties in the Property Report is sufficient to allow access to the service. The XACML engine sends its decision to the Proxy which is then forwarded to the Web service for action.

The obvious disadvantage of this approach is the time of creation and time of use problem. As the Property Report is created much in advance before the request is initiated, the Service Provider cannot be sure that the report reflects the most recent state of the requestor's platform. The provider can also have policies to define how fresh it expects the property credential to be. In this model also, if SP wishes to use a Validation Service for policy evaluation, it's Proxy can invoke the XACML engine of VS by forwarding the Property Report obtained from the requestor along with its access control policies. When the Proxy receives the policy decision from VS, it can forward it to the SP Web service. Again, there can be an initial negotiation phase on which VS the requestor can use and which VS the provider can use as the VS's can be different entities.

The Delegation Model - In the delegation model, a Service Provider delegates the Validation Service to do all the work on its behalf. When SP's Proxy receives a service request from a requestor, the request is automatically forwarded to the VS-Proxy of the Validation Service. VS-Proxy receives the request and checks if the Integrity Report is available. If the report is not available, it invokes the PTS service of the requestor (it can be assumed that the requestor's URL is available in the request and its PTS service is discoverable). VS-proxy first communicates with ARG to generate the Attestation Report and then with

PRG to generate the Property Report. If required, it also evaluates the access control policies and forwards its decision to the SP-Proxy. SP-Proxy then sends the decision to the Web service which acts accordingly.

7 Policy Extensions

Languages for access control aim to support the expression of authorization policies. While a policy language should be simple enough to understand and manage, it should also be expressive enough to accommodate all the authorization requirements of the system. Recently, there has been a lot of work on mark up language based access control policy languages like SAML [13], IBM's XACL [14] and XACML [12] due to their applicability in Web services. XACML has been already accepted by the Web services community and the WS-XACML specification [15] provides ways to use XACML in the context of Web services for authorization, access control, and privacy policies. In this section we briefly explain the extensions to the XACML policy statements necessary to include platform related property information.

7.1 XACML Policy Statement Extensions

The XACML specification, defines a <Subject> element in <Target> as the actor to whom the policy may be applicable to. Here, a subject could refer to the human user that initiated the application from which the request was issued or the application's executable code responsible for creating the request or even possibly the machine on which the application was executing. Although, the specification has some provision for limited platform related information, it is not expressive enough to include the components of a platform and their properties. Extensions to the <Target> element are *Rule/Target/Platforms* to include platform details, *Rule/Target/Platforms/Platform/Components* to include platform component details and *Rule/Target/Platforms/Platform/Components/Component/Properties* to include properties of components and their types.

7.2 Extensions to XACML Context Request and Response

An XACML context is a canonical representation for the inputs and the outputs of the policy evaluation engine. The input context is called the context request and the output context is called the context response. The <Request> element is a top-level element in the XACML context schema which contains the <Subject>, <Resource>, <Action> and <Environment> elements. Similarly, The <Response> element is a top-level element in the XACML context schema. The <Response> element encapsulates the authorization decision produced by the policy evaluation engine after the policy evaluation process has been completed. The <Result> element includes the <Decision> element with the policy decision, the <Status> element to indicate whether errors have occurred during the evaluation process and the <Obligation> element that need to be sent to

the Service Provider. We refer the reader to [12] for more information on the XACML context schema.

The context <Request> is extended to include information about the requesting platform, its components and its properties. For the context <Response> element, the <MissingAttributeDetail> element inside <Status> is extended to include the details of the component and its properties that were missing or unverified at the time of policy validation.

8 Application Scenario

In this section, we provide a sample scenario for the application of property based trust management to protecting medical records in hospitals and clinics. Many people consider their health issues as very private and expect the strongest protection against misuse. Medical records always need to be transferred between different entities who may not all be trusted. Health records need to be shared by different hospitals because the patient might not always visit the same hospital (in case of an emergency, or moving cities etc). Within the same hospital, different doctors and health workers like nurses need to access information based on who is attending the patient. There are also other entities like government agencies, pharmaceutical companies, insurance companies, ambulance services, and others who might require access.

In this section, we provide an example to show the applicability of trust management with trusted platforms in protecting medical records. Let us imagine that a patient Bob who normally visits hospital Hosp-A needs a specialist's consultation at Hospital Hosp-B. All hospitals register to one or more trusted third party brokers who aid in the sharing of information and protecting the interest of a patient. The first time Bob visited Hosp-A, Bob was asked to chose one such broker on his behalf if in the future, information has to be transferred to another hospital. Bob chose Brok-A. Hosp-A has promised Bob that 1) information will be securely transferred such that no illegitimate party can gain access 2) His health records will not be manipulated on transfer 3) Hosp-A will maintain a audit on all transfers and 4) Hosp-B will not store the information for more than one month without Bob's consent.

Hospital A (Hosp-A): Hosp-A is the Service Provider entity. Hosp-A hosts a 'Records Provider' service as a Web service. Any authorized party can invoke this request providing a patient ID as input. The service either rejects the request or provides the ID's corresponding medical history record as output.

Broker A (Brok-A): Broker is the Validation Service entity and performs functions as described in section 5. The broker provides the 'Delegation Proxy' service as a Web service. It is invoked by a delegation request with Patient ID and requestor ID as inputs. The output is either a 'permission granted' or 'permission rejected' response. Its main functions are to invoke the Platform Trust Service (PTS) of the Service Requestor and to communicate internally for generating the Property Report. Brok-A also has access to Hosp-A's XACML policy describing the conditions under which Bob is willing to disclose his data.

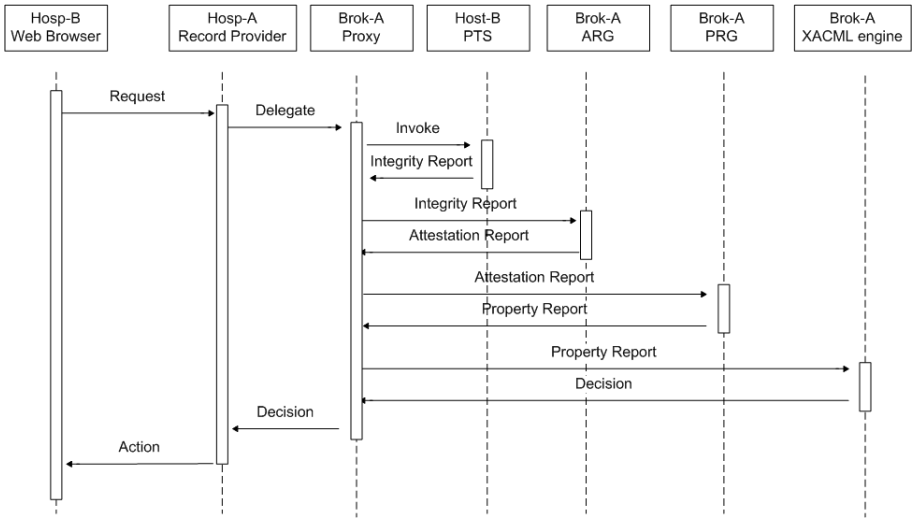

Fig. 2. Sequence Diagram

Hospital B (Hosp-B): Hosp-B is the Service Requestor. It is identified using an ID that is unique in the Internet (e.g UDDI). It has a Requestor Application that can invokes a request to another hospital providing the patient's ID and its own ID. All requests are sent out from trusted machines in the hospital (with a TPM). Host-B supports attestation mechanism using its PTS service.

The following diagram shows the sequence of events between Hosp-A, Brok-A and Hosp-B. We assume that the scenario follows a delegated model as described in 6. When Bob arrives at Hospital B to consult a specialist, Hospital B's platform launches a request on Hospital A's record provider service by entering Bob's patient ID and its ID. This request on Hosp-A recogizes Bob's ID and checks for his preferred broker. It identifies Brok-A and automatically delegates the request by invoking the Brok-A's Proxy. The Proxy first invokes the PTS service of Host-B (it is assumed that the Proxy can discover PTS of Hosp-B). PTS generates the Integrity Report and presents it to the Proxy. Proxy uses the Integrity Report to invoke the Attestation Report Generator service in VS. ARG now generates the Attestation Report by looking up the Reference Manifests and returns it to the Proxy. Proxy invokes the Property Report Generator service in VS. PRG generates the Property Report looking up the Property Manifests and returns it to the Proxy. Essentially, the Property Report must contain information that will suffice the four conditions of Bob. Hosp-B could achieve this by installing the necessary components that will provide confidentiality of data for condition 1, integrity of data for condition 2, a logging component for condition 3, a policy enforcer that ensures that data will be deleted after one month for condition 4. Hosp-B then proves that all these components are integrity protected and that they have the necessary Property Manifests. When Brok-A Proxy receives

the Property Report, it presents the report to the XACML engine. The XACML authorization engine checks if PR has the necessary properties to satisfy the policies of Hosp-A available in the Policy Repository. It arrives at a decision, either to allow or deny the request which it forwards to the Proxy. Proxy forwards this decision to the Record Provider service which enforces the action accordingly. If the decision is to allow, it sends Bob's medical history file over to Hosp-B.

9 Discussion

In this section we discuss some issues that are relevant to property based attestation. One important issue to be considered is how much flexibility should one have when it comes to software updates such as patches, given that state of the platform and configuration will change in these circumstances. One of the main motivations to use properties instead of hashes of configurations is that properties do not necessarily change as often as hashes do. On the one hand, we do want to reflect state changes of the attesting platform to the challenger to decide whether it should interact or not. However if every time a 'minor' change happens, a new Property Manifest needs to be generated, this would limit the usage flexibility. of course, here the issue is to determine what changes are 'minor' and do not affect the 'security and trust' on the platform. Such policy issues also need to be designed and negotiated between the involved parties.

Another concern is related to the area of privacy. On the one hand, a trusted platform can gain confidence (and hence trust) of a challenging host it is communicating with by validating its state. But on the other hand, from the privacy point of view, it may not be appropriate for the challenging host to learn complete information about the components and state of the requestor's platform. At first glance, property based attestation may seem to abstract the low level implementation details to a higher level and thereby provide more privacy. However, deeper inspection will reveal that properties can be 'reverse engineered' to determine the implementation details of a platform. For example, certain components might have unique properties that may not be available in other components or a component can be challenged for different properties to work out what implementations are available and what is not. One recent proposal to enhance privacy in Property Based Attestation is based on the Zero Knowledge Protocol [16]. The protocol assumes that a property certificate is issued as a mapping between a state of a component and its properties. Using the zero knowledge protocol, a trusted platform proves to the challenger that there exists a valid link between the state measured by the Certificate Authority and the property requested without actually revealing the state information in the certificate.

In other words, the protocol allows for a simple equality check between the measured state of the platform and the certified state but hides the state values. If the equality check is successful, the verifier believes that the trusted platform has the certified properties. The problem however is, the certified state cannot be the overall state of the platform. This is because, we cannot guess all the possible

platform values to create property certificates that match up. On the other hand, the certified state cannot be a component's state as well. This is because an individual measurement of a component is invalid without the transitive chain of measurements. This in turn will require the log of measurements to be made available in order to verify the chain of trust. Providing the log will defeat the purpose of the zero knowledge protocol. Alternatively, a chain of property certificates can be verified to maintain the trust chain. Another simple approach that we have adopted for privacy in this paper is to slice the properties at different granularities. A trusted platform and a verifier can negotiate in the beginning as to what properties will be revealed and at what granularities. This could enable better privacy for the trusted platform.

Other issues that can be addressed by our architecture are primarily extensions to the current implementation. The example illustrated in section 8 is a sample scenario only. One can imagine that there are many design issues possible here. For example, When Brok-A invokes the PTS service of Hosp-B, both the parties can enter a negotiation phase to determine what properties will be disclosed, at what granularities (S1,S2 or S3) and what actions should be taken if the necessary properties are unavailable. Hosp-B can also have a trust management system of its own to determine what Brok-A should do with Hosp-B's property information e.g delete immediately after use etc. Similarly, Hosp-A can have an agreement with Brok-A on which of Hosp-A's policies will be disclosed to Brok-A, for which patients, under what circumstances and what Brok-A can do with those policies. We can therefore see that each of the entities can themselves have a trust management system on their own. We are considering such extensions to the overall architecture.

10 Conclusion

In this paper, we proposed an authorization architecture for distributed systems leveraging trusted computing platforms. We believe that unlike hash measurements, security properties provide a neat way of defining security policies for systems. We introduced the notion of a Property Manifest, similar to the Reference Manifest to represent security properties. Properties in Manifest were expressed at different steps of granularity S1,S2 etc. We then provided an overview of our system architecture and its components. Like in any distributed system, the authorization mechanism supported different strategies like push, pull and delegation. The system has been implemented using Web services with policy extensions for XACML. We concluded with some interesting thoughts that need to be considered while using property based authorization.

There are many avenues for future work. Presently, we are trying to understand the notion of a property and the issues associated with its evaluation and certification. This will enable us to specify an algebra for trusted platforms, their components and properties. Using this algebra, one might be able to reason out if a certain platform has the necessary 'privileges' to access a certain

resource. We will then try to combine user based authorization with platform based authorization to see how it impacts a security decision.

References

1. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Security and Privacy. (1996) 164–173 TY - CONF.
2. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The KeyNote Trust-Management System Version 2 (RFC 2704). Internet Engineering Task Force. (September 1999)
3. DeTreville, J.: Binder, a Logic-Based Security Language. In: SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2002) 105
4. Chu, Y.H., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M.: Referee: Trust Management for Web Applications. World Wide Web J. **2**(3) (1997) 127–139
5. Herzberg, A., Mass, Y., Michaeli, J., Ravid, Y., Naor, D.: Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In: SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (2000) 2
6. Trusted Computing Group: TCG TPM Main Specification Version 1.1b. (2005)
7. Poritz, J., Schunter, M., Herreweghen, E.V., Waidner, M.: Property Attestation:Scalable and Privacy-Friendly Security Assessment of Peer Computers. Technical report, IBM Research (May 2004)
8. Sadeghi, A.R., Stueble, C.: Property-Based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In: NSPW '04: Proceedings of the New Security Paradigm Workshop. (2004)
9. Nagarajan, A., Varadharajan, V., Hitchens, M.: Trust Management for Trusted Computing Platforms in Web Services. In: STC '07: Proceedings of the 2007 ACM workshop on Scalable Trusted Computing, New York, NY, USA (2007) 58–62
10. Yoshihama, S., Ebringer, T., Nakamura, M., Munetoh, S., Maruyama, H.: WS-Attestation: Efficient and Fine-Grained Remote Attestation on Web Services. Technical report, IBM Research (February 2005)
11. TCG Infrastructure Working Group: Core Integrity Schema Specification. (November 2006)
12. OASIS XACML Technical Committee: eXtensible Access Control Markup Language 3 (XACML) Version 2.0. (February 2005)
13. Hughes, J., Maler, E.: Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1. OASIS. (May 2004)
14. Kudo, M., Hada, S.: XML Document Security Based on Provisional Authorization. In: CCS '00: Proceedings of the 7th ACM conference on Computer and Communications Security, New York, NY, USA, ACM (2000) 87–96
15. OASIS XACML Technical Committee: Web Services Profile of XACML (WS-XACML) Version 1.0. (December 2006)
16. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.R., Stüble, C.: A Protocol for Property-Based Attestation. In: STC '06: Proceedings of the first ACM workshop on Scalable Trusted Computing, New York, NY, USA (2006) 7–16
17. Balacheff, B., Chen, L., Pearson, S., Plaquin, D., Proudler, G.: Trusted Computing Platforms - TCPA Technology in Context. Hewlett-Packard Books (2003)