# QuanTM: A Quantitative Trust Management System*

Andrew G. West, Adam J. Aviv, Jian Chang, Vinayak S. Prabhu,
Matt Blaze, Sampath Kannan, Insup Lee, Jonathan M. Smith, and Oleg Sokolsky

Department of Computer and Information Science - University of Pennsylvania, Philadelphia

{westand, aviv, jianchan, vinayak, blaze, kannan, lee, jms, sokolsky}@cis.upenn.edu

## ABSTRACT

Quantitative Trust Management (QTM) provides a dynamic interpretation of authorization policies for access control decisions based on upon evolving reputations of the entities involved. QuanTM, a QTM system, selectively combines elements from *trust management* and *reputation management* to create a novel method for policy evaluation. Trust management, while effective in managing access with delegated credentials (as in PolicyMaker and KeyNote), needs greater flexibility in handling situations of partial trust. Reputation management provides a means to quantify trust, but lacks delegation and policy enforcement.

This paper reports on QuanTM's design decisions and novel policy evaluation procedure. A representation of quantified trust relationships, the *trust dependency graph*, and a sample QuanTM application specific to the KeyNote trust management language, are also proposed.

## Keywords

Quantified Trust Management, Trust Management, Reputation Management, QuanTM, KeyNote, TNA-SL

## 1. INTRODUCTION

The emergence of distributed topologies and networked services has resulted in applications that are stored, maintained, and accessed remotely via a client/server model. The advantages of such a setup are many, but the challenges of access control and identity management must be addressed. *Trust management* and *reputation management* are two differing approaches to the problem. While effective with regard to explicit declarations, trust management lacks applicability when relationships are characterized by uncertainty. Thus, trust management is useful in enforcing existing trust relationships but ineffective in the formation of partially-trusted ones. Reputation management provides a means of quantifying trust relationships dynamically, but lacks access enforcement and delegation mechanisms.

To address this divide we introduce the notion of Quantitative Trust Management (QTM), an approach that merges concepts from trust and reputation management. It (QTM) creates a method for specifying both policy and reputation for dynamic decision making in access control settings. A system built upon QTM can not only enforce delegated authorizations but also adapt its policy as partial information becomes more complete. The output is a quantitative trust value that expresses *how* much a policy-based decision should be trusted given the reputations of the entities involved. Further, to make this novel concept concrete, we propose QuanTM, an architecture for supporting QTM.

In this application of QuanTM, we use the KeyNote [8, 7] (KN) trust management language and specification, due to its well defined delegation logic and compliance system. Summarily, a KN evaluator checks a user's access credentials against local policy to produce a *compliance value* from a finite and predefined set of values. The compliance value is then used to make access decisions. KN allows principals to delegate access rights to other principals without affecting the resulting compliance value. Further, KN is monotonic: If a given request evaluates to some compliance value, adding more credentials or delegations will not lower that value.

We argue that credentials should not be explicitly trusted, nor should the trustworthiness of delegating principals be ignored. Furthermore, the result of evaluation for a given access request may need to be dynamic [9]. Service providers may find it desirable to arrive at different opinions based on local constraints, policies, and principals for the same request. In QuanTM, this is easily expressed.

We address these issues in the following two ways: (1) It includes a means to dynamically assign reputation to principals and their relationships within a request, and (2) It provides a mechanism for combining this information to produce a trust value. In QuanTM, a trust value (often a real number) is used to represent the the trustworthiness of a given compliance value and how it was reached.

Our proposed QuanTM architecture (see Fig. 1) consists of three sub-systems:

1. Trust management consists of a *trust language evaluator* that verifies requests meet policy constraints, and a *trust dependency graph (TDG) extractor* that constructs a graph representing trust relationships.

2. Reputation management consists of two modules. First, a *reputation algorithm* to dynamically produce

*This research was supported in part by ONR MURI N00014-07-1-0907. POC: Insup Lee, lee@cis.upenn.edu


*EUROSEC* '09 Nuremberg, Germany

reputation values by combining feedback. These reputation values weigh TDG edges. Second, a *reputation quantifier* computes the trust value for a given request by evaluating the weighted TDG.

3. DECISION MANAGEMENT is composed of a *decision maker* that arrives at an access determination based on a trust value, context, and an application specific meta-policy that encodes a cost-benefit analysis.

The design of QuanTM has been guided by the requirement that the individual components will be application-specific, and thus, we have designed QuanTM modularly. QuanTM provides a simple interface by which different trust management languages, reputation algorithms, and decision procedures may be included. In this paper, we propose a QuanTM design instance that utilizes the KeyNote language and TNA-SL [11, 12] reputation algorithm. This instance's implementation and evaluation is the subject of future work.

## 2. ARCHITECTURE OVERVIEW

Our proposed architecture is broken into three sub-systems: A trust manager, reputation manager, and decision manager. One goal of QuanTM is to be highly modular, so the system can be customized to application specifics. In Fig. 1 below, we present the QuanTM architecture using KeyNote as the trust management language.

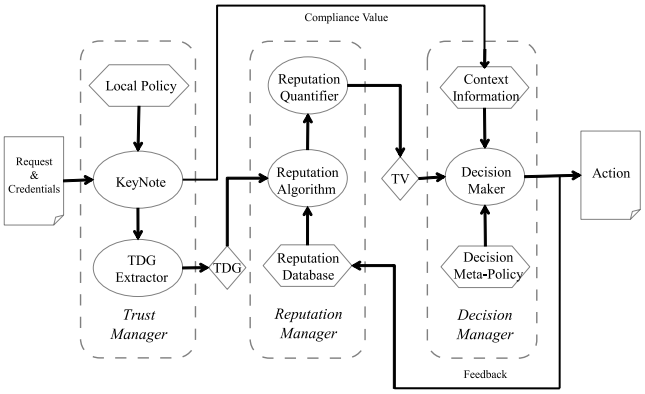


**Figure 1: Proposed QuanTM Architecture**

In what follows we will assume that a group of principals initiate each action request. First, a set of principals present an action request along with their credentials. The *KeyNote (KN) compliance checker* computes the compliance value using local policy. This value is then relayed to the decision manager to (later) make an action decision. The compliance checker also passes relevant delegation data to the *trust dependency graph (TDG) extractor*. The resulting TDG is given to the reputation management sub-system. In Sec. 5 we present a detailed discussion of TDG extraction.

The reputation management sub-system's task is twofold: (1) Use a *reputation algorithm* and a reputation database (that is dynamically updated based on feedback from runtime behavior) to assign reputation values on the edges of the TDG, and (2) Evaluate the TDG using the *reputation quantifier* to produce the final trust value (TV). Sec. 6 provides a description of reputation management modules.

The last sub-system, the decision manager (see Sec. 7), is composed primarily of a *decision maker* that requires input from the previous sub-systems as well as context monitors. It analyzes the information using a decision meta-policy and cost-benefit analysis to grant or deny a request.

## 3. RUNNING EXAMPLE

Trust is at the heart of all access decisions, and to demonstrate this property, we introduce a running example. Assume there is a central database resource that stores orders for bicycle manufacturer Arrow Bikes (`A`). Certainly, stores selling Arrow Bikes should be able to update the DB with new orders. However, should a store have the ability to view (query) the DB and see other stores orders? This should depend on the conditions of their access request.

Let us suppose Arrow Bikes are sold in two stores it owns, Bikes-R-Us (`B`) and Cathy's Bike Supply (`C`). A third store, Driven-2-Bike (`D`), is owned by a third party but operated by `B`. Finally, a fourth store, E-Bike (`E`), sells Arrow Bikes on-line. Because of Internet security concerns (*i.e.,* injection attacks), Arrow Bikes requires all E-Bike query requests to be signed off by a trusted seller, either `B` or `C`.

This scenario presents a number of trust relationships. From `A`'s perspective, its most prominent relations are with `B` and `C`. They are both more trusted than third-party `D`, whose trust follows from that given to `B`. Clearly, requiring the approval of `B` or `C`, seller `E` is in the least trusted position.

Throughout the paper, we will refer back to this example and its trust relationships. We will first encode these relationships as KeyNote assertions and show how this encoding meets the trust needs of Arrow Bikes. Next, relationships will be represented in a trust dependency graph that will be evaluated to determine the trust value for a given request. Lastly, we will describe a decision procedure and meta-policy that best suits this example. We will refer to the principals, *i.e.*, Arrow Bikes and Bikes-R-Us, by the first letter in their name, *i.e.*, `A` and `B`, wherever appropriate.

## 4. KEYNOTE

KeyNote is a declarative language describing relationships among *principals* and evidence that permits principals to perform certain actions. These relationships are specified as *policies*. If cryptographically signed and used externally, a policy can be viewed as a *credential*. KeyNote credentials and policies are known as *assertions*. When a trust inquiry is made, a set of action requesters (principals), present a set of policies and credentials that allow the group to perform an action. The KeyNote 'compliance checker' evaluates this input and returns a compliance value (CV) in a linearly ordered set, between an application specified `MIN_CV` and `MAX_CV`[1], which is used to make application control decisions.

### 4.1 Language Specification

A policy has three primary[2] components; the `Authorizer`, `Licensees`, and `Conditions` fields.

The `Authorizer` is the principal delegating trust, who is essentially 'saying' this assertion. KeyNote provides a special `Authorizer`, `POLICY`, that is the root of all trust. Valid delegation chains must emanate from `POLICY`. Additionally, a `Signature` field specifies the signature of a credential.

[1]KN specifies these as `MIN_TRUST` and `MAX_TRUST`, we label them CVs to distinguish them from the TVs used herein.
[2]For a more complete language specification we refer the reader to RFC 2704 [6].

The Licensees states to which principal(s) the Authorizer is delegating trust to, and it also expresses from whom delegation chains must be present. It is written as a logical statement of AND/OR operators between principals[3]. The logic of the Licensees field also describes how CVs are combined to form a single output. The MAX function is applied to OR delegations and the MIN function to AND ones. For example, if the Licensees field is 'ADAM && JOHN' then the CV of the delegation will be the minimum of the CVs of the delegation chains emanating from Adam and John. Note that when describing the presence of a delegation chain, 'ADAM && JOHN' will evaluate to true iff either (1) Adam and John are both action requesters; or (2) JOHN is an action requester and there is a delegation chain from ADAM to an action requester (i.e., there exists a delegation from ADAM to PAUL and PAUL is an action requester), or vice versa; or (3) there are delegation chains from both JOHN and ADAM to the action requesters.

Finally, the Conditions field permits comparison using environmental variables. Conditions are written as propositions, which when true imply a CV. When more than one value is implied, the maximal one is used.

Provided this specification, one can now determine a CV for an entire delegation chain. First one computes CVs via the comparisons of the Conditions field for all credentials in the chain. These are then combined up the chain using MIN/MAX per the Licensees field until the root node (POLICY) is reached. This final CV is then returned to the application.

By design, the KeyNote evaluator is monotonic. That is, the addition of assertions/principals/etc. will never decrease the returned CV.

## 4.2 Example Policy

We now use our running example to demonstrate how KeyNote is used. Recall that Arrow Bikes (A), with an order database, wants to give sellers full 'update' rights but potentially limit 'query' rights. As shown below, POLICY delegates all trust to A, giving it full database access (ASRT_0). Then, A delegates a portion of these rights to two stores it owns, B and C (CRED_1). Namely, B and C are authorized to update the database and potentially issue a query.

```
Comment: ASRT_0
Authorizer: POLICY
Licensees: A
Conditions:
  operation == "update" -> "True";
  operation == "query" -> "True";

Comment: CRED_1
Authorizer: A
Licensees: B || C
Conditions:
  operation == "update" -> "True";
  operation == "query" -> "Maybe";
Signature: "rsa-sig:1294..."
```

We use {False, Maybe, True} as our CV set. True, representing MAX_CV, implies an action should be permitted. False, representing MIN_CV, implies an action should be denied. Maybe indicates that permission is unclear. In ASRT_0 and CRED_1 above, A has more rights than B or C. If A were to request a query, KN would return True. If B or C made the same request, the result would be Maybe. These assertions precisely match the requirements of the example as described in Sec. 3.

Next, we wish to represent seller D, who has third-party ownership, but is operated by B. This can be encoded by having D's access rights emanate from B, or, more precisely, D's access rights are authorized by B as shown below:

```
Comment: CRED_2
Authorizer: B
Licensees: D
Conditions:
  operation == "update" -> "True";
  operation == "query" -> "Maybe";
Signature: "rsa-sig:8471..."
```

Finally, the relationship between A and seller E is the most complex. In our example, query requests by E must be approved by either B or C. This can be encoded into the Licensees field of the credential as below:

```
Comment: CRED_3
Authorizer: A
Licensees: E && (B || C)
Conditions:
  operation == "update" -> "True";
  operation == "query" -> "Maybe";
Signature: "rsa-sig:3850..."
```

If E presents CRED_1 and CRED_3 without either B or C as co-requesters, then the request will be *invalid* and the CV will be False, by default. However, if E co-requested with either B or C or provided a credential from either authorizing E, the request would be *valid*. This example also provides another valid credential chain for E to gain access rights. If E were to present CRED_1, CRED_2, and CRED_3 with D as a co-action requester, then a different – albeit valid – chain would form. When B delegated access rights to D, it also delegated authorization rights (a property of KN), so D may also authorize requests made by E.

These two valid but structurally different requests result in the same CV. However, as we will later exemplify with our reputation manager, the TVs associated with these requests may differ, since the principals, delegations, and credentials involved also differ.

# 5. TRUST DEPENDENCY GRAPH

We now describe the format and properties of a *trust dependency graph* (TDG). A TDG is a directed graph representation of the relationships between principals, delegations, and credentials that were used to obtain a CV. We design our TDG to be expressive enough to specify, and therefore quantify, at least three types of reputation: (1) Reputation of a principal, $\rho_1$, (2) Reputation of a delegation, $\rho_2$, and (3) Reputation of a credential, $\rho_3$.

These three types allow us to apply reputations to precisely those entities that can influence a CV. The influence of the first type ($\rho_1$) is clear; a principal involved, either as an authorizer or action requester, holds direct influence over the CV computed. The reputation of a delegation ($\rho_2$) is the reputation of a principal with respect to how it delegates its access rights. The third reputation type ($\rho_3$) is of a

[3]The specification also defines a $K$-OF operation, but it can be composed using AND/OR delegations and thus is omitted from this discussion.

credential itself. Since a credential can be used many times in many contexts, past observations of its use can influence its reputation.

If there is a cycle in the credentials, it will be broken by the compliance checker (KeyNote, in our example). However, there may be multiple ways to break a cycle and still arrive at the same CV. Instead of exploring all possible forms we argue that the appropriate TDG encodes the precise dependency structure that the compliance checker used to produce it (the CV). In this way, the computed TV will characterize precisely that CV and the evaluation structure which brought it (the CV) about[4].

## 5.1 TDG Design

A TDG is a directed graph $G = (V, E)$. Every TDG has a root node, $v_{POL}$, that encodes the root of trust, *i.e.*, KeyNote's special authorizer `POLICY`. The set of principal nodes, $V_{PR}$, represent either authorizers or action requesters, *i.e.*, principals. Operator nodes, $V_{OP}$, are used to represent propositions of licensing, *i.e.*, `AND` and `OR` operations of the `Licensees` field. Operator nodes must have two children. To facilitate this, null nodes denoted by $V_\emptyset$, are used in place of principals who are neither authorizers or action requesters.

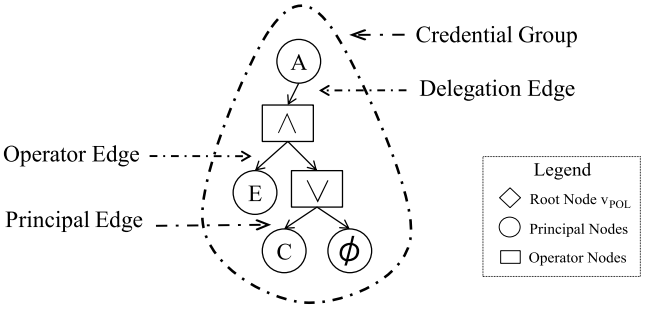


**Figure 2: Defining TDG Edges and Nodes**

There are four mutually disjoint sets of edges ($E$). The first set of edges consists of *policy edges*, which are used only to connect the $v_{POL}$ node with operator nodes. The set of policy edges is denoted as $E_{POL}$ where $E_{POL} \subset \{v_{POL}\} \times V_{OP}$. These edges represent assertions authorized by the root of trust. The set of *delegation edges*, $E_{DEL}$, consists of edges between principal nodes and operator nodes, *i.e.*, $E_{DEL} \subset V_{PR} \times V_{OP}$. These edges represent credentials authorized by the principal at the tail of the edge. The set of *operator edges*, $E_{OP} \subset V_{OP} \times V_{OP}$, is used to express complex delegations and is a result of our binary constraint. The final set of edges, $E_{PR} \subset V_{OP} \times V_{PR}$, consists of *principal edges*, which connect operator nodes with principal nodes.

## 5.2 TDG Properties

Edge weights (reputations) will be assigned using the reputation algorithm. The edge type determines the reputation type that will be applied. More precisely, weights applied to edges from $E_{PR}$ are for reputations of a principal ($\rho_1$). Reputations of delegations ($\rho_2$) are placed on $E_{DEL}$ edges. In this manner, the first two types of reputation are represented. We identify sub-graphs of the TDG according to the credential from which they were extracted. These sub-graphs form *credential groups* (indicated by dashed groupings in Fig. 2) that are weighed by $\rho_3$ reputations.

Edges from $E_{OP}$ are not weighed because operator edges are used to combine operations for a complex delegation and do not represent separate delegations themselves. Edges from $E_{POL}$ are not weighed either, as permitting weights on policy edges would allow a discount in the root of trust, *i.e.*, `POLICY`, which is assumed to be fully trusted.

## 5.3 TDG Examples

Returning to our running example, recall the two request formats presented in Sec. 4.2. `E` wants to make a request to the database and there are two scenarios by which it can gain access. In the first request format ($R_1$), `E` makes a request with `B` (or `C`), and presents `CRED_1` and `CRED_3` in order to form a valid request. `ASSRT_0` is used locally to complete the chain to `POLICY`. In the second format ($R_2$), `E` requests with D and presents one additional credential, `CRED_2`, which delegates rights from `B` to `D`. In Fig. 3, we present the TDG for both requests.

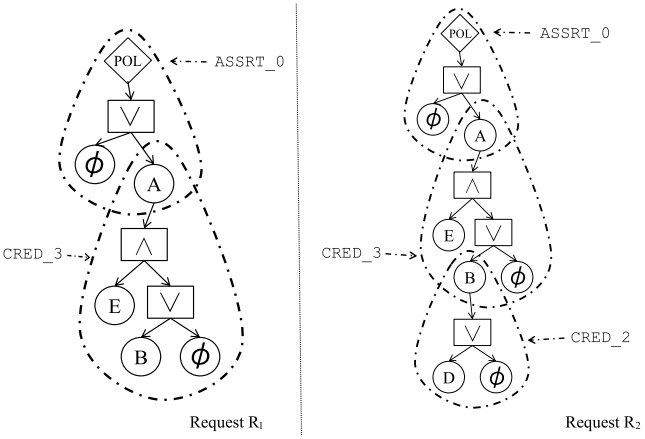


**Figure 3: Running Example TDGs**

In both requests the dependencies are clear. In $R_1$, `A` authorizes `CRED_3` whose `Licensees` field is satisfied by the presence of `B` and `E` as action requesters. In $R_2$, the action requesters are `E` and `D`. Principal `B` is still present, but as an authorizer, not an action requester. Reputations of principals ($\rho_1$), can now be assigned on edges from $E_{PR}$, *i.e.*, those edges terminating at nodes `E`, `B`, and `A` in $R_1$ and additionally `D` in $R_2$. Reputations of delegations ($\rho_2$) are placed on edges from $E_{DEL}$, *i.e.*, the edges connecting `A` and `B` with an operator node. Credential reputations ($\rho_3$) are placed on the groupings.

The additional credential and edges present in $R_2$ reflect a longer delegation chain by which `E` may gain access rights; they will have a profound effect on the resulting TV. This will become clear in the next section when we assign reputation values to the TDGs and evaluate the structure to produce a TV for each request.

# 6. REPUTATION MANAGEMENT

The reputation management sub-system has two distinct tasks: (1) It assigns reputation values to TDG edges using a *reputation algorithm* that aggregates *reputation database* entries, and (2) It evaluates the TDG using a *reputation quantifier* to produce a TV.

[4]It is possible to compute a TV from a TDG with cycles as a least fixed point, but we do not describe it here.

## 6.1 Reputation Database

We utilize a reputation database (DB) to obtain interaction histories concerning the principals, delegations, and credentials in a TDG. The DB may, for example, store positive and negative feedbacks[5] between directed principal pairs. Context data is stored so reputation can be calculated at varying granularity. An example reputation DB specific to our running example is shown in Tab. 1. Database entry $F1$, for example (which would result from request $R_1$), states that principal `A` had a positive experience with requesters `B` and `E`, when they utilized a service authorized by `A` via `CRED_3`. Note the presence of principal `F` (Frank's Bikes), which despite being external to the TDG(s), may have experience relevant to reputation computation.

| ID | SRC | DEST | +/- | AUTH | CRED |
|---|---|---|---|---|---|
| $F1$ | A | {B,E} | + | {A} | {CRED_3} |
| $F2$ | A | {D,E} | − | {A,B} | {CRED_2,CRED_3} |
| $F3$ | A | {F} | + | {A,C} | {CRED_W,CRED_X} |
| $F4$ | F | {D} | + | {C,F} | {CRED_Y,CRED_Z} |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

**Table 1: Example Reputation Database**

Depending on the reputation type ($\rho$) the reputation algorithm is asked to calculate, it will query the DB differently, thereby limiting the entries it aggregates. To calculate reputation of a principal ($\rho_1$), the entire DB is used. Reputation of a delegation ($\rho_2$) would select only rows where the authorizer matches the principal that is delegating, *i.e.*, the start terminal of $E_{DEL}$. Reputation of a credential ($\rho_3$) would do a similar matching on the credential field.

For the sake of simplicity, we will focus on calculating trust in principals ($\rho_1$), keeping in mind the capability for such refinement. Further, we omit feedback mechanisms in this paper and assume the reputation database is given. We plan to present a detailed feedback system in future work.

## 6.2 Reputation Algorithm

The reputation algorithm describes the procedure by which reputation DB entries are combined to assign weights to appropriate TDG edges. Reputations represent a directed valuation of trust between a principal and another entity. In particular, the source of this directed reputation will always be the service provider, $SP$. This is intuitive; $SP$ hosts the service, so it should be the root of all reputations used to make an access decision regarding that service.

For example, an $E_{PR}$ edge to a licensee $L$ would be weighed by calling `computeRep(`$L$`)`. This returns $SP$'s reputation in $L$. Depending on the algorithm implemented, the value may be user-centric, *i.e.*, `computeRep(`$L$`)` may return different values to different service providers (reputation DBs can span many services), or it may be global, *i.e.*, `computeRep(`$L$`)` will return the same value, regardless the service provider.

There are two $\rho$ types our proceeding (and eventually continued) example neglects. To weigh an $E_{DEL}$ edge representative of a delegation made by $L$, `computeRep(DEL(`$L$`))` would be called. The reputation algorithm will utilize the query style of Sec. 6.1 to obtain a feedback set, perform computation, and return $SP$'s trust in $L$ with respect to authorizations. In a similar fashion, a `computeRep(`$CRED_X$`)` call will return $SP$'s reputation in credential $CRED_X$.

How feedbacks are combined is reputation-algorithm dependent. Generally, some combination of direct and indirect (transitive) experience is used. Feedback aggregation may involve principals external to the TDG, as we exemplified with principal `F` in the reputation database example. Further, reputation algorithm design is primarily focused on limiting the capabilities of malicious principals who may purposely manipulate feedback to gain an advantage. There are many reputation algorithms in existence. The survey work of Li and Singhai [16] describes several. We do not believe that any single reputation algorithm is most effective on all application domains.

Most reputation algorithms can be interfaced into our system, with one limiting constraint; the resulting reputation value must have an absolute interpretation. This way, applications can set and consistently interpret policies based on reputation thresholds. Systems like EigenTrust [14], which uses relative trust, are therefore inappropriate.

To provide an example of a reputation algorithm, we present Trust Network Analysis with Subjective Logic [11, 12] (TNA-SL), which will be exclusively used in forthcoming examples.

### 6.2.1 TNA-SL

The expressiveness and rigor of TNA-SL make its use appropriate in this application. While theoretically sound, TNA-SL suffers from a lack of scalability [23]. In TNA-SL, trust is stored as an *opinion* ($\omega$), a 4-tuple $(b, d, u, \alpha)$ which represent belief, disbelief, uncertainty, and a base-rate, respectively. At all times $(b + d + u) = 1$, and $\alpha \in \mathbb{R}$ on $[0 \dots 1]$ is used to store a-priori notions of trust. TNA-SL converts past (positive and negative) interactions into a 4-tuple opinion per Tab. 2.

| | | |
|---|---|---|
| belief ($b$) | = | $(pos/(pos + neg + 2.0))$ |
| disbelief ($d$) | = | $(neg/(pos + neg + 2.0))$ |
| uncertainty ($u$) | = | $(2.0/(pos + neg + 2.0))$ |
| base-rate ($\alpha$) | = | User-Defined |

**Table 2: Calculating Opinions from Feedback**

Two logical operators, *discount* and *consensus* are of note. Tab. 3 shows their calculation. *Discount* is used to evaluate transitive chains. It would be used, for example, if user $A$ wanted to calculate an opinion about user $C$ using information at intermediate user $B$ ($\omega_C^{A:B} = \omega_B^A \otimes \omega_C^B$). *Consensus* is used to average together two opinions. Suppose user $A$ and user $B$ both have opinions about user $C$. To consolidate these into a single opinion, consensus would be used ($\omega_C^{A \diamond B} = \omega_C^A \oplus \omega_C^B$). To export opinions to a numeric value, the *expected value* is calculated as $EV = b + \alpha u$.

When TNA-SL is called, it first converts reputation DB entries into an opinion digraph. Then, an uncertainty minimal graph between two (provided) terminals is constructed. Finally, the SL-operators reduce the graph to a single opinion. We will take a more detailed look at how TNA-SL is computed in conjunction with our running example.

### 6.2.2 Edge Weight Example

Returning to the example, recall the TDGs of Fig. 2. We weigh the $E_{PR}$ edge (present in $R_1$ and $R_2$) terminating at E, via service provider `A`, by calling `computeRep(E)`.

[5]Though we discuss only binary feedback, it could take the form of an ordered set, or a continuous variable.

| Discount: $\otimes$ | Consensus: $\oplus$ |
|---|---|
| $b_C^{A:B} = b_B^A b_C^B$ | $b_C^{A \diamond B} = \frac{b_C^A u_C^B + b_C^B u_C^A}{u_C^A + u_C^B - u_C^A u_C^B}$ |
| $d_C^{A:B} = b_B^A d_C^B$ | $d_C^{A \diamond B} = \frac{d_C^A u_C^B + d_C^B u_C^A}{u_C^A + u_C^B - u_C^A u_C^B}$ |
| $u_C^{A:B} = d_B^A + u_B^A + b_B^A u_C^B$ | $u_C^{A \diamond B} = \frac{u_C^A u_C^B}{u_C^A + u_C^B - u_C^A u_C^B}$ |
| $\alpha_C^{A:B} = \alpha_C^B$ | $\alpha_C^{A \diamond B} = \alpha_C^A$ |

**Table 3: Discount and Consensus Operators**

Suppose we are utilizing the (extended) reputation database of Tab. 1, which we assume to contain all the relevant feedback history. We begin by summing interaction histories pair-wise between principals, *i.e.*, the direct relation from A to E is characterized by 3 positive and 1 negative feedbacks. Tab. 2 describes how to convert these relations to opinions, *i.e.*, $\omega_{\mathtt{E}}^{\mathtt{A}} = (0.50, 0.17, 0.33, \alpha)$[6]. Using this, we can construct a digraph with principals as nodes and opinions on edges.

Using this digraph, we enumerate all acyclic paths between our start terminal (A) and end terminal (E). The confidence, $(1-u)$, in each path is calculated using the *discount* operator, and the path list is sorted from most-to-least confident. This is presented in Tab. 4.

| ID | Path | Confidence-Expression | Conf |
|---|---|---|---|
| $P1$ | A⇒E | $[1-u \in \omega_{\mathtt{E}}^{\mathtt{A}}]$ | 0.66 |
| $P2$ | A⇒F⇒B⇒E | $[1-u \in (\omega_{\mathtt{F}}^{\mathtt{A}} \otimes \omega_{\mathtt{B}}^{\mathtt{F}} \otimes \omega_{\mathtt{E}}^{\mathtt{B}})]$ | 0.58 |
| $P3$ | A⇒F⇒E | $[1-u \in (\omega_{\mathtt{F}}^{\mathtt{A}} \otimes \omega_{\mathtt{E}}^{\mathtt{F}})]$ | 0.56 |
| $P4$ | A⇒B⇒E | $[1-u \in (\omega_{\mathtt{B}}^{\mathtt{A}} \otimes \omega_{\mathtt{E}}^{\mathtt{B}})]$ | 0.17 |
| $P5$ | A⇒B⇒F⇒E | $[1-u \in (\omega_{\mathtt{B}}^{\mathtt{A}} \otimes \omega_{\mathtt{F}}^{\mathtt{B}} \otimes \omega_{\mathtt{E}}^{\mathtt{F}})]$ | 0.05 |

**Table 4: Confidence-sorted Path List**

A graph, $G$, is now constructed. Traversing the sorted list, a path $p$ is added to $G$ iff $(G+p)$ is a direct-series parallel graph (DSPG) [22]. At list's end, $G$ is the DSPG between A and E that minimizes uncertainty. In our example, $G = \{P1 \cup P2 \cup P4\}$. $G$ may be written as a canonical expression of SL-operators (notice that *discount* and *consensus* apply to series and parallel composition, respectively):

$$OP_G = \omega_{\mathtt{E}}^{\mathtt{A}} \oplus [\{\omega_{\mathtt{F}}^{\mathtt{A}} \oplus (\omega_{\mathtt{B}}^{\mathtt{A}} \otimes \omega_{\mathtt{F}}^{\mathtt{B}})\} \otimes \omega_{\mathtt{E}}^{\mathtt{F}}] = (0.52, 0.18, 0.30, 0.50)$$

This opinion, $OP_G$, or its expected-value, $EV(OP_G) = 0.67$, defines the reputation of A in principal E and weighs the aforementioned edge.

## 6.3 Reputation Quantifier

Now that reputation values have been assigned to TDG edges, it is the responsibility of the *reputation quantifier* to evaluate the TDG to produce a final TV. In order to do this, the quantifier requires: (1) *Parallel operations* and (2) *Transitive operations*. Parallel operations combine reputations across edges on the same level of the graph, *i.e.*, two reputation values beneath a single edge. Transitive operations combine reputation values up the graph, *i.e.*, between a sub-graph and its connecting edge or credential group.

Because there are three reputation types ($\rho$), seven different functions must be defined per application specifics:

[6]For the purposes of this example, $\alpha = 0.5$ in all opinions, as we assume there is no a-priori trust between principals.

**Parallel Operations**:

- $f_\vee$,$f_\wedge$ combine values on edges beneath an operator node, with respect to the operator label.
- $f_{\mathsf{del}}$ combines values on edges beneath a principal node.
- $f_{\mathsf{cred}}$ combines values across credential groupings.

**Transitive Operations**:

- $g_{\mathsf{prin}}$ combines a $\rho_1$ value on an edge with the value of the edge's connecting sub-tree, the output of $f_{\mathsf{cred}}$.
- $g_{\mathsf{del}}$ combines a $\rho_2$ value on an edge with the value of the edge's connecting sub-tree, the output of $f_\vee/f_\wedge$.
- $g_{\mathsf{cred}}$ combines a $\rho_3$ value of the credential group with the value of the containing sub-tree, the output of $f_{\mathsf{del}}$.

### 6.3.1 Computing Trust

The function evalTrust$(v)$ computes the TV of a TDG, and it proceeds in a depth first matter beginning at $v_{POL}$. To properly define the function we introduce access functions: (1) chld$(v, c)$ returns a tuple $\langle \overline{e}, \overline{v} \rangle$ with respect to credential group $c$, where $\overline{e} = \langle e_1, \ldots, e_k \rangle$ denotes the list of outgoing edges from node $v$ in the credential group $c$, $\overline{v} = \langle v_1, \ldots, v_k \rangle$ denotes the corresponding list of child nodes, and each $v_i$ is connected by edge $e_i$ (if $v \in V_{OP}$, then irrespective of $c$, $\langle e_1, e_2, v_1, v_2 \rangle$ corresponding to the credential group of $v$ is returned because by definition a $V_{OP}$ node can only be a member of a single credential group); (2) crd$(v)$ returns $\langle c_1, \ldots, c_k \rangle$ where each $c_i$ corresponds to a different credential group that node $v$ is the authorizer per the graph; and finally, (3) wt$(e)$ and wt$(c)$ return the weight on an edge $e$ and on the credential group $c$, respectively (if no weight is present, NULL_REP[7] is returned).

The function evalTrust is defined as follows:[8]

**Function** evalTrust$(v)$ for the trust value of a TDG

```
switch v do
    case v ∈ V_OP s.t. OP ∈ {∨, ∧}
        ⟨e1, e2, v1, v2⟩ = chld(v, ∗);
        p1 = g_prin(wt(e1), evalTrust(v1));
        p2 = g_prin(wt(e2), evalTrust(v2));
        return f_OP(p1, p2)
    case v ∈ V_PR ∪ {v_POL} ∪ V_∅
        c = NULL_REP;
        foreach c' in crd(v) do
            d = NULL_REP;
            foreach e', v' in chld(v, c') do
                d = f_del(g_del(wt(e'), evalTrust(v')), d);
            end
            c = f_cred(g_cred(d, wt(c')), c);
        end
        return c;
end
```

[7]NULL_REP corresponds to null-trust for all reputation types. It differs from the notion of complete distrust, and when combined with other reputation values, NULL_REP should leave them unchanged: *i.e.*, MIN($a$, NULL_REP)$= a$.

[8]In KeyNote it is not possible for two delegations to occur in a single credential. Hence, for a KeyNote based trust manager, $f_{\mathsf{cred}}$ is the identity function.

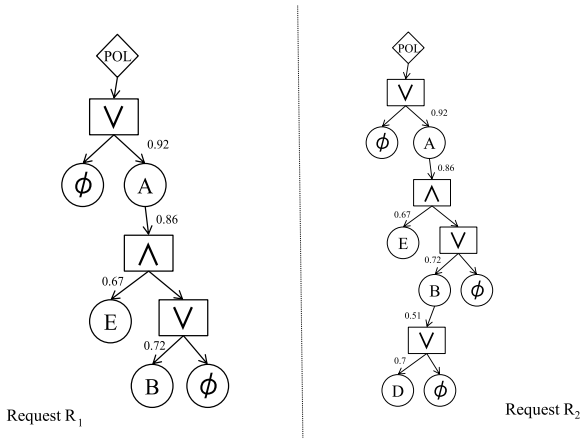


**Figure 4: $R_1$ and $R_2$ with Reputation Values**

### 6.3.2 *Example Quantification*

We now refer the reader back to the running example. In the last section, we assigned a $\rho_1$ value to the edge that connects `E` to the TDG. Now, we present the TDGs of $R_1$ and $R_2$ with the rest of the reputation values applied (see Fig. 4). Recall the TDG for $R_1$ is the same as that for $R_2$, but the latter has an additional delegation from `B`. Given both requests occur with equivalent reputation database values, the edge weights are the same wherever applicable. We also choose functions for the parallel/transitive operations. For the purposes of this example, we choose $f_\vee$ and $f_\wedge$ to be `AVG` and `MIN`, respectively[9] Multiplication is used for all transitive functions. In general, function selection is made on an application specific basis.

By application of the above algorithm to the TDGs presented in Fig. 4, we arrive at the following TV for $R_1$:

$$0.53 = (0.92 * 0.86) * \texttt{MIN}(0.67, 0.72).$$

The TV for $R_2$ evaluates to

$$0.20 = (0.92 * 0.86) * \texttt{MIN}(0.67, (0.72 * 0.51 * 0.7)).$$

## 7. DECISION MANAGER

The *decision manager* (DM) is the last component in the QuanTM workflow. It aims to make an optimal decision with respect to the current context, CV, TV, and a predefined meta-policy. The final decision will be sent to the service provider where the action request will either be fulfilled or denied.

The decision manager's meta-policy is defined during setup and is application dependent. For example, a simple strategy is to define the meta-policy in terms of a trust threshold, *i.e.*, fulfill the request if TV is greater than a threshold. A more sophisticated meta-policy may take context information into account. This could be the specific `true` conditional statements that produced the CV, or it could include system states and other context monitors.

Complex meta-policies will also utilize a cost-benefit evaluation strategy. For example, a TV may be read as the probability a request is trustworthy, and risk/reward functions will be defined as quantified measures describing the consequences of some action. Then, the optimal decision is the one that maximizes the expected benefit and minimizes the cost. The precise specification and formal model of this meta-policy is a focus of future work.

[9]Our choice of `AVG` may produce non-monotonic TV results for some delegation chains. Such monotonicity is neither required nor desired. For simplicity of presentation, we define parallel operation functions to compositional, *e.g.*, we define $\texttt{AVG}(x, y, z))$ to be $\texttt{AVG}(x, \texttt{AVG}(y, z))$. The evalTrust algorithm needs minor modifications to lift this restriction.

### 7.1 Decision Example

Concluding our running example, let us assume that service provider `A` is implementing a simple meta-policy of the threshold style. When the calculated CV is `True` the request is permitted, if `False` the request is denied; this decision is irregardless of the TV. Only when the CV is `Maybe` is the TV examined. In such a situation, `A` has determined that requests with TVs greater than 0.5 are trustworthy enough to proceed, and otherwise the request is denied.

One can now see how the differing structure of the two requests bears heavily on the final access decision. We above calculated the TV of $R_1$ as 0.53 and the TV of $R_2$ as 0.20. $R_1$ has a higher TV because more reputable principals/delegations are involved. The addition of `CRED_3` and principal `D` in $R_2$ produce a lower TV. As it pertains to this meta-policy, even though `E` has an authorization in both requests, the authorization in $R_1$ is stronger than in $R_2$. Thus in the $R_2$ case, `E` is denied access, while an $R_1$ request is permitted.

## 8. RELATED WORK

While literature is plentiful with respect to trust management and reputation management as separate domains, their intersection offers fewer resources. Perhaps most similar to our system is the work of Columbo *et al.* [10] who combine a reputation system with the Role-based Trust Management Language [18] to permit fine-grain access control for Grid computing. Much as our reputation quantifier permits, Colombo *et al.* use operators to combine trust across and down delegation paths. While the operations Columbo employs are fixed, ours are modular. Additionally, Columbo stores reputation as a separate credential using [19]; we prefer a reputation database. At the highest level, QuanTM is a general-purpose framework whereas Columbo's work is specific to Grid computing and perhaps a subset of QuanTM.

Our work also draws parallels to the PACE approach of Suryanrayana *et al.* [21]. PACE describes an integration architecture for trust/reputation systems into decentralized applications. A layered approach with visibility constraints is utilized to prevent exposing the infrastructure to malicious attacks. Much like our design it is application driven, complete with trust, credential, and key managers. However, it differs in the fact it is only specifies a generalized model and is not an actual use-case.

It should be noted many languages could have been chosen to exemplify the QuanTM framework, not just KeyNote. Examples include SecPAL [5], Delegation Logic [17], DCC [2, 3], and WS-Security [4], among others [13]. KeyNote was chosen for a number of reasons. First, the KeyNote language is well documented and thus allows for easy implementation and extension. Second, KeyNote is terse and has few delegation formats, simplifying both our development process and our examples herein. Finally, KeyNote is one of the few TM languages that has been deployed as a standard, including modules in operating systems like FreeBSD and OpenBSD [20, 1].

# 9. CONCLUSION

We have proposed the QuanTM framework which provides a dynamic interpretation of authorization policies for access control decisions. In the past, the statement and evaluation of application-relevant authorization policies has been a major challenge for decentralized systems and distributed topologies, especially in the context of networked services. While trust management has addressed aspects of this problem, it lacks the ability to provide partial trust analysis in principals, delegations, and credentials. Reputation management allows for a more nuanced approach to handling partial trust, but lacks access enforcement and delegation.

We make three major claims concerning the work we have described herein. First, we have presented the concept of a *Quantitative Trust Management* (QTM) system, one which has the desirable attributes of both trust and reputation management. Second, we have refined this approach in a novel framework, *QuanTM*, which adds reputation mechanisms to trust management enforcement. In particular, QuanTM arrives at dynamic decisions based upon context, a quantified trust value, and a cost-benefit aware meta-policy. Third, we have developed the *trust dependency graph (TDG)* to represent information internal to QuanTM's unified trust and reputation management systems.

We see several promising directions for future work. First, we intend to implement the QuanTM instance describe herin and apply it to real life data sets to validate its effectiveness and applicability. The Internet Domain Name System (DNS) is one possibile target for analysis, as it has both delegation and reputation aspects.

Second, we intend to further analyze each QuanTM subsystem, with particular emphasis on developing a formal presentation of the decision meta-policy, a thorough specification of the feedback mechanism, and a detailed study of the properties of TDGs and their evaluation procedure.

Third, by using KeyNote in our example implementation, we gained the benefit of identity authentication via a public/private key model. Some languages require third-party certifying authorities, and the inclusion of such in QuanTM would provide greater generality. The use of KeyNote illustrates one intriguing possibility to explore; the use of reputation management as a possible control path by which credential *revocation* (a challenge for KeyNote [15]) can be achieved. This can be accomplished by applying a minimal reputation value for a revoked credential (see Sec. 6).

Fourth and finally, we wish to analyze QuanTM's effectiveness against different attack scenarios, especially as it relates to malicious manipulation of feedback mechanisms and the handling of stale credentials.

We believe much experimentation is necessary to understand the benefits of QTM, and QuanTM is a first step in demonstrating its feasibility and applicability.

# 10. REFERENCES


[1] OpenBSD. `http://www.openbsd.org`.

[2] M. Abadi. Access control in a core calculus of dependency. *ACM SIGPLAN Notices*, 41(9):263–273, 1999.

[3] M. Abadi, A. B. N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, pages 147–160, January 1999.

[4] B. Atkinson. Web services security (WS-Security). `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp`, 2002.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. Technical report, Microsoft Research, 2006.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote trust-management system, version 2. *IETF RFC*, 2704:164–173, September 1999.

[7] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). In *Security Protocols Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63, 1999.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society, 1996.

[9] M. Blaze, S. Kannan, A. D. Keromytis, I. Lee, W. Lee, O. Sokolsky, and J. M. Smith. Dynamic trust management. *IEEE Computer (Sp. Issue on Trust Mangement)*, 2009.

[10] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for Globus. In *GADA '07*, volume 4804 of *LNCS*, pages 1505–1515, 2007.

[11] A. Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[12] A. Jøsang, R. Hayward, and S. Pope. Trust network analysis with subjective logic. In *Proceedings of the 29th Australasian Computer Science Conference*, 2006.

[13] L. Kagal, S. Cost, T. Finin, and Y. Peng. A framework for distributed trust management. In *Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, May 2003.

[15] A. D. Keromytis and J. M. Smith. Requirements for scalable access control and security management architectures. *ACM Transactions on Internet Technology*, 7(4), November 2007.

[16] H. Li and M. Singhai. Trust management in distributed systems. *IEEE Computer*, 40(2):45–53, February 2007.

[17] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6, 2003.

[18] N. Li and J. Mitchell. RT: a role-based trust-management framework. *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, 1:201–212, April 2003.

[19] J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. *LNCS*, 2995:48–62, 2004.

[20] ports@FreeBSD.org. FreeBSD port keynote-2.3-1. `http://www.freebsd.org/ports/security.html`.

[21] G. Suryanarayana, J. R. Erenkrantz, and R. N. Taylor. An architectural approach for decentralized trust management. *IEEE Internet Computing*, 9(6):16–23, 2005.

[22] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 1–12, Atlanta, GA, 1979. ACM.

[23] A. G. West, S. Kannan, I. Lee, and O. Sokolsky. An evaluation framework for reputation management systems. Working chapter for *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*, (Zheng Yan, ed.).