Journal of Computer and System Sciences 80 (2014) 916–934

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Trust enhanced distributed authorisation for web services

Aarthi Nagarajan, Vijay Varadharajan *, Nathan Tarr

Information and Networked Systems Security Research, Macquarie University, Sydney, Australia

ARTICLE INFO

Article history:
Received 20 September 2012
Received in revised form 15 March 2013
Accepted 27 August 2013
Available online 11 February 2014

Keywords:
Trusted platforms
Distributed authorisation
Secure web services
Trust enhanced security

ABSTRACT

In this paper, we propose a trust enhanced distributed authorisation architecture (TEDA) that provides a holistic framework for authorisation taking into account the state of a user platform. The model encompasses the notions of 'hard' and 'soft' trust to determine whether a platform can be trusted for authorisation. We first explain the rationale for the overall model and then describe our hybrid model with 'hard' and 'soft' trust components, followed by a description of the system architecture. We then illustrate our implementation of the proposed architecture in the context of authorisation for web services. We discuss the results and demonstrate that such a trust enhanced approach could enable better authorisation decision making, especially in a distributed environment where user platforms are subject to dynamic security threats.

1. Introduction

Distributed systems have fundamentally changed the way individuals and enterprises share, process and store information today. Security issues, and authorisation in particular play a vital role in distributed systems, as greater availability and access to information in turn imply that there is a greater need to protect them. To address this issue, several access control mechanisms, languages and systems [1–7] have been proposed in the past. However, a majority of these systems have been designed to address authorisation requirements that relate to human users. Authentication systems based on passwords, identity certificates, role certificates or even bio-metric information like fingerprints and iris recognition have been developed. Issues of trust arise when one considers whether or how much to trust the certificates or credentials being provided. Trust management systems have evolved from distributed authentication and authorisation systems taking into account aspects such as the nature and the roles of the authorities involved, the types of credentials and attributes to be used in the verification process, the delegation and revocation of credentials involved and the roles played. However many of these secure systems (be they authorisation or authentication) make some basic assumptions about the state of the platform that is hosting and running the systems software and applications. There is an inherent trust that is placed on the underlying platform when an upper level user or an application is authenticated or authorised.

In the current networked world with heterogeneous platforms and numerous software applications and system software running on these platforms, it is important such underlying trust assumption about the system state be properly examined. There are several reasons for this. First, computing platforms have become very powerful and can run many applications together at the same time. In particular, as the number of software applications increases, greater the possibility for security vulnerabilities. These vulnerabilities in turn make the platform more vulnerable to attacks. Second, attacks themselves are becoming more and more sophisticated. Furthermore, attackers also have easier access to ready-made tools that enable

* Corresponding author.
E-mail address: vijay.varadharajan@mq.edu.au (V. Varadharajan).

http://dx.doi.org/10.1016/j.jcss.2014.02.002
0022-0000/

exploitation of platform vulnerabilities more effective. Third, platforms are being shared by multiple users and applications (belonging to different users) both simultaneously as well as at different times. Therefore there is a great chance of the platform being left in a vulnerable state as different users and applications run. Finally, because platforms have become much more complex today, users may themselves be unaware of their platform vulnerabilities. Let us assume for example, a user is authorised to download and execute a confidential file on his/her platform. A typical authentication and authorisation system only verifies if the user is authenticated and has the necessary rights to perform the action. As soon as the file is downloaded and executed, if there is malicious software in the platform (that is outside the knowledge of the user), it can make copies of the file and distribute it to others without the user's knowledge. Such a vulnerability renders the entire authorisation process useless. The challenge is then, given the dynamic nature of attacks, the complexity of software and hence the increased vulnerable nature of these platforms (hosting possibly malicious software), how can one reason about the identity, integrity and security of a platform in a distributed system environment.

This paper aims to exploit the features of trusted computing technology to enhance the design and enforcement of access control policies and mechanisms in a distributed environment. The main rationale behind trusted computing technology is to enhance the reasoning about the identity and state of a platform to determine whether it is in an acceptable state. To achieve this, it needs two features. A trusted platform has special processes that dynamically collect evidence of behaviour of different applications installed on a platform. This is then used to reason about the platform's overall state by comparing it to known or acceptable reference states to decide whether or not a platform can be trusted. Our approach is to use this reasoning to enhance the quality of the authorisation decisions by taking into account the state of the platform. For instance, now the authorisation decisions can be made not only by taking into account the identities of the users, the applications being used, the privileges of users and applications but also the state of the platform in terms of what types of applications and software components they have and their state. Then one can design policies such as restrict the type of communications or the type of activities involved in the interaction not only based on the traditional users and privileges but also based on the platform state. This approach can be thought of as a "trust enhanced authorisation" as we are making use of the notion of trust derived from the state of the platform. In general we refer to this as 'hard' trust as those aspects of the state which are derived from concrete security mechanisms such as verification of certificate and credentials. Then there is the other notion of 'soft' trust. The soft trust relates to aspects of trust derived from social mechanisms such as reputation. Recently, trust derived from social control mechanisms are increasingly becoming popular for secure authorisation systems. Trust here is determined from past experiences and behaviour associated with an entity. We believe a combination of hard and soft trust mechanisms to enhance the authorisation decision making could prove to be fruitful. In this paper we propose such a hybrid trust system which takes into account the hard aspects of trust such as the state of the platform (from the trusted computing technology measurements) as well as the behaviour of the platform based on past interactions (and its reputation). Then this combined hybrid trust is used to specify and reason about authorisation, thereby enhancing the quality of authorisation decision making. For example, more controls on authorisation credentials and tokens (hard trust) can be placed on platforms that are found to have behaved maliciously in the past (soft trust). This paper describes a combined hard and soft trust enhanced distributed authorisation system and architecture (TEDA).

A fundamental concept when it comes to determining the state of a platform is that of attestation. Section 2 provides background information on attestation and describes standard binary attestation and its extension to property based attestation and discusses how they are being used in trusted platforms. Section 3 describes the design issues in the development of a trust enhanced distributed authorisation architecture. This is the core contribution of this paper. It describes our trust model comprising both hard and soft trust aspects, the authorisation service and how trust model is being used to enhance the authorisation service. Section 4 describes an example scenario illustrating the operation of the developed system. We have developed a complete implementation of the system and carried out extensive analysis. Section 5 describes the implementation of the system. Section 6 provides some scenarios for implementation along with the results and Section 7 provides results for TEDA's temporal performance. Finally, Section 8 concludes the paper and briefly mentions some further work. We provide in Appendix A some operations of subjective logic that has been used in our trust model as described in Section 3.

2. Attestation

Attestation is a mechanism proposed by the Trusted Computing Group (TCG) that enables a trusted platform to disclose the state of its components to a third party. To enable in the process of attestation, all hardware and software components in the trusted platform are measured using hash values at the time of boot and measurements are stored securely to prevent modification. When a third party presents an attestation request, the trusted platform provides an attestation report in response. This response includes the measured hash values, and a set of expected hash values that are supported in the form of measurement certificates by trusted certifiers. The basic idea is that, a match between the measured and expected values usually indicates that the components are in a trusted state. This is referred to as binary attestation. Further details on binary attestation can be found in [8]. However, hash values have the disadvantage in that they change often even for trivial changes in the system. They are also cumbersome to use in policies as it is difficult to interpret hash values to meaningful system states.

Recently, many researchers have proposed that it is more useful to reason about the state of a platform based on the security properties of the platform rather than plain hash measurement values [9,10]; this form of attestation is called

property based attestation. Property based attestation leverages binary attestation to abstract low level binary hash values to high level security properties of platforms. Using property based attestation, it is possible to prove that the availability of a certain hash measurement guarantees the availability of a certain security property. A comparison of different property attestation mechanisms proposed in the recent times can be found in [11]. In this paper, we adopt the notion of certificate based property attestation, as this seems to be the most practical approach to realise attestation using properties. Here, a trusted certifier issues a property certificate vouching that a component satisfies a property if it measures up to an expected hash value. By verifying both the measurement certificates and property certificates for a given platform, a third party can infer the properties satisfied by the platform in a trustworthy manner.

3. Trust enhanced distributed architecture

Trusted computing platforms provide the ability to reason about the state of a platform using binary hash values. Property based attestation, on the other hand, enables the abstraction of binary hash values to meaningful properties of systems and system components. Binary attestation and property based attestation both lend to authorisation of platforms based on the hash values or properties that are satisfied by that platform. The basic idea of our trust enhanced distributed authorisation architecture (TEDA) is that a platform must satisfy a set of requirements such that it may be 'trusted' to perform a requested operation. For example, a platform may be checked if it complies with an enterprise's security requirements before it is trusted to access the company's network. There could be a range of security requirements, for instance, they can include the installation of necessary security software like antivirus and keeping them up to date.

TEDA uses a combination of hard and soft trust to determine the overall trust that may be placed on a platform. Here, hard trust refers to trust that is derived from traditional security mechanisms like credentials and certificates that are validated using concrete security functions and are characterised by certainty. Soft trust refers to trust that is derived from social control mechanisms and intangible information such as past experiences, reputation and co-operation. The main idea here is to allow the communicating parties to rate each other based on the outcome of one or more transactions and arrive at an aggregated soft trust score for a given party. This rating will then be used to assist in future decision-making processes, for example, whether the party may be trusted for similar transactions. This trust score is computed based on one's own experience about the other party in the past, as well as referrals or recommendations from others, or a combination of both. While hard trust mechanisms are quite rigid, they are usually based on one-time checks that once bypassed can make systems totally vulnerable. Soft trust mechanisms, on the other hand, are more persistent and are based on long term good behaviour. However soft trust is vulnerable to problems such as trust saturation; on the other hand, hard trust may not be responsive to dynamic changes. Hence we believe that hard trust combined with soft trust of a platform provides the best opportunity to determine the overall trust to be associated with the system; this in turn can then be used in the specification and enforcement of authorisation policies.

We can therefore express TEDA to consist of three main building blocks. First, it must consist of the hard trust block that verifies the necessary hard trust attributes (like certificates) as well as evaluation mechanisms to determine hard trust. Second, there must be a soft trust block that manages the soft trust attributes (like past experience records) to determine the soft trust of the platform. Finally, TEDA must also consist of the authorisation service block. This block performs the functions of authorisation by determining if a platform satisfies the necessary trust requirements by combining the hard and soft trust outcomes. It must in turn consist of an authorisation policy base to store the authorisation requirements (in the form of policy expressions) and an authorisation engine that performs the decision-making process. Below, we explain the function of each block.

3.1. Hard trust

Hard trust in TEDA is derived using the attestation verification mechanism. A platform is considered to satisfy the hard trust requirements if it satisfies the necessary hash measurements and property requirements for authorisation. A property may be defined as any attribute, characteristic, behaviour or function associated with a trusted platform. Examples of properties include security services like authentication, confidentiality and integrity offered by a platform along with the security mechanisms that may be used to implement these services, properties related to coding and implementation of the platform as well as generic properties of the platform such as configuration details. As the scope of the property can be broad, certification of properties can become a complex task. Hence it is inevitable that only certain properties will be verified and other properties will be derived from already verified properties. For example, if a platform has been certified to provide encryption, it might be possible to derive that the platform can support confidentiality property for data (using encryption).

A property may therefore be verified either directly using a certificate issued for that property or by verifying other properties required to satisfy the given property. In this vein, we have developed a simple logic language (called ALOPA) required to express property relationships. Using ALOPA, it is possible to determine if a platform satisfies a given property by extrapolating the available set of platform properties to the required set using ALOPA rules. Due to space restrictions, we will not describe the ALOPA language in detail. However we will give its important features required to describe our model and architecture, which is relevant for this paper. A full description of the ALOPA language can be found in [19]. ALOPA mainly consists of four predicates namely 'HasC', 'HasPF', 'SatC' and 'SatPF'. 'Has' in general defines a hierarchal relationship between

two entities where entities are platforms and platform components. HasC defines the relationship between two components and HasPF defines the relationship between a platform and a component. For example, $HasC(c_1, c_2)$ is read as component c_1 has (or contains) component c_2. $HasPF(pf_1, c_1)$ defines platform pf_1 contains component c_1. A 'Sat' predicate defines the relationship between an entity and the property it satisfies. For example, $SatC(c_1, p_1)$ is interpreted as component c_1 satisfies property p_1. $SatPF(pf_1, p_1)$ is read as platform pf_1 satisfies property p_1.

Using these four predicates, we now define three rules, the Component-Property (CP) rule, the Platform-Property (PP) rule and the Access Control rule. A CP rule is of the form:

$$SatC(c, p) \leftarrow L_1 \wedge L_2 \wedge \cdots \wedge L_n$$

where c is a component, p is a property, $n \geqslant 0$ and $L_1 \ldots L_n$ are either SatC or HasC literals. Such a rule indicates the abstraction of one property type to another. Examples (i) and (ii) below illustrate this. The rule may also be used to show that a component may satisfy one property if one of its sub-components satisfies the same (or a different) property. This is shown in example (iii) below.

(i) $SatC(c_1, p_1) \leftarrow SatC(c_1, p_2)$
(ii) $SatC(Browser, Trusted) \leftarrow SatC(Browser, Hash\_x)$
(iii) $SatC(c_1, p_1) \leftarrow SatC(c_2, p_2) \wedge HasC(c_1, c_2)$

First rule reads as: for a component c_1 to satisfy a property p_1, c_1 must satisfy a property p_2. For example, an application (c_1) that can perform an AES algorithm (p_2) can perform encryption (p_1), a browser (c_1) that can perform SSL (p_2) is expected to be secure (p_1), a web service (c_1) that can perform identify based authentication (p_2) can perform authentication (p_1). The second rule transforms the 160-bit binary measurement x of a browser component to a property (trusted) of the component. The third rule reads as: for a component c_1 to satisfy a property p_1, c_2 must satisfy p_2 and c_1 must contain c_2. For example, an application (c_1) can perform auditing operations (p_1) of all the inputs and outputs (t) if it (c_1) contains another sub-component (c_2) that records all audit logs (p_2). Similarly, a PP rule states:

$$SatPF(pf, p, t) \leftarrow L_1 \wedge L_2 \wedge \cdots \wedge L_n$$

This rule is used for the abstraction of properties of components to properties of platforms. An example of a very simple yet important PP rule is, if a component satisfies a property and if a platform has that component, then the platform is considered to satisfy that property. The following are examples of some PP rules.

(a) $SatPF(pf_1, p_2) \leftarrow SatC(c_2, p_2) \wedge HasPF(pf_1, c_2)$
(b) $SatPF(pf_1, p_1) \leftarrow (SatC(c_2, p_2) \wedge HasPF(pf_1, c_2)) \wedge SatPF(pf_1, p_3)$

Finally an access control rule states:

$$Do(pf, r, a, per) \leftarrow L_1 \wedge L_2 \wedge \cdots \wedge L_n$$

where $L_1 \ldots L_n$ are SatPF literals. An authorisation rule is used to specify the authorisation permissions for a platform based on the properties the platform satisfies. For example,

(a) $Do(pf_1, a_1, r_1, per_1) \leftarrow SatPF(pf_1, p_1) \wedge SatPF(pf_1, p_2)$

where rule (a) reads: a platform pf_1 has the permission per_1 to perform an action a_1 on the resource r_1 if the platform satisfies the property p_1 and property p_2. For instance, a platform (pf_1) is allowed (per_1) to connect (a_1) to the network (r_1) if it is both virus-free (p_1) and spyware-free (p_2).

An ALOPA policy set consists of a set of CP, PP and Access Control rules. Using these rules, an ALOPA authorisation engine is able to determine if a platform is allowed a set of permissions based on the properties it satisfies. For example, let us consider the following ALOPA policy set:

1. $Do(AIK_1, Network, Connect, Allow) \leftarrow SatPF(AIK_1, Trusted\_True)$
2. $SatPF(AIK_1, Trusted\_True) \leftarrow SatPF(AIK_1, Known\_True) \wedge SatPF(AIK_1, Secure\_True)$
3. $Sat(AIK_1, Known\_True) \leftarrow SatPF(AIK_1, IP\_192.70.4.6)$
4. $SatPF(AIK_1, Secure\_True) \leftarrow SatPF(Firewall, Enabled\_True) \wedge SatPF(Antivirus, Enabled\_True) \wedge SatPF(Antivirus, Uptodate\_True) \wedge SatPF(Spyware, Enabled\_True) \wedge SatPF(Windows, Uptodate\_True)$

Here, the platform identified by key AIK_1 is allowed to connect to the network if it is trusted. In order for it to be trusted, it must be a known system and must be secure. A system is known if it has the IP 192.70.4.6. A system is secure if it has firewall enabled, if the antivirus is enabled and up to date, if the anti-spyware is enabled and if Windows is up to date.

3.2. Soft trust

In the last section, we discussed how hard trust is derived from property based attestation while trust in property-based attestation is itself based on trust in binary attestation. A fundamental question that arises then is – given that the process of property-based attestation is significantly different from binary attestation and that the chain of trust is extended from binary attestation to property attestation mechanisms (e.g. to certification authorities that certify properties), how much can an Attestation Requester (AR) trust the properties that are presented by an Attesting Party (AP)? In other words, when AP reports its system state with a set of properties to AR, how certain can AR be that these reported values are true and that AP actually satisfies these claims?

We believe that the properties derived using property based attestation mechanisms must be used with caution and we attribute two reasons to this. First, all attestation measurements are (usually) taken at the time of boot and not at the time of attestation report. This design admits the potential for time-of-check time-of-use vulnerabilities: measurements reflect the state of the platform when it was measured, not when the attestation report is generated. Second, attestation relies heavily on measurement and property certifiers which are trusted third parties. As in any system this involves third-party certification authorities; trust in the property certification authority (CA) can be subjective and more importantly can vary depending on the context. The trustworthiness of the properties depends on the trustworthiness of the CA and verification mechanisms used by the CA to evaluate a component for that property. Therefore, trust in property attestation and the properties certified are directly dependent on AR's (attestation requester) trust in the CA that certifies that property. In our architecture TEDA, we take these uncertainties into consideration and design a dynamic Trust Model (*TM*) for property-based attestation. The main aim of the model is to provide a way of determining whether a platform can be trusted to satisfy a property given these uncertainties. Below, we provide a brief description of *TM* and a full description of the model can be found in [20].

The dynamic trust model *TM* uses subjective logic to model trust and uncertainties. Subjective logic proposed by Jøsang [12] is a logic that can be used to model trust that includes uncertain outcomes. In this logic, trust is represented using an opinion metric which is denoted as ω where $\omega = (b, d, u)$ and b, d and u are belief, disbelief and uncertainty respectively. Values of $b, d, u \in [0, 1]$ and $b + d + u = 1$. Some operations in this logic that are relevant to this paper are given in Appendix A. We refer the reader to [12] for more details on subjective logic. We describe the trust model *TM* for any Trusted Platform *TP* as $TM = (E, TR, OP)$. E represents the set of entities that share one or more trust relationships, *TR* is the set of trust relationships between the entities and *OP* is the set of trust operations for the management of these trust relationships.

Entities. Entities E include the different members of the trust model that share one or more trust relationships with each other. These are the Attestation Requester (AR), Attesting Party (AP) and the Property Certifying Authority (CA) entities.

Trust relationship. *TR* defines the trust relationship that is shared between two entities for a given property. Trust Relationship *TR* is defined by the following 10-tuple:

$$TR = (A, B, C, P, K, \Theta, M, pos, neg, unc)$$

The tuple states that an entity A trusts an entity B for a component C to satisfy the property P with trust class K at a given time Θ with experience held in pos, neg, unc and opinion held in M. Entities A and $B \in E$; C is a member of $\{C\}$, a finite set of all components in B; P is a member of $\{P\}$, a finite set of all properties satisfied by C; K is a member of {satisfaction, certification} trust classes (satisfaction trust is the belief that a component in AP will satisfy a given property and certification trust is the belief in the honesty and competence of a CA to certify a property for a component); Θ is the time at which experience values pos, neg, unc were last updated for this trust relationship, i.e. last update for this $\{TR\}$ occurred at time Θ; M is the evidence to opinion mapping operation on this trust relationship given in Appendix A. pos, neg, unc represent the total number of positive, negative and uncertain experiences respectively associated with this trust relationship.

For example, let us assume that the following *TR* is available: $TR = (PF_x, PF_y, Foo.exe, No\text{-}Malware, Satisfaction, Aug$ 01 2011 13:00:00, [0.666, 0.133, 0.2], 10, 2, 3). This entry represents a trust relationship that is shared between a platform PF_x and a trusted platform PF_y. PF_x has satisfaction trust (trust category) in platform PF_y that PF_y's component Foo.exe satisfies the property No-Malware. Trust is given by the opinion [0.666, 0.1333, 0.2] representing belief, disbelief and uncertainty respectively. This opinion is evaluated based on PF_x's past experience with PF_y where the given property was satisfied 10 times in the past (positive), not satisfied twice (negative) and the outcome was indeterminable three times (uncertain). Experience was last updated on Aug 1, 2011 at 13:00 hrs.

Trust operations. Trust operations in *TM* are used for the management of trust relationships. These include the evidence collection, trust evaluation and trust comparison operations as described below.

(i) *Evidence collection*. Evidence collection involves the process of collecting evidences required for trust evaluation. Evidence is obtained either directly using one's own experiences termed direct experience or using referrals from third parties known as recommended experiences. An experience is classified as positive, negative or uncertain. In general, if a platform satisfies

a property that it claims to have, then a positive experience is recorded, if a platform does not satisfy the property, a negative experience is recorded and an uncertain experience is recorded if AR is not able to determine if a platform satisfies a given property or not.

However, in this model we realise that when a property is not satisfied, a negative experience is not only associated with the property but may also be associated with the certification authority (for possibly not certifying the property correctly) and/or the hash value of the property (because the hash value changed after the system was certified for the property). To determine what negative experiences must be recorded, we in turn use the history associated with the previous experiences of the CA and hash outcomes. For example, if a property is not satisfied by a system and the CA that certified the property for the system has a very poor record of certifying this property in the past, then we may associate a negative experience with both the property and the CA. This process of evidence collection has been described in full detail in [20]. It provides a tabulation of how experiences are accumulated for the three conditions: (a) when a property is satisfied, (b) when a property is not satisfied and (c) when it is uncertain if a property is satisfied or not. Experiences for the later two conditions are recorded based on the history of the CA and hash verification outcomes as discussed above. Apart from direct experiences, referrals are obtained from other AR platforms that have also interacted with the AP platform in the past. These recommendations as we call them are also in the form of positive, negative and uncertain experiences.

Definition 3.1 *(Trust decay).* Trust is dynamic in nature and tends to change with time. Over a given timeframe, the value of trust at the beginning of the period is different from the value at the end, even when there are no underlying factors that affect the value of trust directly. Just as the humans tend to forget things or attach less importance to events that have occurred in the past, we model systems also to attach less importance to events that have occurred in the past compared to more recent events. In other words, the system is modelled to gradually become non-decisive about trust (and distrust) as time progresses. The decay operator is a function that is used to represent this nature of trust. Eq. (1) shows the decay function $\Psi_{k,\Delta}$ used to calculate new opinion ω_{new} from an old opinion ω_{old}.

$$\omega_{new} = \Psi_{k,\Delta}[\omega_{old}] \tag{1}$$

where $\Psi_{k,\Delta}$ is given as

$$b_{new} = b_{old}\left[1 - e^{-(k.\Delta)}\right], \qquad d_{new} = d_{old}\left[1 - e^{-(k.\Delta)}\right]$$
$$u_{new} = u_{old} + \left[(b_{old} + d_{old}) - (b_{new} + d_{new})\right]$$

k is rate of decay and its value is fixed as $0 < k \leqslant 1$. For example, if the rate of decay is 1%, $k = 0.01$, if rate of decay is 10%, $k = 0.1$ and if rate of decay is 100%, $k = 1$. Δ is the difference between the current time θ at which trust is evaluated and the time at which opinion for that property was last updated. The value of Δ is chosen such that ω_{new} does not decay rapidly. For example, if $\Delta = 0$, then ω_{new} is same as ω_{old} and if $\Delta = \infty$ then ω_{new} tends to zero. In our model, we chose Δ as the number of years that have elapsed since the opinion was last updated. The minimum value of Δ is 0/365 (zero days) and the maximum value is 730/365 (2 years approximately). At $\Delta = 2$ and $k = 1$, b_{new} is 13 per cent of b_{old}. This is the maximum decay possible for any opinion. Any time greater than 2 years is also assumed to be equal to 2 years such that b_{new} decays at 100% decay rate to a maximum of $0.13(b_{old})$ and not more.

(ii) *Trust evaluation*. Let $({}^{A}pos_{B,sat(c_i,p_j)}, {}^{A}neg_{B,sat(c_i,p_j)}, {}^{A}unc_{B,sat(c_i,p_j)})$ represent the evidence associated with a *TR* of platform A about platform B for the satisfaction of a property p_i of a component c_i. Based on the evidence collected, the evidence mapping function M is used to calculate the opinion for this *TR*. This defines the opinion of platform A about platform B for the satisfaction of property p_i of component c_i.

$${}^{A}\omega_{B,sat(c_i,p_j)} = \left\{{}^{A}b_{B,sat(c_i,p_j)}, {}^{A}d_{B,sat(c_i,p_j)}, {}^{A}u_{B,sat(c_i,p_j)}\right\}$$

Similarly, the certification trust of platform A about Certification Authority CA for the certification of the property p_i of a component c_i is given by

$${}^{A}\omega_{CA,cer(c_i,p_j)} = \left\{{}^{A}b_{CA,cer(c_i,p_j)}, {}^{A}d_{CA,cer(c_i,p_j)}, {}^{A}u_{CA,cer(c_i,p_j)}\right\}$$

The evidence collected is now used to calculate direct, recommended and derived trusts.

Definition 3.2 *(Direct trust).* Direct trust is the belief one entity holds on another entity for a given context, based on its own past experiences with that entity. The direct trust in a component c_i and a property p_j is given by Eq. (2).

$${}^{A\text{-}dir}\omega_{B,c_i,p_j} = \Psi_{k,\Delta}{}^{A}\omega_{B,sat(c_i,p_j)} \odot \Psi_{k,\Delta}{}^{A}\omega_{CA,cer(c_i,p_j)} \tag{2}$$

where ${}^{A}\omega_{B,sat(c_i,p_j)}$ and ${}^{A}\omega_{CA,cer(c_i,p_j)}$ are obtained used the evidence to opinion mapping operator defined in Appendix A.

Definition 3.3 *(Recommended trust).* Recommended trust is the belief one entity holds on another entity for a given context, based on the recommendations obtained from its peer entities' past experiences. ${}^{A\text{-}rec}\omega_{B,c_i,p_j}$ represents the overall recommended opinion of A on B computed from the individual opinions of A's recommenders and it is given by Eq. (3).

$$ {}^{A\text{-}rec}\omega_{B,c_i,p_j} = \left(I_{R_1} \otimes \Psi_{k,\Delta}\left[{}^{R_1}\omega_{B,sat(c_i,p_j)}\right]\right) \oplus \cdots \oplus \left(I_{R_m} \otimes \Psi_{k,\Delta}\left[{}^{R_m}\omega_{B,sat(c_i,p_j)}\right]\right) \tag{3} $$

where $I_{R_1} \ldots I_{R_m}$ are importance factors that determine how much platform A values each recommender. $I_R = (wt, 1 - wt, 0)$ where wt denotes the weight for a recommender R. The sum of the weights of all recommenders equals 1.

Definition 3.4 *(Derived trust).* Derived trust is the belief one entity builds on another entity for a given context, based on other atomic trust relationships such as direct trust and recommended trust. Derived opinion for a property p_j of component c_i is computed by combining the direct and recommended opinions for that property.

$$ {}^{A\text{-}der}\omega_{B,c_i,p_j} = {}^{A\text{-}dir}\omega_{B,c_i,p_j} \oplus {}^{A\text{-}rec}\omega_{B,c_i,p_j} \tag{4} $$

Definition 3.5 *(Derived platform trust).* Derived platform trust is defined as the belief one platform builds on another platform for a given context based on the combined belief of all the individual properties of that platform. Platform trust of A about B is computed by combining the derived opinions of all the properties in platform B.

$$ {}^{A\text{-}der}\omega_B = I_{c_i,p_j} \otimes \left[{}^{A\text{-}der}\omega_{B,c_i,p_j}\right] \oplus I_{c_k,p_l} \otimes \left[{}^{A\text{-}der}\omega_{B,c_k,p_l}\right] \tag{5} $$

Assuming that platform B has two properties – property p_i of component c_j and property p_l of component c_k where $p_i, p_l \in P$ and $c_j, c_k \in C$ – the derived platform opinion ${}^{A\text{-}der}\omega_B$ equals the consensus of the derived opinions ${}^{A\text{-}der}\omega_{B,c_i,p_j}$ and ${}^{A\text{-}der}\omega_{B,c_k,p_l}$ on properties p_j and p_l respectively. The opinions are decayed using the decay operator given in Eq. (1). An importance factor I is attached to each opinion. This importance factor determines how much A values each property to contribute to the overall trust of the platform. The sum of the importance factors is equal to 1 and $I = (wt, 1 - wt, 0)$ where wt denotes the weight for each property.

(iii) *Trust comparison*. Given two opinions ω_1 and ω_2, we define an opinion comparison operator $\geqslant_\omega$, whereby $\omega_1 \geqslant_\omega \omega_2$ holds if $b_1 > b_2$, $d_1 < d_2$ and $u_1 < u_2$. In such cases, we say that ω_1 is greater than the threshold presented by ω_2.

An overall picture of the authorisation process using soft trust alone can be given as follows. Initially, platform A computes its direct trust for a given property of platform B using Eq. (2). A seeks out recommenders and gathers their recommendations for the satisfaction of the given property. Each recommended opinion is decayed for the time elapsed since the last recommendation was recorded. The decay time Δ is different for each recommender depending on when the recommender last interacted with platform B. It is also possible for platform A to define a different decay rate k for each recommender. The final recommended trust is then calculated using Eq. (3). Then the direct and recommended trusts are combined together using Eq. (4) to compute the derived trust. For every service that is provided, A defines authorisation policies that include a threshold value ω_{th} as an opinion constant for each property. A compares the derived opinion and the threshold opinion using the comparison operator $\geqslant_\omega$ to determine whether derived opinion is greater than the threshold opinion. If the derived opinion is higher, then A believes that it trusts platform B for that property and services its request.

3.3. Authorisation service

The Authorisation Service (AS) block is the core of the TEDA model and is responsible for performing the platform authorisation functions on behalf of the Attestation Requester (AR). It liaises between the AP and AR domains by accepting requests from AP and determining if AR trusts AP enough to permit the request. It consists of two sub-components namely, the Authorisation Policy Base (APB) and the Authorisation Engine (AE). APB stores policies corresponding to each request and defines the trust conditions under which a request will be permitted. AE is the decision making authority. It compares the trust conditions defined in the policies against the trust derived for a given platform using the hard trust and soft trust services. If the required conditions are met and the platform is trusted for the specific request and the request is honoured.

Authorisation policies. Authorisation policies in TEDA define the platform attributes that are necessary for a request to be authorised. The platform attributes of a requester include the hard trust attributes in terms of the properties that must be satisfied, soft trust attributes in terms of trust thresholds as well as some other attributes such as time and location, which we refer to as dynamic attributes. Authorisation policies in TEDA consist of one or more rules which define the authorisation requirements of the Attesting Requester AR (for the Attesting Party). These requirements must include the following elements.

(i) The identity of the platform from which a request is presented. Trusted Computing Group (TCG) has standardised the use of an Attestation Identity Key (AIK) [8], which is a public key certified by a trusted certification authority as the identity certificate of a trusted platform. By including the identities of the requester platforms in the target, AR can define policies specific to these platforms.

(ii) The resource for which access has been requested. Access to a resource is available only after a requester proves that it has the necessary privileges to access this resource. A resource may be a program, application, file record, web service or even a business function. By including the resource object in the target element, the service provider can define policies specifically to protect access to those resources. Along with the resource, operations associated with the resource may also be included in the target. For example, a policy can be defined to protect an object_foo. The same policy can also include different rules for different operations on object_foo.

(iii) The policy must enable the inclusion of hard trust, soft trust and dynamic attributes as authorisation requirements. Hard trust attributes define the property requirements of the attesting party's platform. It includes the list of properties that the attestation requester expects the attesting platform to satisfy in order to service a request. Soft trust attributes define the soft trust requirements of the attesting platform. These will include the provision to define trust thresholds as required. Policies can be defined in terms of thresholds for direct trust, recommended trust, derived trust and/or platform derived trust. The default option is to include a derived trust threshold (Eq. (4)) for a required property or a conjunction of thresholds of two or more properties. Dynamic attributes are those attributes that change often and may not be vouched for in the form of certificates or credentials. These include requirements such as date and time of request and location from which the request is being made. For example, a policy to perform operation $op1$ on the object_foo during the day can be different from a policy that permits the operation $op1$ on object_foo at night.

4. Example scenario

This section illustrates the working of the proposed TEDA model in the context of a simple scenario involving a social networking system. In such a system, a user is connected to a number of friends and is able to share information such as photographs and messages with them. There are also third-party applications that may be installed on a user's portal for sharing music files, videos and power point presentations with these friends. Recently, these third-party applications have become an issue of major concern for many users. Many of these applications have been known to act maliciously by stealing user identities, birth dates, contact details and other private information corresponding not only to the user but also to all the contacts associated with that user. Users are therefore required to make judgements about the legitimacy of an application before installing it onto their portal.

In this scenario, let us assume that A would like to install an application 'Application1' provided by platform P. A is only willing to install the application if it can be 'trusted' and if the derived trust for this application is greater than a threshold trust of $(0.3, 0.4, 0.3)$. Past experience record is available about the application and some recommendations are also gathered from friends who have installed the application in the past. The decision has to be made on December 26 2009 at 14:00 hrs. The decay rate for this trust is set at 1.0. The following trust relationships are available in the experience base.

1. $(A, P, Application1, Trusted\_True, Sat, Oct\ 01\ 2009, 14:00, [0.789, 0.105, 0.105], 15, 2, 2)$
2. $(A, P, Application1, Trusted\_True, Sat, Nov\ 12\ 2009, 14:00, [1, 0, 0], 1, 0, 0)$
3. $(A, CA, Application1, Trusted\_True, Cert, Oct\ 01\ 2009, 14:00, [0.923, 0, 0.77], 12, 0, 1)$
4. $(R1, P, Application1, Trusted\_True, Sat, Oct\ 31\ 2009, 14:00, [0.166, 0.833, 0], 3, 15, 0)$
5. $(R2, P, Application1, Trusted\_True, Sat, Oct\ 03\ 2009, 14:00, [0.1, 0.90, 0], 2, 20, 0)$

Fig. 1 shows the two domains, the Attesting Party (AP) domain which is that of the service provider X, and the Attesting Requester (AR) domain which is that of the user within the social portal. It is assumed that the social portal is able to perform the functions of TEDA either as an integrated service or by using a plug-in that may be installed onto the user's portal. The following steps are carried out in this system architecture. In Step 1, the user A learns of the application $Application1$ provided by X while he is logged on in his portal. A collects the required details of $Application1$ such as its attestation report and pointers to property certificates. In Step 2, Authorisation Service (AS) is requested to determine if $Application1$ may be installed along with the information collected from the application provider X. In Step 3, AS fetches the required authorisation policy. It then collects the required dynamic attributes. For example, A may have a policy indicating that it will accept applications from service providers who are local or from certain trusted countries. In such cases, the location information will be collected from the attribute provider. In Step 4, AS then performs the validation of attestation report and property certificates. ALOPA Service determines if the required properties are satisfied using available ALOPA policies. For example, let us assume that there exists an ALOPA policy that defines the following rules for the property 'Trusted_True': (1) Application1 is considered to satisfy 'Trusted_True' property if it has the hash value 1 and if Application1 satisfies the property 'Safe_True'. Application1 satisfies property 'Safe_True' if it is free from malware and satisfies the property 'Malware_False' vouched for in a property certificate. If this ALOPA policy is satisfied, then ALOPA Service returns the message that the property 'Trusted_True' is satisfied to the AS. In Step 5, AS proceeds to determine trust status of the application. In Step 6, Trust Service collects the necessary recommendations from the user A's recommenders and computes the derived opinion on $Application1$ as $(0.32, 0.39, 0.27)$. In Step 7, Trust Service lets AS know that the computed value satisfies the required threshold of $(0.3, 0.4, 0.3)$. In Step 8, AS informs this to the user A. In Step 9, User A takes the necessary action based on this and completes the installation process.

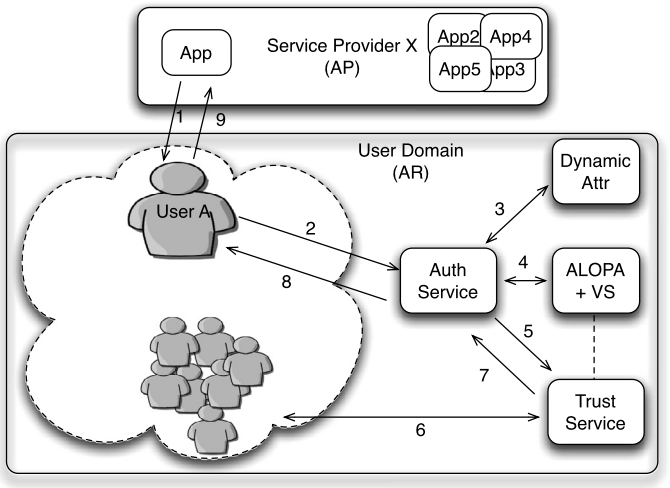

Fig. 1. Example application.

5. Implementation details

We have done a complete implementation of the TEDA model and architecture. The model has been implemented in the context of authorisation for web services in a service oriented architectural framework. Web services adopt a Service Oriented Approach (SOA) to providing loosely coupled services over the Internet. It provides standard protocols and messaging formats for organisations to publish and utilise business functions as services. The decision to use web services for our implementation is twofold: first, web services have become a popular choice today for deploying services and business processes over the Internet, and second, the Trusted Computing Group (TCG) has developed some standard specifications for trusted computing supporting web services. In this implementation, all components including the interfaces between the components, the policy base, authorisation policies, trust base, hosted services and communication protocols have been implemented to comply with the web service standards. TCG standards for attestation report [13] and measurement certificates [14] have been implemented to enable attestation report and verification functions. The implementation setup is given in Fig. 2 and consists of the following components.

5.1. Attesting platform

The Attesting Party AP is implemented using an Infineon TPM 1.2 chip physically embedded onto AP's motherboard. Software support for the TPM is provided by JTSS [15,16], which is a Java based implementation of the Trusted Software Stack (TSS). JTSS enables the communication of applications in AP with the TPM. Attestation measurement and reporting is supported by the Integrity Measurement Architecture (IMA) [17], a Linux-based software developed by IBM. Structures for property certificates and property reports have been designed in line with the TCG XML schemas for measurement certificates [14] and attestation report [13]. Netbeans 6.7 beta is used as the development environment and apache tomcat web server is used to host SOAP/Java-based web services in AP. Finally, the implementation uses IAIK/OpenTC Java TcCert [15] package to create TCG certificates such as AIK certificate according to the Open TC specification [18].

5.2. Attestation requester

The Attestation Requester AR is implemented using a Windows web server. Dot Net framework 2.0 is used as the development environment and services are hosted using the Internet Information Services (IIS) 6.0 web server. Databases of the Authorisation Service and Trust Service components have been implemented using Microsoft SQL Server 2000. The ALOPA engine is implemented as a Dynamic Link Library (DLL). A Prolog interpreter known as C#Prolog developed in C# is integrated in the DLL assembly. The syntax of the Prolog language allows the specification of ALOPA facts and rules using predicates as Prolog programs. The ALOPA engine uses Selective Linear Definite (SLD) clause with backtracking for resolution. At the time of policy resolution, the ALOPA engine presents a query to the Prolog engine which responds with a positive or negative outcome based on the results of the evaluation. Finally, the XACML policy schema is extended to support authorisation policies for property attestation in TEDA for web services. The extended policy accommodates for the identity of the authorised platform along with the hard trust, soft trust and dynamic attribute elements. Subsequently, XACML request/response schemas have also been extended for supporting the authorisation request and authorisation response messages. Figs. 3 and 4 show the authorisation policy in XACML and ALOPA policy respectively for the example scenario discussed in Section 4.

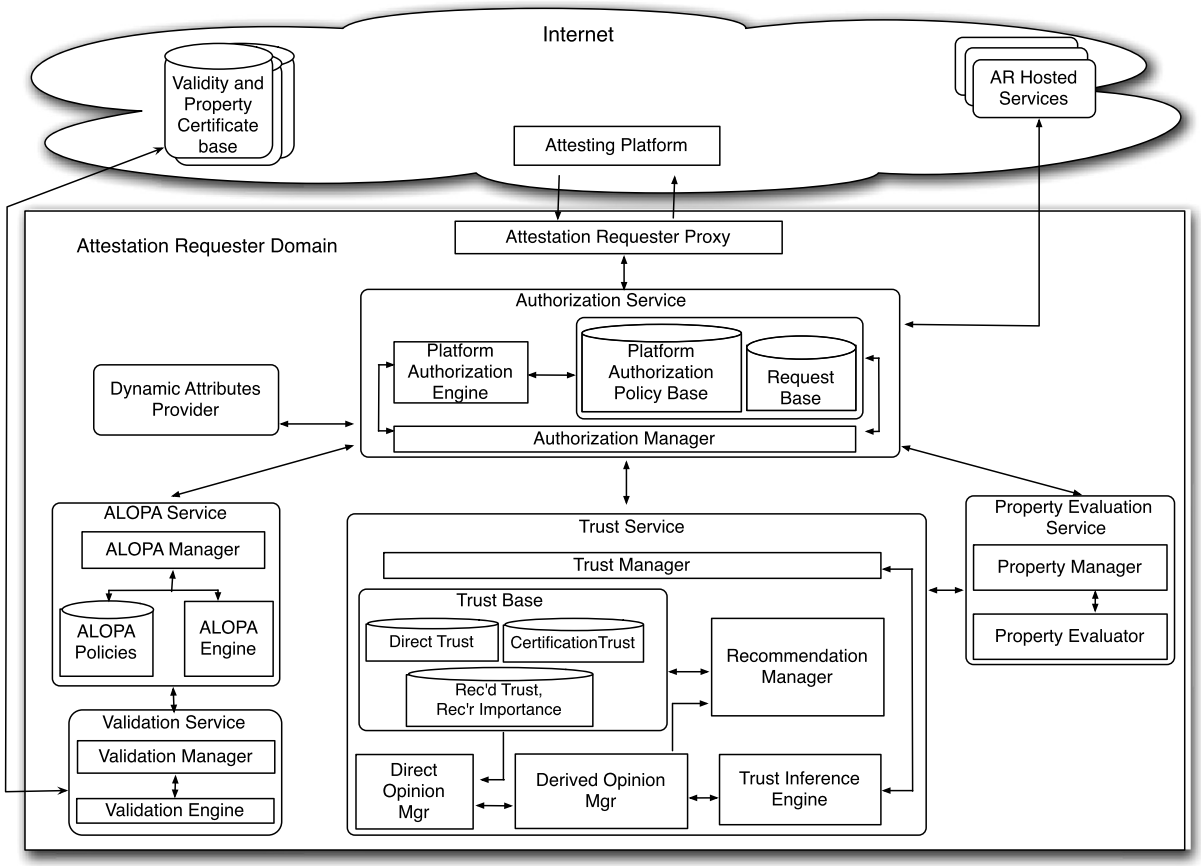

Fig. 2. Trust-enhanced distributed authorisation architecture.

The implementation architecture is shown in Fig. 2. The architecture consists of the AP and AR domains. The AP domain consists of the trusted platform that presents an authorisation request to AR. The AR domain consists of the three main services as defined in Section 3. There is the Authorisation Service (AS) with the policy base and the authorisation engine. The request base is added to store the details of all requests temporarily until a decision is made. The authorisation manager is responsible for liaising between AS and other service managers in the AR domain. Then, there is the ALOPA service that includes the ALOPA policy base and the ALOPA policy engine. ALOPA utilises the Validation Service (VS) to perform the attestation verification function and to generate the property report. A property report is essentially a list of all the properties that are satisfied by AR based on property certificate verification.

The Trust Service (TS) consists of the trust base with the direct trust, recommended trust and certificate trust relationships. The direct opinion manager and the recommendation manager are responsible for computing the direct and recommended trusts respectively. The derived opinion manager then computes the final derived opinion using the direct and recommended trusts. The Trust Inference Engine (TIE) determines if the computed trust values match the expected threshold and conveys the result to the Authorisation Manager. In addition to the ALOPA and Trust services, the model also benefits from a Dynamic Attribute Provider (DAP) that collates the required dynamic attributes for the request and a Property Evaluator Service (PES) that facilitates the evidence collection process. Evidence collection was discussed in Section 3.2. We use the property evaluator component to determine if a property was satisfied, not satisfied or if the result was indeterminable to record a positive, negative or uncertain experience respectively. A full description of the TEDA architecture can be found in [20].

Fig. 5 shows the class diagram for AR. This diagram includes classes that have been implemented for all the different service components of AR. These include the Authorisation Service, ALOPA Service, Trust Service, Property Evaluation Service and Dynamic Attributes Service. Each service class implements a set of functions that enable the functioning of that service. Implementation details of each of the classes and functions can be found in [20].

Fig. 6 below provides the overall authorisation mechanism. It shows the message exchanges between AP and AR and between the various services of AR when a request is received. It is assumed that Attesting Platform (AP) is the service requester and Attestation Requester (AR) is the service provider. When AP presents a service request to AR, AR verifies whether ALOPA attributes as hard trust, soft trust and dynamic attribute requirements for the requested service have been satisfied.

```
<?xml version="1.0" encoding="utf-8"?>
<Policy PolicyId="http://policyid.1" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides"
 xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os access_control_xacml-2.0-policy-schema-os_property.xsd">
 <Description>Platform may perform the specified action on the service if it meets the requirements of the rule</Description>
 <Target>
  <Platforms>
   <AnyPlatform />
  </Platforms>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-equal">
     <AttributeValue DataType="xs:anyURI">Application1</AttributeValue>
     <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="xs:anyURI" />
    </ResourceMatch>
   </Resource>
  </Resources>
  <Actions>
   <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">InstallApp</AttributeValue>
     <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ActionMatch>
   </Action>
  </Actions>
 </Target>
 <Rule RuleId="http://ruleid.1" Effect="Permit">
  <Target />
  <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <ALOPA_Attributes>
     <ALOPA_Attribute>
      <Entity>
       <ComponentID DataType="http://www.w3.org/2001/XMLSchema#string">Application1</ComponentID>
      </Entity>
      <PropertyName DataType="http://www.w3.org/2001/XMLSchema#string">Trusted</PropertyName>
      <PropertyValue DataType="http://www.w3.org/2001/XMLSchema#string">True</PropertyValue>
     </ALOPA_Attribute>
    </ALOPA_Attributes>
    <Trust_Attributes>
     <DecayRate DataType="http://www.w3.org/2001/XMLSchema#double">1.0</DecayRate>
     <Trust_Attribute>
      <Entity>
       <ComponentID DataType="http://www.w3.org/2001/XMLSchema#string">Application1</ComponentID>
      </Entity>
      <PropertyName DataType="http://www.w3.org/2001/XMLSchema#string">Trusted</PropertyName>
      <PropertyValue DataType="http://www.w3.org/2001/XMLSchema#string">True</PropertyValue>
      <ThresholdBelief DataType="http://www.w3.org/2001/XMLSchema#double">0.3</ThresholdBelief>
      <ThresholdDisbelief DataType="http://www.w3.org/2001/XMLSchema#double">0.4</ThresholdDisbelief>
      <ThresholdUncertainty DataType="http://www.w3.org/2001/XMLSchema#double">0.3</ThresholdUncertainty>
     </Trust_Attribute>
    </Trust_Attributes>
   </Apply>
  </Condition>
 </Rule>
</Policy>
```

Fig. 3. Example of an authorisation policy.

```
'SatC'('Application1', 'Trusted_True') :-
        'SatC'('Application1', 'Hash_1'),
        'SatC'('Application1', 'Safe_True').
'SatC'('Application1', 'Safe_True') :-
        'SatC'('Application1', 'Malware_False').
```

Fig. 4. Example of an ALOPA policy.

As shown in Fig. 6, the sequence of messages starts soon after the Attesting Platform (AP) presents a service request to the Attestation Requester. When a hosted service is invoked, the service request is redirected to the AR-proxy. All service requests to AR will be forwarded to the AR-Proxy by default. AR-Proxy forwards this request to the Authorisation Service. The Authorisation Service looks for the policy corresponding to the request. It checks whether the required credentials, such as the attestation report, pointers to measurement certificate repository and pointers to property certificate repository, have been provided with the request. If the necessary platform credentials are not provided, it requests the AR-Proxy to obtain the

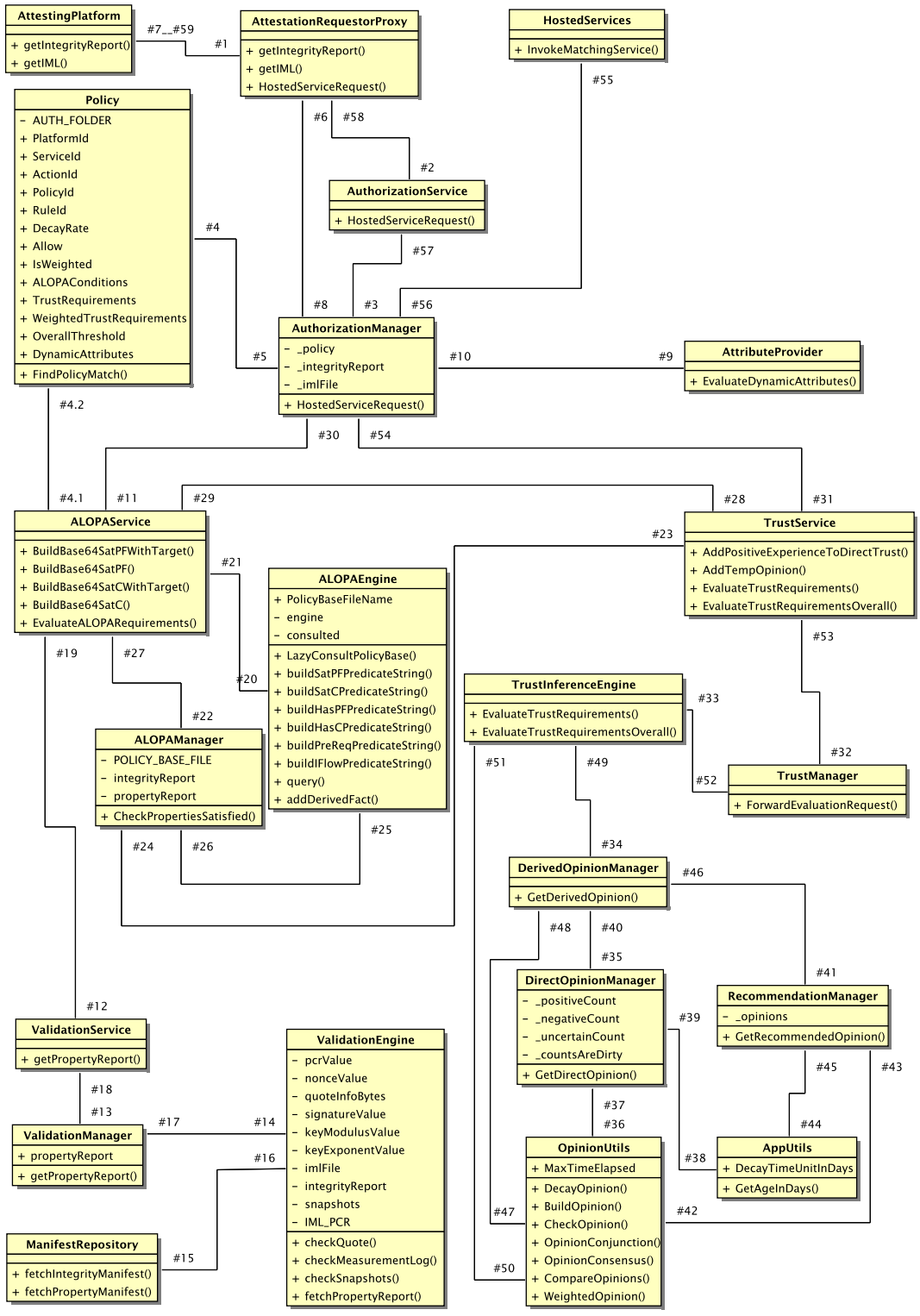

Fig. 5. Attestation requester – overall class diagram.

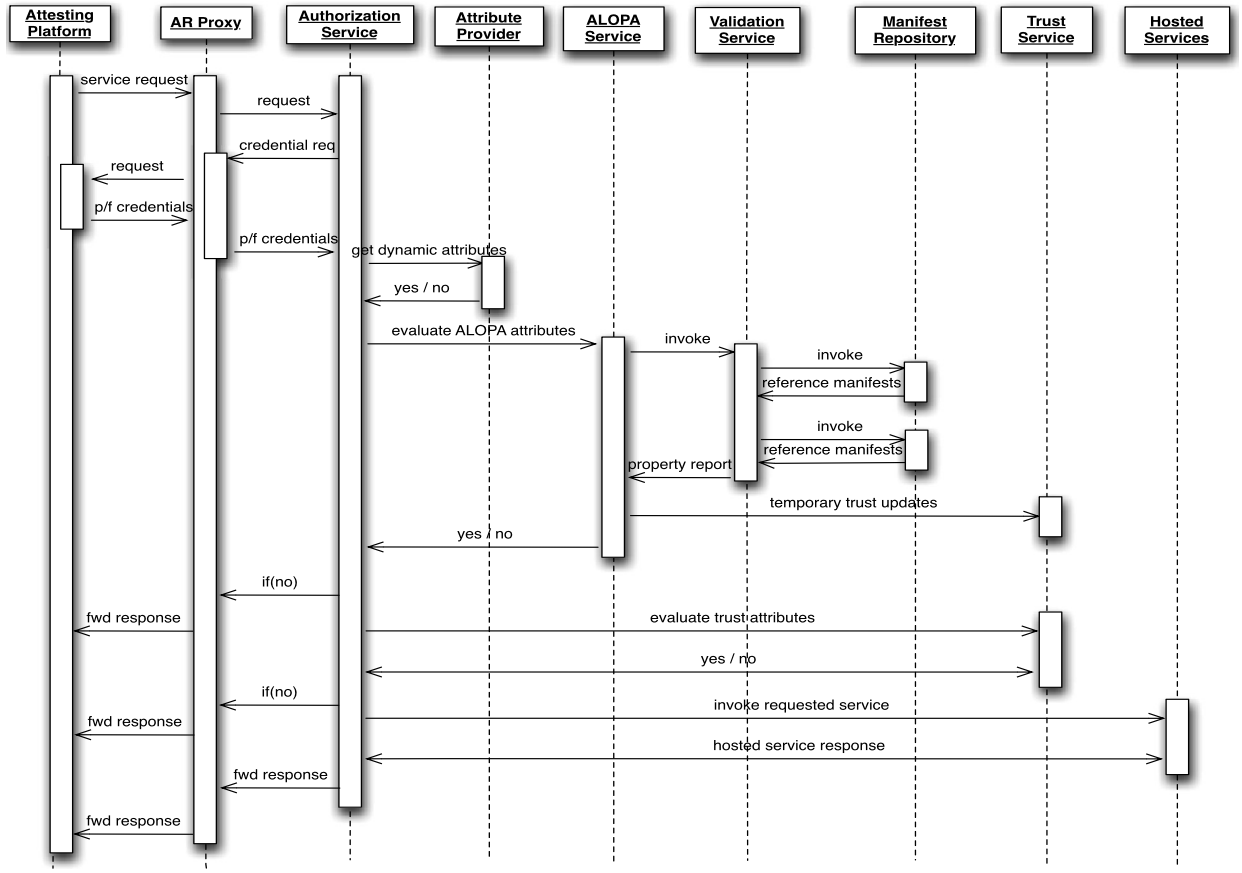

Fig. 6. TEDA – sequence diagram.

credentials. AR-Proxy receives the request and forwards it to the AP. It waits for AP to provide the credentials and forwards them back to the AR-Proxy. The AR-Proxy forwards these credentials to the Authorisation Service. The Authorisation Service requests the Attribute Provider to collect the required dynamic attributes. It receives the list of attributes as a response from the Attribute Provider. Authorisation Service next presents a request to the ALOPA Service. It requests ALOPA to determine whether AP meets the necessary property requirements. ALOPA Service in turn invokes the Validation Service to generate a property report for AP. VS validates the attestation report using the measurement certificates in the repository. It then generates the property report for AP using the property certificates in the repository. It forwards the property report to the ALOPA Service. Using the property report, ALOPA Service determines whether all property requirements have been satisfied. If AP failed to meet the property requirements of the policy, Authorisation Service sends a 'request failed' message to the AR-Proxy. Alternatively, if the decision is yes, meaning all property requirements have been satisfied, then the Authorisation Service updates its decision in the temp opinion database of the Trust Service and presents a request to the Trust Service. Trust Service now evaluates whether the soft trust requirements of the Trust Service have been satisfied and forwards its decision to the Authorisation Service. If the decision is no, meaning AP failed to meet the soft trust requirements of AR, Authorisation Service sends a 'request failed' message to the AR-Proxy which is then forwarded to AP. If the decision is yes from the Trust Service, Authorisation Service invokes the authorisation engine. The dynamic attributes, decision of the ALOPA service and trust services are forwarded to the authorisation engine. The authorisation engine evaluates the policy to determine whether all requirements have been satisfied. User requirements may also be available at this stage. If the authorisation decision is positive, Authorisation Service invokes the hosted service for which AP presented a request. It receives the service response which it forwards to the AR-Proxy which is in turn forwarded to AP.

6. Scenario based analysis of TEDA

We have carried out extensive analysis of how different scenarios are modelled using the TEDA system. In this section, we provide a sample of some common scenarios in the form graphical representation. We illustrate how the model responds to four different situations marked as Case 1 to Case 4. In each of these cases, we assume that the system starts with an initial opinion formed from one positive, one negative and one uncertain experience.

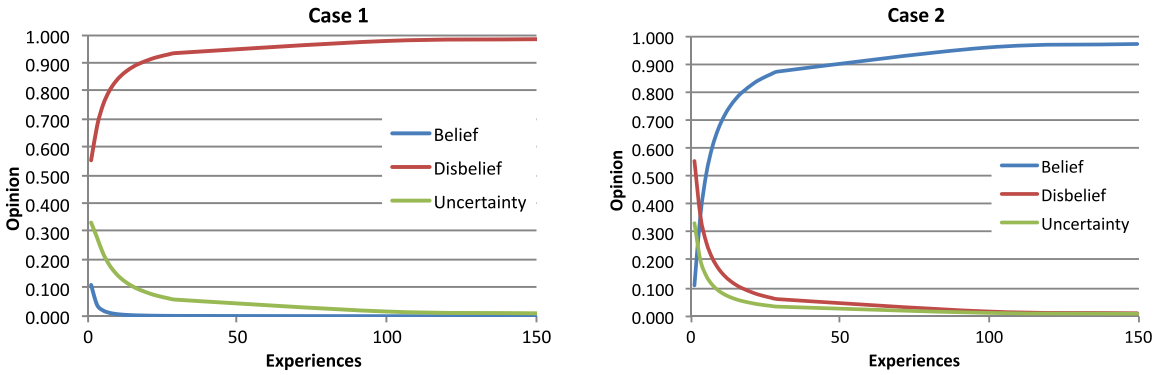

Fig. 7. Case 1: Increasing negative experiences. Case 2: Increasing positive experiences.

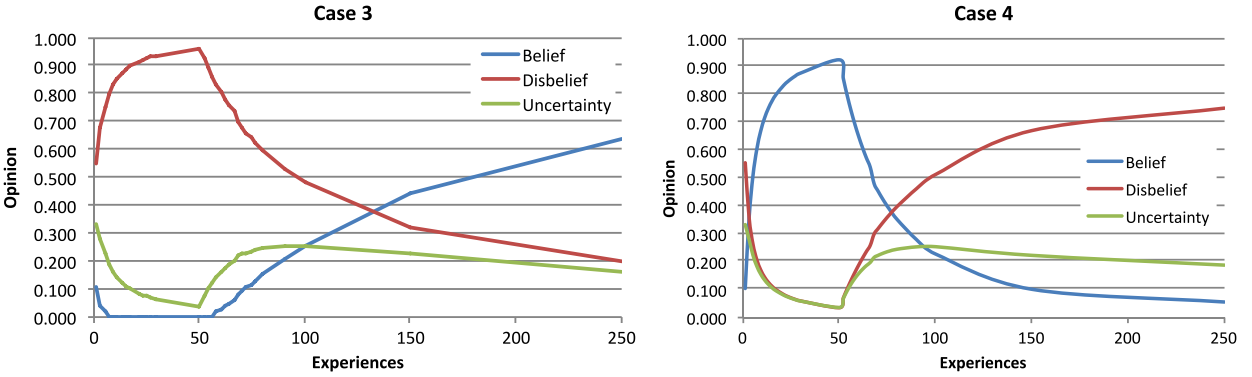

Fig. 8. Case 4: Negative exp followed by positive exp. Case 5: Positive exp followed by negative exp.

Case 1. In the first case, as shown in Fig. 7, there are 150 interactions in total. The initial opinion formed with one positive, one negative and one uncertain experience is $(0.33, 0.33, 0.33)$ for satisfaction trust and $(0.33, 0.33, 0.33)$ for certification trust. The total opinion obtained is $(0.11, 0.55, 0.33)$. Let us assume that the number of bad experiences steadily increases for the rest of the interactions. In the graph, we notice that, because of these bad experiences, disbelief rises and belief falls. We also observe that as the value of belief decreases and the value of disbelief increases, the value of uncertainty decreases, and as the value of belief is almost negligible, the rate of decrease in uncertainty is almost equal to the rate of increase in disbelief.

Case 2. In the second case, as shown in Fig. 7, there are 150 interactions in total. The initial total opinion computed is $(0.11, 0.55, 0.33)$ formed out of equal positive, negative and uncertain experiences for both satisfaction and certification trusts. We assume that the rest of the 149 interactions result only in good experiences. We know that, as the number of good interactions increase, positive experiences are associated with both the properties. Therefore, the value of belief rises and the value of disbelief falls. Accordingly, the value of uncertainty also reduces gradually. After about 100 transactions, belief reaches close to maximum while disbelief and uncertainty are close to minimum.

Case 3. In the third case, as shown in Fig. 8, there are 250 interactions in total. The initial total opinion computed is $(0.11, 0.55, 0.33)$ formed out of equal positive, negative and uncertain experiences for both satisfaction and certification trusts. Let us assume that the first 50 interactions result in bad experiences as a result of which the value of disbelief increases as belief decreases. After this, positive experiences are encountered for the rest of the 200 interactions. We see that the value of belief starts to rise gradually and disbelief starts to fall gradually. However, the gain in belief is slower compared to the fall in disbelief. Uncertainty initially increases to make up for this difference. Subsequently, as belief equals uncertainty, uncertainty starts to fall.

Case 4. In the fourth case, as shown in Fig. 8, there are 250 interactions in total. The initial total opinion is computed as $(0.11, 0.55, 0.33)$. We assume that the first 50 interactions result in good experiences as a result of which the value of belief increases and the value of disbelief decreases. However, after this, the rest of the 200 interactions result in only bad experiences. This increases disbelief and reduces belief. As the rate of fall of belief is greater than the rate of growth of disbelief, uncertainty initially increases to make up for the difference. Soon after, as belief equals uncertainty, uncertainty falls.

The threshold opinion together with the comparison operator determine what opinion values are permitted for authorisation. Currently we are in the process of evaluating various classes of common applications and considering the criteria

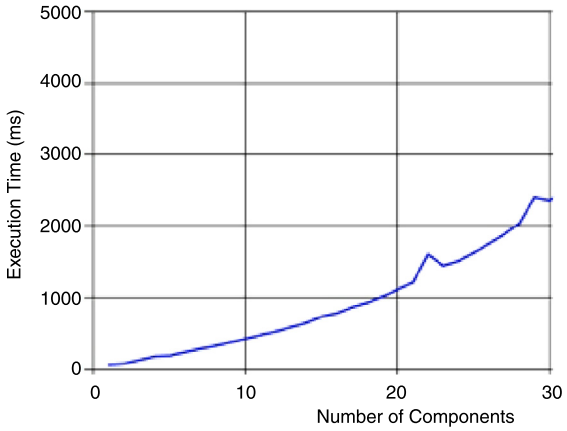

Fig. 9. ALOPA performance graph.

behind these thresholds. We will discuss this as part of our future work in Section 9. The comparison operator is also quite strong at the moment and permits only opinions which satisfy all the three conditions of belief being greater than the threshold and the disbelief and uncertainty being lower than the threshold. Adjusting the comparison operator based on the application requirements is also possible. For example, in many situations, the comparison operator will not authorise components that have belief greater than the threshold but have uncertainty that is slightly less than the threshold. For example, if the threshold is $(0.4, 0.4, 0.2)$ and the derived opinion is $(0.68, 0.1, 0.22)$, the comparison fails because the uncertainty is 0.01 higher than that of the threshold. This is despite the belief being much higher and disbelief being much lower than what is required. The comparison operator can be used to add flexibility if authorisation is desired in such conditions, for example, by ignoring uncertainty when belief and disbelief satisfy the threshold values by a big margin.

7. Temporal performance of TEDA

We have also analysed the temporal performance of our TEDA architecture. In this section, we illustrate this temporal performance by describing some of our simulation results. Our simulations randomly select 'n' number of components in a platform such that $1 \geqslant n \leqslant 30$ plots a graph based on its corresponding execution time. Here, we describe four scenarios for performance evaluation, the first three are the performance simulations for ALOPA, Direct Trust service and Derived Platform Trust service and the fourth is a combined simulation of all the three.

7.1. ALOPA service

The graph in Fig. 9 illustrates that the execution time increases almost linearly with number of components up to 20 randomly selected components. After this number, it begins to increase in an exponential manner. When we investigated this further, we found this to be purely a software implementation issue, due to the implementation of the ALOPA property satisfaction module simulated using Prolog engine recursively using stack for inference. We also found that the spikes at regular intervals were due to interrupts in CPU execution cycles coinciding with regular memory refresh cycles. As the number of components increases, the spikes are consumed within the steep slope of the curve.

7.2. Direct trust service

The execution time for the direct trust service increases almost linearly as the number of components increase. Again the minor spikes at random intervals are attributed to execution time discrepancy due to task switching in multi-programming operating systems where several hidden system and utility processes running in the background. The low slope of the graph indicates that with the increase in the number of components being tested, the average execution time per component gradually decreases. (See Fig. 10.)

7.3. Derived platform trust

The execution time of derived platform trust is similar to that of the direct trust performance graph. (See Fig. 11.)

7.4. TESA–ALOPA combined trust service

In this scenario, we simulate the execution of the complete trust model that combines the ALOPA service along with the trust services. Here, we observed that the execution time increases almost linearly with number of components. (See Fig. 12.)

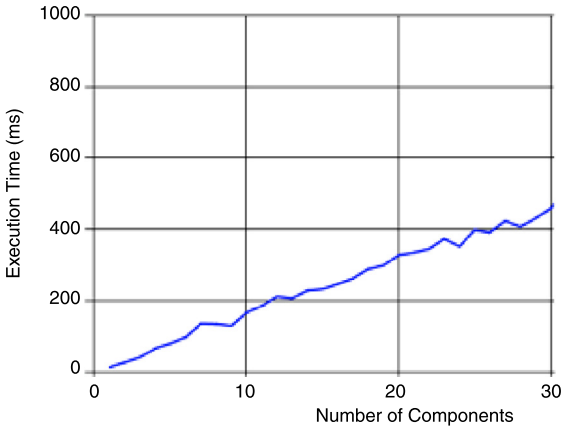

Fig. 10. TEDA direct trust performance graph.

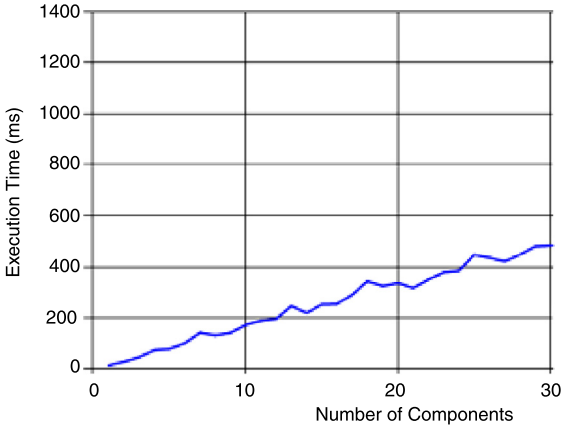

Fig. 11. TESA weighted trust performance graph.

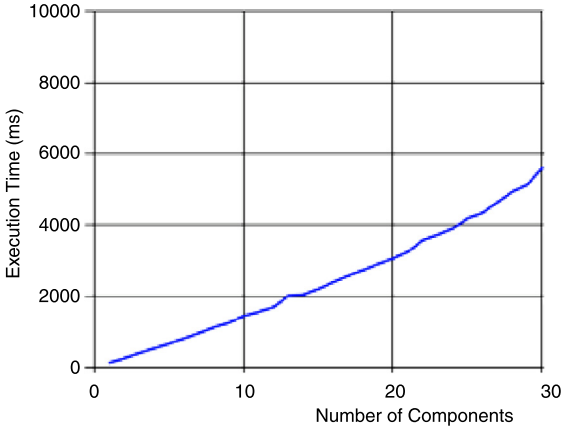

Fig. 12. TESA–ALOPA combined performance graph.

8. Discussion

The developed trust enhanced distributed authorisation architecture (TEDA) has several distinct advantages over currently existing authorisation systems. From the results, we can see that the performance of the proposed architecture TEDA is better than traditional authorisation system as it incorporates both dynamic changes as well as platform based characteristics in addition to the user credential based features. As far as we are aware, there exists no authorisation system like the one proposed here with these dynamic trust enhanced characteristics in the authorisation decision making.

- Integration of hard and soft trust factors: The TEDA architecture enables better decision-making using a combination of both hard and soft trusts of a trusted platform. Hard trust is derived from the properties that are satisfied by the platform at the time of attestation. Soft trust is derived from the opinions maintained on the past behaviour of the platform based on both direct experiences and referrals received from recommenders within the system. The properties satisfied by the platform at the time of attestation are also translated into opinions and are incorporated in the computation of overall soft trust. This ensures that the 'present' state of the platform also contributes significantly to the computation of soft trust.
- Support for complex property requirements: The TEDA architecture provides the ability to abstract an authorisation property requirement to other properties using ALOPA policies. For example, the abstracted property may define how the authorisation property is implemented in the system.
- Support for different policy design choices: Our authorisation model provides support for different policy options for trust-enhanced authorisation. The system consists of two policy sets, one in the form of ALOPA policies and the other in the form of access control policies. ALOPA policies facilitate the expression of complex conditions that enable the satisfaction of a given property. Authorisation policies, clearly segregated from the ALOPA policies, are used to represent the direct authorisation requirements of a system. It supports the expression of hard trust requirements as property sets, soft trust requirements as opinions and other dynamic attributes. In the soft trust requirements, support is available for the expression of opinion thresholds of component sets or the platform as a whole. Support for trust-specific policies, such as how much a recommender is trusted for a recommendation, is confined to the trust service and is clearly separated from authorisation requirements.
- Flexible policy management: The segregation of ALOPA policies, authorisation policies and trust-specific policies provides for flexible management. For example, if the conditions under which a property is satisfied in a system must be changed, it only affects the ALOPA aspect of policy administration and leaves the authorisation policy intact. This is expected to be the usual case where authorisation policy administrators are different from property administrators.
- Integration of user and platform authorisation: The TEDA architecture supports both user and platform authorisation requirements in the system. Our model is flexible enough to either segregate the two types of policies or combine them, depending on the system requirements. In the current implementation, user-related policies are assumed to be stored in a separate policy base and the user authorisation engine has access to this policy base. User-related requirements are specified and evaluated independently of the platform requirements. Such a model enables different policy administration teams to maintain user and platform policies separately. Alternatively, other design choices are possible, such as merging user and platform requirements in the same policy but performing evaluation by different authorisation engines, or merging the policy specification and using a single evaluation engine that handles the policy in its entirety. Such designs can be easily accomplished by extending the authorisation policy with user requirements and extending the platform engine to resolve a user policy. Policies involving a feedback between the user and platform components may also be integrated. For example, the system performs platform or user authorisation first and only if this is successful does it progress to perform the other by reserving the more resource-intensive checks to be performed later.

9. Conclusion

Distributed authorisation aims towards protecting access to resources from unwarranted access. While authorisation of a user is based on privileges held, authorisation of a platform must be based on the properties satisfied by that platform. In this vein, we have proposed a trust enhanced distributed authorisation architecture (TEDA) for trusted platforms. In TEDA, we have combined the notions of hard trust and soft trust to determine if a platform can be trusted and if it can be authorised to access a requested resource. Hard trust is determined using attestation and property based attestation that enable reasoning of system properties. However, as all properties may not be certified, we have proposed a simple logic language ALOPA that defines rules for property satisfaction. A rule in ALOPA defines the conditions under which a property is considered satisfied for a given platform. For soft trust, we have proposed a trust model TM that takes into account uncertainties. TM uses past experiences with a platform to determine different trust values for direct, recommended and derived trusts for a platform. Trust requirements are then expressed in terms of corresponding trust thresholds. By combing the hard trust and soft trust requirements, a platform authorisation service is able to determine if a platform can be trusted. We then discussed the implementation of the model in the context of authorisation for web services and provided an analysis of the results. Some key advantages of TEDA include the integration of hard and soft trust factors, the ability to support for complex property requirements (using ALOPA), flexibility in policy management (platform policies in turn consisting of ALOPA and trust policies), ability to combine user and platform authorisation and easy integration in different distributed architectures.

In TEDA, the threshold opinion together with the comparison operator determines what opinion values are permitted for authorisation in the soft trust service. The threshold value must be set based on the authorisation policy requirements. We believe that setting the threshold is also dependent on the type of application for which authorisation is sought. Some critical applications, such as health care and financial systems, may choose to set very high thresholds, while relatively less critical applications, such as certain online gaming scenarios, may set lower thresholds. This is part of our current work where we are analysing our model for different application contexts. Also, in this paper, we have analysed the user

and platform authorisation issues independent of each other and have only considered simple combinations such as only perform user authorisation only if platform authorisation is successful. As future work, we would like to integrate these two aspects more closely using the feedback to update policy requirements. For example, extra authorisation checks may be imposed on the user if trust in the platform is not very high but just enough to exceed the required threshold. This in turn can be used to optimise risk due to dynamic changes in security threats. Our results show that such a trust enhanced authorisation architecture, integrating user and platform authorisation taking into account dynamic changes in trust with uncertainties is able to make better authorisation decisions in a distributed web services environment.

Acknowledgment

We would like to thank Tarashankar Rudra for carrying out the temporal performance simulation analysis of TEDA.

Appendix A. Subjective logic

Evidence to opinion mapping. Let *pos*, *neg*, *unc* denote the total number of positive, negative and uncertain experiences of A on B regarding the property x. Then A's opinion about property x in B is given by ${}^{A}\omega_B^x$ where ${}^{A}b_B^x$ is A's belief on B about x, ${}^{A}d_B^x$ is A's disbelief on B about x and ${}^{A}u_B^x$ is A's ignorance on B about x.

$$
\begin{aligned}
{}^{A}\omega_B^x &= {}^{A}b_B^x, {}^{A}d_B^x, {}^{A}u_B^x \\
{}^{A}b_B^x &= {}^{A}pos_B^x/\left({}^{A}pos_B^x + {}^{A}neg_B^x + {}^{A}unc_B^x\right) \\
{}^{A}d_B^x &= {}^{A}neg_B^x/\left({}^{A}pos_B^x + {}^{A}neg_B^x + {}^{A}unc_B^x\right) \\
{}^{A}u_B^x &= {}^{A}unc_B^x/\left({}^{A}pos_B^x + {}^{A}neg_B^x + {}^{A}unc_B^x\right)
\end{aligned}
$$

Conjunction of opinions. Let A define two opinions ${}^{A}\omega_B^x$ and ${}^{A}\omega_B^y$ about two different properties x and y in the same platform B. Then ${}^{A}\omega_B^{x,y}$ is called the conjunction ($\odot$) of the opinions ${}^{A}\omega_B^x$ and ${}^{A}\omega_B^y$ representing A's opinion about both x and y in B.

$$
\begin{aligned}
{}^{A}\omega_B^{x,y} &= {}^{A}\omega_B^x \odot {}^{A}\omega_B^y \\
{}^{A}b_B^{x,y} &= {}^{A}b_B^x.{}^{A}b_B^y, \qquad {}^{A}d_B^{x,y} = {}^{A}d_B^x + {}^{A}d_B^y - {}^{A}d_B^x.{}^{A}d_B^y, \qquad {}^{A}u_B^{x,y} = {}^{A}b_B^x.{}^{A}u_B^y + {}^{A}u_B^x.{}^{A}b_B^y + {}^{A}u_B^x.{}^{A}u_B^y
\end{aligned}
$$

Consensus of opinions. If A forms an opinion ${}^{A}\omega_B^x$ on B about the property x and C forms another opinion ${}^{C}\omega_B^x$ on B about the same property x, then the consensus ($\oplus$) of the two opinions is equivalent to the opinion ${}^{A,C}\omega_B^x$ formed on B about x by an imaginary system that represents both A and C.

$$
\begin{aligned}
{}^{A,C}\omega_B &= {}^{A}\omega_B^x \oplus {}^{C}\omega_B^x \\
{}^{A,C}b_B^x &= \left({}^{A}b_B^x.{}^{C}u_B^x + {}^{C}b_B^x.{}^{A}u_B^x\right), \qquad {}^{A,C}d_B^x = \left({}^{A}d_B^x.{}^{C}u_B^x + {}^{C}d_B^x.{}^{A}u_B^x\right), \qquad {}^{A,C}u_B^x = \left({}^{A}u_B^x.{}^{C}u_B^x\right)
\end{aligned}
$$

Discounting opinions. If A has an opinion on B and if B has an opinion on C, then A's opinion about C is computed by discounting B's opinion about C with A's opinion about B. Let ${}^{A}\omega_B = ({}^{A}b_B, {}^{A}d_B, {}^{A}u_B)$ and ${}^{B}\omega_C = ({}^{B}b_C, {}^{B}d_C, {}^{U}u_C)$, then ${}^{A}\omega_C$ gives the discounted opinion ($\otimes$) of ${}^{A}\omega_B$ and ${}^{B}\omega_C$.

$$
\begin{aligned}
{}^{A}\omega_X &= {}^{A}\omega_B \otimes {}^{B}\omega_C \\
{}^{AB}b_C &= {}^{A}b_B.{}^{B}b_C, \qquad {}^{AB}d_C = {}^{A}b_B.{}^{B}d_C, \qquad {}^{AB}u_C = {}^{A}d_B + {}^{A}u_B + {}^{A}b_B.{}^{B}u_C
\end{aligned}
$$

References

[1] D. Ferraiolo, R. Kuhn, Role-based access control, in: Proceedings of the 15th NIST-NCSC National Computer Security Conference, 1992, pp. 554–563.
[2] S. Jajodia, P. Samarati, V.S. Subrahmanian, A logical language for expressing authorizations, in: SP'97: Proceedings of the 1997 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 1997, p. 31.
[3] J. DeTreville, Binder: A logic-based security language, in: SP'02: Proceedings of the 2002 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 2002, p. 105.
[4] N. Li, J.C. Mitchell, Rt: A role-based trust-management framework, in: Proceedings of the Third DARPA Information Survivability Conference and Exposition, IEEE Computer Society, 2003, pp. 201–212.
[5] A. Herzberg, Y. Mass, J. Michaeli, Y. Ravid, D. Naor, Access control meets public key infrastructure, or: Assigning roles to strangers, in: SP'00: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 2000, p. 2.
[6] M. Blaze, J. Feigenbaum, J. Lacy, Decentralized trust management, in: Security and Privacy, 1996, pp. 164–173, tY – CONF.
[7] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis, The KeyNote trust-management system version 2, RFC 2704, September 1999.
[8] TCG, TPM main – part 1 design principles, version 1.2, revision 103, Trusted Computing Group, July 2007.
[9] J. Poritz, M. Schunter, E.V. Herreweghen, M. Waidner, Property attestation: Scalable and privacy-friendly security assessment of peer computers, IBM Research, Tech. Rep., May 2004.
[10] A.-R. Sadeghi, C. Stüble, Property-based attestation for computing platforms: Caring about properties, not mechanisms, in: NSPW'04: Proceedings of the 2004 Workshop on New Security Paradigms, ACM, USA, 2004, pp. 67–77.

[11] A. Nagarajan, V. Varadharajan, M. Hitchens, E. Gallery, Property based attestation and trusted computing: Analysis and challenges, in: International Conference on Network and System Security, 2009, pp. 278–285.
[12] A. Jøsang, A logic for uncertain probabilities, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 9 (3) (2001) 279–311.
[13] TCG, TCG infrastructure working group: Integrity report schema specification, 2006.
[14] TCG, TCG infrastructure working group: Reference manifest schema specification, November 2006.
[15] TrustedJava, Trusted computing for the Java platform, Graz University of Technology, Graz, Austria, http://trustedjava.sourceforge.net/.
[16] T. Winkler, et al., jTSS Wrapper for TSS 1.1b stack, The Institute for Applied Information Processing and Communications/Open Trusted Computing [online]; available at: http://trustedjava.sourceforge.net/index.php?item=jtssw/about.
[17] R. Sailer, X. Zhang, T. Jaeger, L. van Doorn, Design and implementation of a TCG-based integrity measurement architecture, in: 13th Usenix Security Symposium, San Diego, CA, August 2004.
[18] OpenTC, The open trusted computing consortium [online]; available at: http://www.opentc.net/.
[19] A. Nagarajan, V. Varadharajan, M. Hitchens, ALOPA: Authorisation logic for property attestation in trusted platforms, in: 6th International Conference on Autonomic and Trusted Computing, Brisbane, Australia, 2009, pp. 134–148, 2011, pp. 564–573.
[20] A. Nagarajan, Techniques for trust enhanced distributed authorisation using trusted platforms, PhD thesis, Macquarie University, Australia, August 2010.