Risk Management for Distributed Authorization

Christian Skalka*
University of Vermont

X. Sean Wang
University of Vermont

Peter Chapin
University of Vermont

Abstract

Distributed authorization takes into account several elements, including certificates that may be provided by non-local actors. While most trust management systems treat all assertions as equally valid up to certificate authentication, realistic considerations may associate risk with some of these elements, for example some actors may be less trusted than others. Furthermore, practical online authorization may require certain levels of risk to be tolerated. In this paper, we introduce a trust management logic based on the system *RT* that incorporates formal risk assessment. This formalization allows risk levels to be associated with authorization, and authorization risk thresholds to be precisely specified and enforced. We also develop an algorithm for automatic authorization in a distributed environment, that is directed by risk considerations. A variety of practical applications are discussed.

Keywords: Distributed Authorization, Trust Management Logic.

1 Introduction

Trust management systems provide a means to specify and enforce distributed authorization policies. Many such systems possess a formal foundation for making authorization decisions, so that security is rigorously enforced and so that designers and users have a clear understanding of policies and semantics. Current state-of-the-art includes SPKI/SDSI [23, 13] and RT [19]. The expressiveness and rigor of these systems have become increasingly important to security in modern distributed computing infrastructures, as web-based interactions continue to evolve in popularity and complexity.

Authorization in trust management usually takes into account several facts and assertions, including certificates provided by non-local, untrusted actors. Although cryptographic techniques provide certain measures of confidence in

*Corresonding author. Address: University of Vermont, Department of Computer Science, Burlington, VT 05405. Phone: (802)656-1920. Email: skalka@cs.uvm.edu

this setting, not all components of authorization can realistically be used with the same level of confidence. The Pretty Good Privacy (PGP) framework acknowledges this, by including a notion of trustworthiness of certificates in their legitimacy measure [2]. Furthermore, efficient online authorization decisions often require a weakening of ideal security, since the latter may be prohibitively expensive. This weakening may involve the acceptance of assertions that would otherwise be verified, in case lowered confidence levels are more tolerable than the danger of intractability. Thus, many practical distributed authorization decisions include elements of *risk* associated with authorization components, where risk could be associated with trust or other practical considerations making some facts more or less risky than others.

A rigorous assessment of authorization should accurately assess risk, but risk in trust management is usually an informal consideration. In this paper, we develop a trust management logic called RT^R, introduced in a simpler form in previous work [11], that formally incorporates formal risk assessment. The system is a variant of RT [19], and includes an abstract definition of risk, a means to associate risk with individual assertions, and a semantics that assesses risk of authorization by combining the risk of assertions used in authorization decisions. This formalization promotes development of a distributed authorization algorithm allowing tolerable levels of risk to be precisely specified and rigorously enforced.

1.1 Contributions

The main contributions of this paper are twofold. First, we develop a rigorous formal foundation for an authorization calculus that incorporates a notion of risk, and aggregation of risk for particular authorization decisions, in the system RT^R. The system is designed as an extension to the system RT [20]. The definition of risk, risk ordering, and aggregation of risk are left abstract modulo some basic sanity requirements, so RT^R is a framework for risk management in authorization, that can be specialized for particular applications. Our theory also features thresholds, which are formal specifications of tolerable risk. The per-role granularity of thresholds allows different security domains to specify their own risk tolerance, and allows these specifications to interact in particular authorization decisions.

Our second main contribution is an algorithm for performing authorization in a distributed setting, called distributed chain discovery. The algorithm does not depend on all credentials for authorization to be known locally, but allows certificates relevant to particular decisions to be retrieved dynamically from remote locations based on a simple storage scheme. The technique is based on one defined by other authors [21], but modified to reflect risk management as specified in the semantics. More importantly, the algorithm is risk-directed: as partial authorization proofs are constructed, associated risk is maintained, and certificates that would cause thresholds to be exceeded are avoided. If risk is chosen to reflect computational expense, this technique provides a heuristic to improve efficiency, and modulating thresholds allows computational cost to be

balanced with e.g. issues of trust.

1.2 Paper Outline

The remainder of the paper is organized as follows. In Sect. 2, an overview of the RT_0 system is given for background. In Sect. 3, motivations for adding risk measures and management to authorization is discussed. In Sect. 4, we define the syntax and set-theoretic semantics of RT^R, an authorization logic with risk assessment, including a formalization of risks, risk ordering, risk aggregation and thresholds. It is demonstrated that this semantics provides a meaningful interpretation of any set of credentials. The system RT^R is defined as a framework, and several example instances are given in Sect. 5 to illustrate the use and flexibility of the system. In Sect. 6, we give a graph-theoretic interpretation of RT^R that is equivalent to the set-theoretic semantics, and show that so-called credential graphs can be automatically reconstructed by a distributed chain discovery algorithm, as an implementation of distributed authorization. In Sect. 7, we discuss some interesting practical applications of RT^R, and we conclude with a summary of the paper and remarks on related work in Sect. 8.

2 Overview of RT

Rather than defining a new trust management logic for a formalization of risk, we take advantage of the existing RT system [19]. This system combines the strengths of role-based access control with an expressive trust management logic, and enjoys a variety of existing implementation techniques [21]. We believe these features make RT one of the most advanced trust management systems, and an appealing setting for the development of formal risk assessment.

The RT role-based trust management system is actually a collection of trust management logics, all of which are variations on a basic logic called RT_0 [19]. Variations include separation of duties and delegation. In this same spirit, we propose a variation on RT_0 to incorporate a formalization of risk assessment, so we briefly review RT_0 here to provide necessary background.

In RT_0, individual actors, or principals, are called *Entities* and are defined by public keys. We let A, B, C, D, E range over entities. Each entity A can create an arbitrary number of *Roles* in a namespace local to the entity, denoted $A.r$. The *RoleExpressions* of RT^R, denoted f, are either entities or roles or constructed from other role expressions by *linking* and *intersection*. Formally, role expressions are generated by the following grammar:

$$f ::= A \mid A.r \mid A.r.r \mid f \cap \cdots \cap f$$

Role expressions are used to define roles via credentials. To define a role an entity issues credentials that specify the role's membership. Some of these credentials may be a part of private policy; others may be signed by the issuer and made publically available. The overall membership of a role is taken as the memberships specified by all the defining credentials.

RT_0 provides four credential forms:

1. $A.r \longleftarrow E$

   This form asserts that entity E is a member of role $A.r$.

2. $A.r \longleftarrow B.s$

   This form asserts that all members of role $B.s$ are members of role $A.r$. Credentials of this form can be used to delegate control over the membership of a role to another entity.

3. $A.r \longleftarrow B.s.t$

   This form asserts that for each member E of $B.s$, all members of role $E.t$ are members of role $A.r$. Credentials of this form can be used to delegate control over the membership of a role to all entities that have the attribute represented by $B.s$. The expression $B.s.t$ is called a *linked role*.

4. $A.r \longleftarrow f_1 \cap \cdots \cap f_n$

   This form asserts that each entity that is a member of all role expression forms $f_1, \ldots, f_n$ is also a member of role $A.r$. The expression $f_1 \cap \cdots \cap f_n$ is called an *intersection role*.

Authorization is then cast as a role membership decision: an access target is represented as some role expression f, and authorization for that target for some entity A is equivalent to determining whether A is a member of f. In such a decision, we call f the *governing role*. Authorization always assumes some given finite set of credentials, denoted $\mathcal{C}$. We use $\mathit{Entities}(\mathcal{C})$ to represent the entities used in a particular set of credentials $\mathcal{C}$, and similarly $\mathit{RoleNames}(\mathcal{C})$, $\mathit{Roles}(\mathcal{C})$, etc.

2.1 Example

Suppose a hotel H offers a room discount to certain preferred customers, who are members of $H.\mathit{preferred}$. The policy of H is to grant a discount to all of its preferred customers in $H.\mathit{preferred}$ as well as to members of certain organizations. H defines a role $H.\mathit{orgs}$ that contains the public keys of these organizations. Into that role H places, for example, the key of the AAA, the American Auto Association. These credentials are summarized as follows:

$$H.\mathit{discount} \longleftarrow H.\mathit{preferred} \qquad H.\mathit{discount} \longleftarrow H.\mathit{orgs}.\mathit{members}$$

$$H.\mathit{orgs} \longleftarrow \mathit{AAA}$$

Now imagine that at a later time a special marketing plan is created to encourage travelers to stay at H. A decision is made that all members of the AAA are automatically preferred customers and thus the credential $H.\mathit{preferred} \longleftarrow \mathit{AAA}.\mathit{members}$ is added to the policy.

Finally suppose that Mary is a member of the AAA. She has a credential issued by the AAA, $\mathit{AAA}.\mathit{members} \longleftarrow M$, attesting to that fact. By presenting

this credential to H's web service Mary can prove in two distinct ways that she is authorized to receive the discount. On one hand she is a member of an organization in $H.orgs$. On the other hand she is, indirectly, a preferred customer of H. Certain practical considerations may motivate H's decision about which "proof" to use. As we'll see in Sect. 6, specified risk thresholds in RT^R can steer authorization in the right direction.

3 Practical Motivations

Credentials in RT are all created equal, in that each represents a true statement in the knowledge base of an authorization decision. There is no facility for denoting that one credential may be more or less "believable" than another. In this RT is similar to other trust management systems, such as SPKI/SDSI [13]. However, recent practice has shown that such a manichean view is not consistent with reality. In this section we discuss practical issues that suggest the need for a more fine-grained view of the different risks associated with credentials.

3.1 Risk as an Authentication Metric

The system RT treats PKI transparently. That is, the semantics of RT does not concern itself with the details of associating keys with users, nor authenticating this association. Nevertheless, credentials for authorization decisions are established by certificates, that must be authenticated. Furthermore, authentication is not necessarily a simple process in open distributed systems, rather modern PKI allows for construction of global public-key namespaces on the basis of local certification authorities. In this setting authentication can involve traversing a chain of intermediary authorities in distinct administrative domains [7]. Distinct domains may be trusted to varying degrees, so depending on what domain boundaries are crossed, one public key certification may be more trustworthy than another, or there may exist multiple paths of varying degrees of trust to establish the same certification. *Authentication chains* of this sort resemble certificate chains as we study them in this paper, though the latter is at a level of abstraction above the former.

To address issues of trust in authentication chains, *authentication metrics* have been developed to assign measurements of trust to particular certifications [22]. The measure of assurance of a certification is based on the set of chains that establish it, which are in turn based on a combination of trust measures assigned to nodes in the chain, i.e. particular administrative domains. A number of schemes have been proposed, including a *legitimacy* measure for PGP [2], illustrating the practical relevance of the idea.

Thus, in standard authorization frameworks, the treatment of credentials as inerrable facts ignores any degrees of assurance that are established for particular certifications. But such degrees are clearly of potential interest to the authorizer, for example if multiple credentials are involved in a particular decision and each is of low assurance, then these may compound to yield unaccept-

ably low assurance for the decision, whereas it may be tolerable if only one of them has low assurance. By allowing an assignment of risks to RT credentials, and providing a means of combining them in the authorization semantics, our proposed extension to RT allows a formal accounting of these considerations.

3.2 Balancing Security and Efficiency

In addition to issues of trust, efficiency issues may affect the risk associated with credentials. For example, if cryptographic certification of a particular credential via the PKI would be too time-consuming, the authorizer may prefer to avoid using that credential. The algorithm we define for online authorization is given a threshold of tolerable risk for authorization, and avoids authorization chains that exceed this threshold. Thus, our directed search technique provides a heuristic for efficiency in case risk is measured in computational cost.

A more interesting example is the use of cached credentials. Caching credentials is a useful technique to avoid re-retrieving and re-authenticating certificates. However, certificates are commonly assigned expiration dates, as in x509 [17], and most trust management systems, including RT and SPKI/SDSI, do not consider expiration in the formal authorization semantics but only in the initial credential certification [13]. Therefore, re-use of a cached credential runs the risk of re-using expired rights. Similarly, systems with certificate revocation must take care not to re-use cached credentials that have been revoked. However, if the system uses certificate revocation lists as does for example SPKI/SDSI [13], maintaining a current view of revocation may be problematic due to the likely need to update lists with non-local information. This reality is reflected in the *verify-only* mode of QCM [15]. To implement certificate revocation, QCM relies on a database of revoked certificate identifiers, but in verify-only mode its external communications are blocked for the sake of efficiency. In such situations, our formalism allows trust management systems to represent and manage the risks associated with credentials that have expired or that may have been revoked. A sufficiently abstract representation of risk even allows to balance the cost of trusting questionable credentials against the efficiency benefits gained by their usage, as discussed in Sect. 7.3.

4 The System RT^R

The system RT^R is RT_0 extended with a formal definition of risk assessments. In this section we define and characterize the syntax and semantics of RT^R. We define a set theoretic semantics for RT^R, since this allows an easy correspondence with the graph theoretic characterization of RT^R for distributed chain discovery given in the next section. Unlike RT_0, the semantics of RT^R not only takes into account role membership, but the risk associated with role membership. This semantic representation allows formal discussion of *thresholds*, the specification of tolerable risk levels for role membership on a per-role basis.

4.1 Syntax and Semantics

The system RT^R is defined as a framework, parameterized by a *risk ordering*, which is required to be a complete lattice $(\mathcal{K}, \preccurlyeq)$. We let κ and K range over elements and subsets of $\mathcal{K}$ respectively, and let $\top$ and $\bot$ denote the top and bottom elements of the lattice. Any instantiation of RT^R is expected to specify a set of lattice elements and a decidable risk ordering $\preccurlyeq$. Intuitively, $\preccurlyeq$ allows comparison between greater and lesser risk, e.g. if one credential is more trusted than another, or less computationally expensive to retrieve. Also, operations for *aggregating* risks– that is, combining risks in proofs of authorization– must be specified. The system provides two forms of aggregation, risk aggregation and intersection aggregation. Each of these are total functions from pairs of risks to risks. The latter is provided for combining risks in intersection role definitions, while the former handles all other forms of aggregation. We believe this distinction should be made, because intersection roles can represent agreement between principals, which in practice may reduce risk. Also, this distinction is necessary to encode certain features such as delegation depth as observed in Sect. 5. To ensure termination and flexibility of our chain discovery algorithm we require that both forms of aggregation be monotonic and associative.

Definition 4.1 *An* instance of RT^R *is obtained by defining a complete lattice* $(\mathcal{K}, \preccurlyeq)$*, where* $\mathcal{K}$ *is a set of* risks *and* $\preccurlyeq$ *is a decidable* risk order relation*. We let* κ *and* K *range over elements and subsets of* $\mathcal{K}$ *respectively, and let* $\top$ *and* $\bot$ *denote the top and bottom elements of the lattice. The instantiation also includes two associative and monotonic* aggregation *operators:* risk aggregation $\oplus$*, and* intersection aggregation $\otimes$*.*

As an example, we can instantiate RT^R with the usual $\leq$ ordering on natural numbers plus ω as an upper bound, and specify that addition is the sole aggregation operation.

Example 4.1 *Let* $\mathcal{N} \triangleq (\mathbb{N} \cup \{\omega\}, \leq)$ *where* $\leq$ *is the usual relation on natural numbers extended such that* $n \leq \omega$ *for all* $n \in \mathbb{N}$*. Let* $\oplus = \otimes = +$*, with* $+$ *extended such that* $n + \omega = \omega$ *for all* $n \in \mathbb{N} \cup \{\omega\}$*. These definitions together specify an instance of* RT^R*.*

The basis of risk assessment is the association of risk with individual credentials, since credentials are the fundamental assertions used in authorization decisions. Thus, credentials in RT^R are of the form $A.r \xleftarrow{\kappa} f$ where κ is the risk associated with the credential. The manner in which risks are assigned to credentials is left unspecified, but we discuss some concrete possibilities in Sect. 3. In essence, the aggregation of risks associated with credentials used in an authorization decision constitutes the risk of that decision.

Definition 4.2 *Given an instance of* RT^R*, we let* $\mathcal{C}$ *range over finite sets of* credentials*, and* c *range over individual credentials of the form* $A.r \xleftarrow{\kappa} f$*. We refer to credential forms type 1 through 4 as in Sect. 2.*

Formally, the semantics of RT^R associates risk κ with the membership of entities B in roles $A.r$. Thus, the meaning of roles $A.r$ are finite sets of pairs of the form (B, κ), called *RiskAssessment*s. Note that any *RiskAssessment* may associate more than one risk with any entity, i.e. there may exist $(A, \kappa_1), (A, \kappa_2) \in R$ such that $\kappa_1 \neq \kappa_2$. This reflects the possibility of more that one path to role membership, each associated with incomparable risk. Taking the glb of incomparable risks in risk assessments as a semantic basis of RT^R is unsound, since the glb will assess a lesser risk of membership than is in fact possible to obtain through any path.

Definition 4.3 *Sets of type RiskAssessment, denoted R, are finite collections of pairs of the form (B, κ). For any $\mathcal{A} \subseteq$ Entities, the type RiskAssessment$(\mathcal{A})$ denotes the set of risk assessments R such that $(A, \kappa) \in R$ implies $A \in \mathcal{A}$.*

Of course, if a risk assessment associates two distinct but comparable risks with a given role membership, the lesser of the two can be taken as representative. In other words, risk assessments can be taken as a set of lower bound constraints on risk in authorization. Not only is this idea logically appealing, but it also ensures that the semantics of RT^R is constructive in the presence of cyclic credentials. Otherwise, if every path with different risk were explicitly represented, the monotonic aggregation of risk along a cycle could generate an infinitely large set of increasingly expensive risk assessments for a given role membership. In Sect. 6 we show that this semantic specification does not disallow the use in practice of a higher-than-minimal authorization path. We define an equivalence relation to capture the idea, inducing equivalence classes identified by *canonical* risk assessments containing no (A, κ_1) and (A, κ_2) where $\kappa_1 \preccurlyeq \kappa_2$. Furthermore, the canonical representation of any assessment R, denoted $\hat{R}$, is decidable since assessments are finite and $\preccurlyeq$ is decidable.

Definition 4.4 *Equivalence classes of risk assessments are obtained by the following axiom schema:*

$$R \cup \{(A, \kappa_1), (A, \kappa_2)\} = R \cup \{(A, \kappa_1)\} \qquad \text{where } \kappa_1 \preccurlyeq \kappa_2$$

We call canonical *those risk assessments R where there exist no $(A, \kappa_1), (A, \kappa_2) \in R$ such that $\kappa_1 \preccurlyeq \kappa_2$, and observe that any equivalence class of risk assessments has a unique canonical form. We extend the ordering $\preccurlyeq$ to risk assessments as follows:*

$$R_1 \preccurlyeq R_2 \iff \forall (A, \kappa_1) \in \hat{R}_1 . \exists \kappa_2 . (A, \kappa_2) \in \hat{R}_2 \wedge \kappa_1 \preccurlyeq \kappa_2$$

Hereafter we restrict our consideration to canonical risk assessments without loss of generality.

The semantic definition is defined via several auxiliary operations. Canonical risk assessments are combined by taking their canonical union:

Definition 4.5 *The* canonical union *of risk assessments is denoted $\uplus$, i.e. define $R_1 \uplus R_2 = \hat{R}$, where $R = R_1 \cup R_2$.*

Entire risk assessments may be aggregated together, or incremented by some risk. We therefore extend aggregation to risk assessments for notational convenience as follows. Note that these operations preserve canonical form of risk assessments.

Definition 4.6 *Letting $\star$ range over $\{\oplus, \otimes\}$, aggregation is extended to risk assessments as follows:*

$$\begin{array}{rcl} R \star \kappa & \triangleq & \hat{R}' \text{ where } R' = \{(A, \kappa' \star \kappa) \mid (A, \kappa') \in R\} \\ R_1 \star R_2 & \triangleq & \hat{R}' \text{ where } R' = \{(A, \kappa_1 \star \kappa_2) \mid (A, \kappa_1), (A, \kappa_2) \in R_1 \times R_2\} \end{array}$$

Now we can specify the semantics of RT^R, via interpretations that map roles to risk assessments.

Definition 4.7 *A* role interpretation *is a total function of type:*

$$\mathit{Role} \to \mathit{RiskAssessment}$$

Letting f and g be role interpretations, define:

$$f \preccurlyeq g \quad \iff \quad f(A.r) \preccurlyeq g(A.r) \text{ for all roles } A.r$$

An interpretation is taken to be a solution to a set of credentials if it maps each role to the "right" risk assessment, given the intended meaning of the given credentials. This meaning is characterized by the functionals bounds and expr, defined in Fig. 1, and the following definition.

Definition 4.8 (Semantics of $\mathbf{RT}^R$) *Given a set $\mathcal{C}$ of RT^R credentials, the* solution $\mathcal{S}_\mathcal{C}$ of $\mathcal{C}$ *is the least role interpretation* rmem *such that* bounds[rmem] $\preccurlyeq$ rmem*, where* bounds *and the auxiliary function* expr*, mapping interpretations to interpretations, are defined in Fig. 1.*

We discuss several extended examples of the system in Sect. 5. Here is a brief example illustrating the semantics.

Example 4.2 *Assume given the instance of RT^R as defined in Example 4.1, and let $\mathcal{C}$ consist of the following:*

$$A.r_0 \xleftarrow{2} B.r_3 \qquad A.r_0 \xleftarrow{1} C.r_1.r_2 \qquad C.r_1 \xleftarrow{3} D \qquad B.r_3 \xleftarrow{4} E$$

$$D.r_2 \xleftarrow{0} F$$

Then $\mathcal{S}_\mathcal{C}$ is the interpretation mapping every role to $\varnothing$, except:

$$\mathcal{S}_\mathcal{C}(C.r_1) = \{(D, 3)\} \qquad \mathcal{S}_\mathcal{C}(B.r_3) = \{(E, 4)\} \qquad \mathcal{S}_\mathcal{C}(D.r_2) = \{(F, 0)\}$$

$$\mathcal{S}_\mathcal{C}(A.r_0) = \{(E, 6), (F, 4)\}$$

$$
\begin{aligned}
\text{bounds}[\text{rmem}](A.r) &= \biguplus_{A.r \xleftarrow{\kappa} e \in \mathcal{C}} \text{expr}[\text{rmem}](e) \oplus \kappa \\
\\
\text{expr}[\text{rmem}](B) &= \{(B, \bot)\} \\
\text{expr}[\text{rmem}](A.r) &= \text{rmem}(A.r) \\
\text{expr}[\text{rmem}](A.r_1.r_2) &= \biguplus_{(B,\kappa) \in \text{rmem}(A.r_1)} \text{rmem}(B.r_2) \oplus \kappa \\
\text{expr}[\text{rmem}](f_1 \cap \cdots \cap f_n) &= \bigotimes_{1 \leq i \leq n} \text{expr}[\text{rmem}](f_i)
\end{aligned}
$$

Figure 1: RT^R semantic functions

4.2 Existence of a Solution

The semantics of RT^R just defined is meaningful only if any credential set $\mathcal{C}$ has a solution. We establish this by an inductive construction that converges to a fixpoint, obtained by iterating bounds over an initial empty risk assessment.

Definition 4.9 *A* partial role interpretation over $\mathcal{C}$ *is a function of type:*

$$\mathit{Roles}(\mathcal{C}) \rightarrow \mathit{RiskAssessment}(\mathit{Entities}(\mathcal{C})))$$

The family of partial role interpretations over $\mathcal{C}$ *denoted* $\{\text{rmem}_i\}_{i \in \mathbb{N}}$ *is defined inductively by taking* $\text{rmem}_0(A.r) = \varnothing$ *for every role* $A.r$*, and letting:*

$$\text{rmem}_{i+1}(A.r) = \text{bounds}[\text{rmem}_i](A.r)$$

for every $A.r$*.*

The argument for existence of a solution proceeds by showing that this construction converges to a fixpoint, which is clearly a solution of $\mathcal{C}$.

To show that the construction converges to a fixpoint, we observe that bounds is monotonic over a lattice of partial role interpretations over given $\mathcal{C}$. The lattice is induced by a relation that is similar to $\preccurlyeq$, but with an important difference. In each iteration, the solution is built up by adding new elements (A, κ) to the interpretations of roles. However, the use of canonical union ensures any such element is added only if κ is lesser than or incomparable with existing elements of the solution, rather than greater than as would be possible if bounds were monotonic in $\preccurlyeq$. Hence, we define an appropriate ordering denoted $\trianglelefteq$:

Definition 4.10 *Define* $\trianglelefteq$ *as a relation on risk assessments:*

$$R_1 \trianglelefteq R_2 \iff \forall (A, \kappa_1) \in \hat{R_1}.\exists \kappa_2.(A, \kappa_2) \in \hat{R_2} \wedge \kappa_2 \preccurlyeq \kappa_1$$

The relation is extended pointwise to partial role interpretations, i.e. given that f and g are partial role interpretations, define $f \trianglelefteq g \iff \forall A.r \in Dom(f).f(A.r) \trianglelefteq g(A.r)$.

We observe that $\trianglelefteq$ is a partial order, since $\preccurlyeq$ is:

Lemma 4.1 *The relation $\trianglelefteq$ is a partial order on both risk assessments and partial role interpretations.*

It is essential to show that bounds is monotonic over this ordering of partial role interpretations. The property is immediate, since both bounds and expr are defined via operations that preserve canonical forms when combining risk assessments, i.e. $\uplus$, and $\oplus$ and $\otimes$.

Lemma 4.2 *The function* bounds *is monotonic in $\trianglelefteq$ over partial role interpretations.*

Now we show that $\trianglelefteq$ induces a complete lattice structure on risk assessments and partial role interpretations. This allows us to assert the existence of a solution, since this result and monotonicity of bounds in $\trianglelefteq$ implies that $\{\mathrm{rmem}_i\}_{i\in\mathbb{N}}$ converges to a fixpoint.

Lemma 4.3 *Given finite $\mathcal{C}$, the posets*

$$(RiskAssessment(Entities(\mathcal{C})), \trianglelefteq)$$
$$\textit{and}$$
$$(Roles(\mathcal{C}) \to RiskAssessment(Entities(\mathcal{C})), \trianglelefteq)$$

are complete lattices.

Proof. We begin by showing that the first poset is a complete lattice. Given $\mathcal{A} = Entities(\mathcal{C})$ and $\mathcal{R} \subseteq RiskAssessment(\mathcal{A})$. For each entity $A \in \mathcal{A}$ let $K_A = \{\kappa \mid \exists R \in \mathcal{R}.(A, \kappa) \in R\}$ and let κ_A be the glb of K_A, which must exist since we require risk orderings to be complete lattices. Let $R_{\mathcal{R}} = \{(A, \kappa_A) \mid A \in \mathcal{A}\}$. Clearly $R_{\mathcal{R}}$ is an element of $RiskAssessment(\mathcal{A})$, and is a lub of $\mathcal{R}$. The existence of a glb for $\mathcal{R}$ follows dually. The second poset is clearly also a complete lattice, since the ordering in that poset is just the pointwise extension of $\trianglelefteq$ for risk assessments. □

Of course, a remaining issue is whether $\{\mathrm{rmem}_i\}_{i\in\mathbb{N}}$ converges to a fixpoint in a *finite* number of iterations. This is obviously true if we restrict our consideration to finite risk domains $\mathcal{K}$, since any set of partial role interpretations forms a finite lattice under $\trianglelefteq$ by the preceding result. In case $\mathcal{K}$ is infinite, the situation is complicated somewhat. However, any finite set of credentials $\mathcal{C}$ can only be combined in a finite number of ways to obtain role membership, yielding a finite number of possible membership risks, unless credentials are cyclic, but encountering cycles only obtains increased risk of existing memberships due to monotonicity of risk aggregation. These increased risks will be ignored by bounds since it preserves canonical forms of risk assessments. Hence, the lattice

of partial role interpretations relevant to a sequence $\{\text{rmem}_i\}_{i\in\mathbb{N}}$ is effectively finite, even if $\mathcal{K}$ is infinite. The details of the argument are mostly tedious and are omitted here for brevity. We assert:

Lemma 4.4 *Given finite $\mathcal{C}$, there exists finite n such that* rmem_n *is a fixpoint of* bounds.

A solution is than effectively constructed from this fixpoint. Specifically, given finite $\mathcal{C}$, let rmem_n be the least fixpoint of the sequence $\{\text{rmem}_i\}_{i\in\mathbb{N}}$, and define:

$$\begin{array}{lcll} \mathcal{S}_{\mathcal{C}}(A.r) & = & \text{rmem}_\omega(A.r) & A.r \in \mathit{Roles}(\mathcal{C}) \\ \mathcal{S}_{\mathcal{C}}(A.r) & = & \varnothing & A.r \notin \mathit{Roles}(\mathcal{C}) \end{array}$$

4.3 Thresholds and Constrained Solutions

In an authorization setting that takes risk into account, acceptable levels of risk should be specifiable as a component of policy. Furthermore, these levels should be specifiable on a per-role basis, to allow a fine-grained view of risk tolerance. We believe this level of flexibility is desirable, since it allows the disparate entities maintaining particular roles to assign different levels of risk tolerance, or allows a local authorizer to associate different levels of risk with different entities, both of which are likely scenarios. To formalize this in our model, we introduce thresholds Θ, which are mappings from roles to risks.

Definition 4.11 *A* threshold Θ *is a total function of type:*

$$\mathit{Role} \to \mathcal{K}$$

We write $\Theta_\top$ to denote the threshold such that $\Theta_\top(A.r) = \top$ for all $A.r$, and we write $\Theta[A.r : \kappa]$ to denote Θ' such that $\Theta'(A.r) = \kappa$, and $\Theta'(B.r') = \Theta(B.r')$ for all $B.r' \neq A.r$.

Incorporating thresholds into the model is not simply a matter of eliminating those elements of role solutions that exceed the given threshold. The problem is that some role memberships that are not explicitly constrained by a threshold may nevertheless be eliminated by it, due to a dependence on other constrained role memberships; see Example 4.3 below. Therefore, we make a slight modification to bounds, to take into account threshold constraints.

Definition 4.12 *Given some Θ, the* Θ-constrained solution $\mathcal{S}^{\Theta}_{\mathcal{C}}$ of $\mathcal{C}$ *is the least role interpretation* rmem *such that:*

$$\text{bounds}_\Theta[\text{rmem}] \preccurlyeq \text{rmem}$$

where for all $A.r$:

$$\text{bounds}_\Theta[\text{rmem}](A.r) = \{(B,\kappa) \mid (B,\kappa) \in \text{bounds}[\text{rmem}](A.r) \text{ and } \kappa \preccurlyeq \Theta(A.r)\}$$

The existence of threshold constrained solutions is established by nearly the same argument as that given in Sect. 4.2.

Lemma 4.5 *$\mathcal{S}^{\Theta}_{\mathcal{C}}$ exists for arbitrary Θ and $\mathcal{C}$.*

Here is a brief example illustrating threshold constrained solutions, highlighting non-local effects of constraints on role membership. Note that even though all members of $A.r_0$ in the full solution are below the given threshold, one of them will be eliminated in the threshold constrained solution, due to its dependence on the role $B.r_3$. That is, since $(E, 4)$ is eliminated from $B.r_3$'s solution by the given threshold constraint, $(E, 6)$ cannot be established as part of $A.r_0$'s solution.

Example 4.3 *Assume given the instance of RT^R and credential set $\mathcal{C}$ specified in Example 4.2. Define:*

$$\Theta = \Theta_{\top}[A.r_0 : 10][B.r_3 : 3]$$

Then $\mathcal{S}^{\Theta}_{\mathcal{C}}$ is the interpretation mapping every role to $\varnothing$, except:

$$\mathcal{S}^{\Theta}_{\mathcal{C}}(C.r_1) = \{(D, 3)\} \qquad \mathcal{S}^{\Theta}_{\mathcal{C}}(D.r_2) = \{(F, 0)\} \qquad \mathcal{S}^{\Theta}_{\mathcal{C}}(A.r_0) = \{(F, 4)\}$$

5 Examples

In this section we present some instances of RT^R that illustrate how the system is used, and how it is able to capture a variety of risk management schemes.

5.1 Bound-of-Risks

In [12], an information flow security model is presented where all static data is assigned to a security class. Security classifications of variables are then assigned based on the combination of security classes of data flowing into those variables, as determined by an abstract program interpretation. Security classes are identified by elements in a complete lattice, where "class-combination" is defined as the lub of combined classes.

We propose that an adaptation of this model is useful in the context of authorization risk assessment. We do not propose an abstract interpretation of authorization, incorporating some form of "may-analysis", but rather a purely dynamic authorization and risk assessment model, so in this sense we differ from the model proposed in [12]. Nevertheless, we may adopt the use of least upper bounds as a "class-combination" mechanism– in our terminology, risk aggregation– that assesses the risk of any authorization decision as the least upper bound of risks associated with all credentials used in the decision. Thus, we define each of $\oplus$ and $\otimes$ as the lub operator on risks in the given partial ordering.

Consider a risk ordering where three classifications $\mathcal{K} = \{low, medium, high\}$ are defined, and the following relations are imposed:

$$low \preccurlyeq medium \preccurlyeq high$$

Imagine also that an online vendor called *Store* maintains a purchasing policy whereby representatives of the *Acme* corporation have *buyer* power only if they are both employees and official purchasers. Since this policy is maintained locally, it is associated with a *low* risk of usage, hence *Store* could specify:

$$Store.buyer \xleftarrow{low} Acme.purchaser \cap Acme.employee$$

Imagine further that *Ed* attempts to make a purchase from *Store*, providing certificates claiming *employee* and *purchaser* status. However, if we assume that these certificates can possibly be faked, or that role membership within the *Acme* corporation has a volatile status, higher risk can be assigned to these certificates:

$$Acme.employee \xleftarrow{medium} Ed \qquad Acme.purchaser \xleftarrow{high} Ed$$

We also assume that a less risky path of establishing *Ed*'s membership in the *Acme.purchaser* role is through a *manager* certificate obtained directly from the issuer *Personnel*, and via *Acme*'s own policy specifying *purchaser* power for all *manager*s:

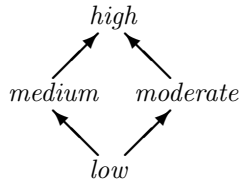
$$Acme.purchaser \xleftarrow{low} Personnel.manager \qquad Personnel.manager \xleftarrow{low} Ed$$

Although using *Ed*'s certificate asserting his membership in the *Acme.purchaser* role will incur a *high* risk, because of the less risky path to this relation, the risk assessment of this set of credentials will find that establishing *Ed*'s membership in the *Store.buyer* role requires a lower bound of *medium* risk. The solution for this set of credentials is as follows:

$$\begin{aligned} Store.buyer &: \{(Ed, medium)\} \\ Acme.employee &: \{(Ed, medium)\} \\ Acme.purchaser &: \{(Ed, low)\} \\ Personnel.manager &: \{(Ed, low)\} \end{aligned}$$

Of course, in certain cases it may be preferable to use the certificate *Ed* provides, instead of going through *Personnel*– if wait times for distributed communication with that node are prohibitively long, for example. In this case it should be specified that a *high* level of risk will be tolerated in the credential chain. This is accomplished by defining an appropriate threshold. Although the semantics do not explicitly list a *high* risk membership in *Store.buyer*, it does exist, and may be used in practice as discussed in Sect. 6.

Returning to the example, for the purposes of illustration we imagine that the risk ordering is extended with an element *moderate*, that is incomparable with *medium*, inducing the lattice:



We also imagine that *Store* has cached an old certificate, establishing *Ed*'s membership in the *Acme.employee* role with *moderate* risk:

$$Acme.employee \overset{moderate}{\longleftarrow} Ed$$

In this case, since *moderate* and *medium* are incomparable, the risk assessment will reflect that *Ed*'s membership in the *Store.buyer* and *Acme.employee* roles can be established via two paths with incomparable risk:

$$\begin{aligned} Store.buyer &: \{(Ed, medium), (Ed, moderate)\} \\ Acme.employee &: \{(Ed, medium), (Ed, moderate)\} \end{aligned}$$

5.1.1 Agreement Decreases Risk

In case a PGP-like scheme of allowing agreement to reduce risk is desired, intersection aggregation can be modified appropriately without having to change risk aggregation. For example, returning to the risk ordering comprising just $\{high, medium, low\}$, and specifying that $\otimes$ be commutative, we could define:

$$low \otimes low = low \qquad low \otimes medium = low \qquad low \otimes high = low$$

$$medium \otimes medium = low \qquad medium \otimes high = medium \qquad high \otimes high = medium$$

Intersection aggregation thus defined is both monotonic and associative as can easily be checked. Given these definitions and the following credentials:

$$Store.buyer \overset{low}{\longleftarrow} Acme.purchaser \cap Acme.employee$$

$$Acme.employee \overset{medium}{\longleftarrow} Ed \qquad Acme.purchaser \overset{high}{\longleftarrow} Ed$$

role memberships will reflect the reduction in risk achieved via intersection:

$$\begin{aligned} Store.buyer &: \{(Ed, medium)\} \\ Acme.employee &: \{(Ed, medium)\} \\ Acme.purchaser &: \{(Ed, high)\} \end{aligned}$$

5.2 Sum-of-Risks

An alternative to the bound-of-risks model is a sum-of-risks model, where credentials are assigned numeric risk values and the total risk for any authorization decision is the sum of all risks associated with the credentials used in the decision. Thus, we take the risk ordering in this model to be the lattice of natural numbers up to ω induced by $\leq$, and we take $\oplus$ and $\otimes$ to be addition. This model is useful in case risk is considered additive, or in case the number of credentials used in an authorization decision is an element of risk, the more the riskier.

Imagining a similar situation as above, the following risks could be assigned, where 1 is considered "not risky" and 4 is considered "risky":

$$Store.buyer \xleftarrow{1} Acme.purchaser \cap Acme.employee \qquad Acme.employee \xleftarrow{3} Ed$$

$$Acme.purchaser \xleftarrow{4} Ed \qquad Acme.purchaser \xleftarrow{2} Personnel.manager$$

$$Personnel.manager \xleftarrow{3} Ed$$

Note that *Ed*'s certificate claiming membership in the role *Acme.purchaser* is still assigned higher risk than both the certificate establishing his *manager* status and the certificate establishing *purchaser* rights for *manager*s. However, the sum-of-risks model will still ascertain that the use of *Ed*'s certificate will be the least risky way to establish his membership in the *Store.buyer* role. The solution of the given credentials will comprise the following risk assessments:

$$\begin{aligned} Store.buyer &: \{(Ed, 8)\} \\ Acme.employee &: \{(Ed, 3)\} \\ Acme.purchaser &: \{(Ed, 4)\} \\ Personnel.manager &: \{(Ed, 3)\} \end{aligned}$$

If a pure count of credentials used in authorization is the basis of risk assessment, this model can be formally obtained in the sum-of-risks model by associating risk 1 with every credential.

Just as in the bound-of-risks model, intersection aggregation can be modified to interpret agreement as reducing risk. For example, $\otimes$ can be defined as the average of its operands, or some other fraction of their sum.

5.3 Delegation Depth and Width

In RT_0, type 2 credentials allow delegation of authority across domain boundaries. For example, the credential $A.r_0 \longleftarrow B.r_1$ allows the entity A to delegate authority to define a role within its namespace to the entity B, which may denote a different security domain. Furthermore, B is able to delegate authority to define $A.r_0$ to another entity C via the credential $B.r_1 \longleftarrow C.r_2$. However, as observed by various authors, trust is not necessarily transitive, so that A may wish to prevent B from from further delegation of authority to define $A.r_0$ to C or anyone else. This sort of control might also be more fine-grained, in that A might wish to allow one level of delegation of authority to define $A.r_0$, from B to C for example, but no further, so that C should not be allowed to delegate authority to define $A.r_0$ to another entity. The idea clearly generalizes to delegations of arbitrary depth.

The system RT as originally conceived [19] does not allow delegation depth to be restricted in this way. An extension of RT_0 called RT_+ [16] was proposed to allow expression of delegation depth policies. Here, we show how to specify similar delegation depth control policies in an instance of RT^R.

Assume given $\mathcal{N}$ as defined in Example 4.1 as a risk ordering. The encoding is then based on a numeric representation of depth for individual credentials. We consider the specification of risk values and aggregation operations by considering each credential type in turn. Type 1 credentials define a role membership directly within a namespace, so the delegation depth associated with those credentials is 0. Hence, all credentials match the schema:

$$A.r \xleftarrow{0} B$$

Type 2 credentials do allow delegation of role definition authority, so type 2 credentials have a delegation depth of 1 if the delegation crosses namespaces. Otherwise the depth of a type 2 credential is 0. Hence, all type 2 credentials match the schemas:

$$A.r_1 \xleftarrow{0} A.r_2$$
$$A.r_1 \xleftarrow{1} B.r_2 \quad A \neq B$$

Naturally, risk aggregation is defined as addition, so that depth is added as credential edges are crossed:

$$\kappa_1 \oplus \kappa_2 \triangleq \kappa_1 + \kappa_2$$

Type 3 credentials allow indirect delegation. Recall that an RT_0 credential of the form $A.r_1 \longleftarrow B.r_2.r_3$ allows us to assert that $C \in A.r_1$ if $D \in B.r_2$ and $C \in D.r_3$. In our view, this means that B is thereby capable of delegating to D the authority to define $A.r_1$, and therefore also A delegates to B the authority to define the role. So firstly, this means that type 3 credentials should be assigned a delegation depth of 1 if A and B are distinct namespaces, and 0 otherwise:

$$A.r_1 \xleftarrow{0} A.r_2.r_3$$
$$A.r_1 \xleftarrow{1} B.r_2.r_3 \quad A \neq B$$

Secondly, given some credential $A.r_1 \longleftarrow B.r_2.r_3$, linking aggregation should sum the delegation depth associated with determining $D \in B.r_2$ with the delegation depth associated with determining $C \in D.r_3$ to determine $C \in A.r_1$. Hence, linking aggregation is also defined as addition:

$$\kappa_1 \oplus \kappa_2 \triangleq \kappa_1 + \kappa_2$$

We note that our model differs from the RT_+ model with respect to risk aggregation and depth of type 3 credentials. In that paper, type 3 credentials are always assigned depth 1, and the depth associated with determining $D \in B.r_2$ is ignored in linking aggregation. The authors do not clarify the reasons for these choices, but we believe they are flawed. Regarding the depth of type 3 credentials, consider the following RT_0 example:

$$A.r_1 \longleftarrow A.r_2.r_3 \qquad A.r_2 \longleftarrow A.r_3 \qquad A.r_3 \longleftarrow B$$

These credentials allow us to establish that $B \in A.r_1$ with no delegation of authority, whereas the scheme in RT_+ would assign a delegation depth of 1. Multiple links within the same namespace would extend the spurious depth, allowing arbitrarily large overestimates of delegation depth.

Underestimates of delegation depth are also possible given the scheme in RT_+, as follows. Consider the following credentials, where $C_1.s_1 \longleftarrow \cdots \longleftarrow C_n.s_n$ denotes $n-1$ type 2 credentials in the obvious manner.

$$A.r_1 \longleftarrow B.r_2 \qquad B.r_2 \longleftarrow C_1.s_1 \qquad C_1.s_1 \longleftarrow \cdots \longleftarrow C_n.s_n \qquad C_n.s_n \longleftarrow D$$

Assuming that A, B, D, and $C_1, \ldots, C_n$ all denote distinct namespaces, establishing $D \in A.r_1$ involves a delegation depth of $n+1$. However, there is an easy attack that B can use to reduce an $n+1$ delegation depth to a depth of 2: B could eliminate $B.r_2 \longleftarrow C_1.s_1$, and add the following credentials:

$$B.r_2 \longleftarrow B.r_3.r_4 \qquad B.r_3 \longleftarrow B \qquad B.r_4 \longleftarrow C_1.s_1$$

In contrast, our monotonicity requirements on aggregation prevents such an attack.

In intersection roles, depth of components should not be summed, instead each component should be considered an independent "branch". Intersection aggregation is therefore defined as the max height of its operands:

$$\kappa_1 \otimes \kappa_2 \triangleq max(\kappa_1, \kappa_2)$$

Like the other credential forms, type 4 credentials are assigned a depth risk on the basis of whether they cross domain boundaries. Since role expressions of any form may be intersected, we need to specify the *subjects* of role expressions:

$$\begin{aligned} subjects(A) &= \varnothing \\ subjects(A.r) &= \{A\} \\ subjects(A.r_1.r_2) &= \{A\} \\ subjects(f_1 \cap \cdots \cap f_n) &= subjects(f_1) \cup \cdots \cup subjects(f_n) \end{aligned}$$

and we assign depth risks to type 4 credentials as follows:

$$A.r \xleftarrow{0} f_1 \cap \cdots \cap f_n \quad \text{if } subjects(f_1 \cap \cdots \cap f_n) \subseteq \{A\}$$

$$A.r \xleftarrow{1} f_1 \cap \cdots \cap f_n \quad \text{if } subjects(f_1 \cap \cdots \cap f_n) \nsubseteq \{A\}$$

5.3.1 Controlling Delegation Width

The designers of SPKI/SDSI provided a simple scheme of boolean control for delegation, expressible in our model by specifying a threshold Θ such that $\Theta(A.r) = \omega$ if it was desired that authority over the role $A.r$ could be delegated, and $\Theta(A.r) = 0$ if not. The complexity of full integer depth control as in our general model was not adopted, in part because depth control does not

control delegation *width*, hence does not address problems of proliferation [13]. Any given principal may delegate authority to an unlimited number of other principals:

$$A.r \longleftarrow B_1.s_1 \qquad \cdots \qquad A.r \longleftarrow B_n.s_n \qquad A \neq B_1 \neq \cdots \neq B_n$$

and notions of delegation depth provide no control on the size of n. Furthermore, this "fanning out" can continue more than one level deep in the credential chain, in that authority over any role $B_i.r_i$ may in turn be delegated to an arbitrary number of principals, and so on.

However, forms of width control can be obtained by appropriate instantiations of RT^R. In particular, limits can be placed on the sets of principals to which authority can be delegated for a given role definition. Letting $\mathcal{P}$ be the set of principals, we take the set of risks $\mathcal{K}$ to be the powerset of $\mathcal{P}$, risk ordering $\preccurlyeq$ to be set containment, and both forms of aggregation $\{\oplus, \otimes\}$ to be set union. Credential risks are then defined as the set of subjects in the credential, so that all credentials adhere to the following schema:

$$B.s \xleftarrow{subjects(f)} f$$

Now, suppose that A wished to specify that only principals in the set $\{B, C, D\}$ should be allowed any sort of authority over the role $A.r$. In this case a threshold Θ would be defined such that $\Theta(A.r) = \{B, C, D\}$, and suppose that Θ maps all other roles to $\mathcal{P}$ for the purposes of the example. Hence, given the following set of credentials:

$$A.r \xleftarrow{\{B\}} B.s \qquad B.s \xleftarrow{\{C\}} C.q \qquad B.s \xleftarrow{\{E\}} E.q \qquad C.q \xleftarrow{\varnothing} E \qquad E.q \xleftarrow{\varnothing} D$$

The Θ-constrained solution is as follows:

$$\begin{array}{rcl} E.q & : & \{(D, \varnothing)\} \\ C.q & : & \{(E, \varnothing)\} \\ B.s & : & \{(D, \{E\}), (E, \{C\})\} \\ A.r & : & \{(E, \{B, C\})\} \end{array}$$

Note that D cannot be established as a member of $A.r$, since it can be so only under the authority of E which is disallowed by the width threshold for $A.r$.

6 RT^R Distributed Credential Chain Discovery

In this section we discuss an algorithm for authorization with risk in a distributed environment, where not all credentials are required to be known a priori. Rather, non-local certificates may be retrieved automatically to establish new credentials if necessary. Following RT credential chain discovery [21], our technique is to characterize credential sets graph-theoretically, except that

our credential graphs are risk-weighted multigraphs, to accommodate risk assessments. Credential graphs are shown to be a full abstraction of solutions as in Definition 4.8, and the RT^R discovery algorithm is shown to correctly reconstruct credential graphs.

In addition to theoretical correctness, our chain discovery algorithm has two important practical features:

1. The algorithm need not verify a role membership in a risk-optimal fashion, but rather is parameterized by a threshold, specifying maximum tolerable risks for role memberships.

2. The discovery procedure is *directed*, in the sense that it is aborted along search paths whose risk overruns the maximum threshold.

The first feature allows end-users to modulate tolerable levels of risk in authorization. The second feature reaps any efficiency benefits intended by associating risks with credentials, as high risk may be associated with high expense, e.g. if risks are wait times.

6.1 Credential Graphs

We begin by defining an interpretation of credential sets $\mathcal{C}$ as a credential graph. More precisely, a set of credentials is interpreted as a weighted multigraph, where nodes are role expressions, edges are credentials, and weights are risks. Authorization is implemented by determining reachability, via risk weighted paths, where the aggregation of edge risk along the path is the risk of authorization. Reachability is predicated on simple paths, since traversing cycles can only increase risk due to monotonicity of risk aggregation, and any path with a cycle would otherwise generate an infinite number of risk weighted paths. Allowing the latter would preclude a constructive definition of credential graphs, since chains are distinguished by risk and cycle traversal increases risk monotonically.

Definition 6.1 (Risk weighted credential chains) *Letting $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a multigraph with nodes $f \in \mathcal{N}$ and edges $f_1 \xrightarrow{\kappa} f_2 \in \mathcal{E}$ weighted by elements κ of a given risk ordering, the pair:*

$$((f_1, \ldots, f_n), \kappa_1 \oplus \cdots \oplus \kappa_{n-1})$$

is a risk weighted path *in $\mathcal{G}$ iff for all $i \in [1..n-1]$, there exists $f_i \xrightarrow{\kappa_i} f_{i+1} \in \mathcal{E}$. A weighted path $((f_1, \ldots, f_n), \kappa)$ is* simple *iff no node is repeated in $(f_1, \ldots, f_n)$. We write $f \xrightarrow{\kappa}\!\!\!\!\rightarrow f'$, pronounced "there exists a credential chain from f to f' with risk κ", iff $((f, \ldots, f'), \kappa)$ is a simple risk weighted path. We write $f \xrightarrow{\kappa}\!\!\!\!\rightarrow f' \in \mathcal{G}$ iff $f \xrightarrow{\kappa}\!\!\!\!\rightarrow f'$ holds given $\mathcal{G}$.*

The ability to isolate different weighted paths between the same nodes in a graph benefits our larger goals. In particular, while credential solutions in the sense defined in Sect. 4 explicitly reflect only minimal risks associated with role membership, the abstraction of paths allows a formal designation of role

memberships with comparable but unequal risk– given some graph $\mathcal{G}$, it may be the case that $A \xrightarrow{\kappa} B.r \in \mathcal{G}$ and $A \xrightarrow{\kappa'} B.r \in \mathcal{G}$ where $\kappa \preccurlyeq \kappa'$ and $\kappa \neq \kappa'$. The ability to establish role membership with non-minimal risk is an important feature of our distributed chain discovery algorithm defined below.

The definition of credential graphs is founded on the definition of risk weighted chains, since edges derived from linked and intersection credentials are supported by them.

Definition 6.2 (Credential graph) *Given finite $\mathcal{C}$, its* credential graph *is a weighted multigraph $\mathcal{G}_\mathcal{C} = (\mathcal{N}_\mathcal{C}, \mathcal{E}_\mathcal{C})$, where:*

$$\mathcal{N}_\mathcal{C} = \bigcup_{A.r \xleftarrow{\kappa} e} \{A.r, e\}$$

And $\mathcal{E}_\mathcal{C}$ is the least set of risk-weighted edges satisfying the following closure properties:

1. *If $A.r \xleftarrow{\kappa} e \in \mathcal{C}$ then $e \xrightarrow{\kappa} A.r \in \mathcal{E}_\mathcal{C}$.*
2. *If $B.r_2, A.r_1.r_2 \in \mathcal{N}_\mathcal{C}$ and $B \xrightarrow{\kappa} A.r_1$, then $B.r_2 \xrightarrow{\kappa} A.r_1.r_2 \in \mathcal{E}_\mathcal{C}$.*
3. *If $D, f_1 \cap \cdots \cap f_n \in \mathcal{N}_\mathcal{C}$ and for each $i \in [1..n]$ there exists $D \xrightarrow{\kappa_i} f_i$, then $D \xrightarrow{\kappa} f_1 \cap \cdots \cap f_n \in \mathcal{E}_\mathcal{C}$, where $\kappa = \kappa_1 \otimes \cdots \otimes \kappa_n$.*

The definition of credential graphs can be made constructive by iterating closure over an initial edge set $\mathcal{E}^0_\mathcal{C}$:

$$\mathcal{E}^0_\mathcal{C} = \left\{ A.r \xrightarrow{\kappa} e \mid A.r \xleftarrow{\kappa} e \in \mathcal{C} \right\}$$

In rules (2) and (3), the paths predicating membership in $\mathcal{E}_\mathcal{C}$ are called *support paths*, and the edges are called *derived*. On each iteration, add a new weighted edge according to closure rule (2) or (3). Since $\mathcal{C}$ is finite, and support paths must be simple, the process will reach a fixpoint in a finite number of iterations; this fixpoint is $\mathcal{E}_\mathcal{C}$.

We observe that the characterization of credential sets $\mathcal{C}$ is sound and complete with respect to the set theoretic semantics given in the previous section. These results will form a bridge with the semantics of RT^R for establishing correctness of credential chain discovery. The statement of soundness reflects the fact that while risk assessments of credential sets express minimum risk bounds of role membership, the credential graph does not preclude reachability via paths of higher risk.

Theorem 6.1 (Soundness) *For all $B, A.r$, if $B \xrightarrow{\kappa} A.r \in \mathcal{G}_\mathcal{C}$, then $(B, \kappa') \in \mathcal{S}_\mathcal{C}(A.r)$ with $\kappa' \preccurlyeq \kappa$.*

Following [21], the result follows by a double induction; an outer induction on the number of closure iterations to obtain the graph $\mathcal{G}_\mathcal{C}$, and an inner induction on the length of the path $B \xrightarrow{\kappa} A.r$.

The statement of completeness reflects that any assessed risk is the weight of some related path in the graph:

Theorem 6.2 (Completeness) *For all $A.r$, if $(B, \kappa) \in \mathcal{S}_{\mathcal{C}}(A.r)$, then $B \xrightarrow{\kappa\!\!\rightarrow} A.r \in \mathcal{G}_{\mathcal{C}}$.*

Following [21], the result follows by induction on n, where rmem_n is a fixpoint of bounds as constructed in Sect. 4.2. Proofs for both of these results are uninteresting modifications of analogous results in [21], and we omit them here for brevity.

6.1.1 Threshold-Constrained Credential Chains

Some additional definitions are required to incorporate thresholds into the graph model of credentials. Given a particular threshold, we need to filter out those paths that exceed a given threshold. But because a threshold has role-level granularity, the filtering of a path must take into account the filtering of its role-terminated subpaths. In the case of intersection and linking nodes, we also need to take into account the paths supporting the derived edges leading into them.

Definition 6.3 *Given $\mathcal{G}_{\mathcal{C}}$. A* tail *of a chain $f_1 \xrightarrow{\kappa\!\!\rightarrow} f_2 \in \mathcal{G}_{\mathcal{C}}$ is a chain $f_1 \xrightarrow{\kappa'\!\!\rightarrow} f' \in \mathcal{G}_{\mathcal{C}}$ such that $f' \xrightarrow{\kappa''\!\!\rightarrow} f_2 \in \mathcal{E}_{\mathcal{C}}$ and $\kappa = \kappa' \oplus \kappa''$. The chains $B \xrightarrow{\kappa\!\!\rightarrow} A.r_1, f \xrightarrow{\kappa''\!\!\rightarrow} B.r_2 \in \mathcal{G}_{\mathcal{C}}$* support *$f \xrightarrow{\kappa'\!\!\rightarrow} A.r_1.r_2 \in \mathcal{G}_{\mathcal{C}}$ iff $f \xrightarrow{\kappa''\!\!\rightarrow} B.r_2 \in \mathcal{G}_{\mathcal{C}}$ is a tail of $f \xrightarrow{\kappa\!\!\rightarrow} A.r_1.r_2 \in \mathcal{G}_{\mathcal{C}}$ and $\kappa = \kappa' \oplus \kappa''$. The chains $B \xrightarrow{\kappa_1\!\!\rightarrow} f_1, \ldots, B \xrightarrow{\kappa_n\!\!\rightarrow} f_n \in \mathcal{G}_{\mathcal{C}}$ are said to* support *$B \xrightarrow{\kappa\!\!\rightarrow} f_1 \cap \cdots \cap f_n \in \mathcal{G}_{\mathcal{C}}$ iff $\kappa = \kappa_1 \otimes \cdots \otimes \kappa_n$.*

Now we can define the threshold constrained credential chains as a predicate on the paths in a credential graph.

Definition 6.4 *The set of* Θ-constrained credential chains *of $\mathcal{G}_{\mathcal{C}}$, written $\mathcal{G}_{\mathcal{C}}^{\Theta}$, is a subset of $\mathcal{G}_{\mathcal{C}}$ where membership is predicated on the following inductively defined conditions:*

$$
\begin{array}{rll}
D \xrightarrow{\kappa\!\!\rightarrow} A.r \in \mathcal{G}_{\mathcal{C}}^{\Theta} & \textit{if} & \kappa \preccurlyeq \Theta(A.r) \textit{ and } D \xrightarrow{\kappa\!\rightarrow} A.r \in \mathcal{E}_{\mathcal{C}} \\
D \xrightarrow{\kappa\!\!\rightarrow} A.r \in \mathcal{G}_{\mathcal{C}}^{\Theta} & \textit{if} & \kappa \preccurlyeq \Theta(A.r) \textit{ and } \exists c \in \mathcal{G}_{\mathcal{C}}^{\Theta}.\ c \textit{ is a tail of } D \xrightarrow{\kappa\!\!\rightarrow} A.r \\
D \xrightarrow{\kappa\!\!\rightarrow} A.r_1.r_2 \in \mathcal{G}_{\mathcal{C}}^{\Theta} & \textit{if} & \exists c_1, c_2 \in \mathcal{G}_{\mathcal{C}}^{\Theta}.\ c_1, c_2 \textit{ support } D \xrightarrow{\kappa\!\!\rightarrow} A.r_1.r_2 \\
D \xrightarrow{\kappa\!\!\rightarrow} f_1 \cap \cdots \cap f_n \in \mathcal{G}_{\mathcal{C}}^{\Theta} & \textit{if} & \exists c_1, \ldots, c_n \in \mathcal{G}_{\mathcal{C}}^{\Theta}. \\
 & & \quad c_1, \ldots, c_n \textit{ support } D \xrightarrow{\kappa\!\!\rightarrow} f_1 \cap \cdots \cap f_n
\end{array}
$$

Full abstraction of the constrained graph model with respect to the set theoretic model is established via the following theorems.

Theorem 6.3 *For all $B, A.r$, if $B \xrightarrow{\kappa\!\!\rightarrow} A.r \in \mathcal{G}_{\mathcal{C}}^{\Theta}$, then $(B, \kappa') \in \mathcal{S}_{\mathcal{C}}^{\Theta}(A.r)$ with $\kappa' \preccurlyeq \kappa$.*

Theorem 6.4 *For all $A.r$, if $(B, \kappa) \in \mathcal{S}_{\mathcal{C}}^{\Theta}(A.r)$, then $B \xrightarrow{\kappa\!\!\rightarrow} A.r \in \mathcal{G}_{\mathcal{C}}^{\Theta}$.*

The results follow by a straightforward generalization of Theorem 6.1 and Theorem 6.2.

6.2 Backward Chain Discovery Algorithm `checkmem`

In centralized chain discovery, all credentials are maintained locally by assumption. In distributed chain discovery, some credentials may be retrieved over the network, in the form of certificates. This of course presupposes that the location of certificates can be determined in some manner. *Backwards* chain discovery assumes that the certificates for credentials defining a role $A.r$ are obtained from the entity A, so that chains need to be reconstructed "backwards", beginning with the governing role of an authorization decision [21]. We now define a backwards credential chain discovery algorithm `checkmem` for RT^R, possessing features described at the beginning of Sect. 6. We abstract the details of credential retrieval and risk assignment, other than its "backwards" nature, assuming that remote risk-weighted credentials can always be retrieved on demand (and cached, presumably). Forwards and mixed discovery techniques for RT are also discussed in previous work [21]; analogous techniques for RT^R can be adapted in the same way as we have adapted backward discovery here.

Much like the algorithm developed for RT_0, we define distributed chain discovery for RT^R as a credential graph reconstruction algorithm. The primary difference is that ours maintains a record of the risk encountered along partially reconstructed paths. If this "search risk" exceeds the maximum tolerable risk allowed by a given threshold, then search along that path is aborted. Monotonicity and associativity of risk aggregation ensures that any fully reconstructed paths in that direction would exceed the threshold, so aborting search in this manner is a heuristic to improve efficiency of search for threshold-constrained solutions.

In the following text, we describe an algorithm `checkmem` in English, for which we have developed a prototype implementation in OCaml described in Appendix A. The English description refers to the Appendix in key spots for clarification. The algorithm `checkmem` itself is described in Sect. 6.2.4, after preceding sections that describe data structures and auxiliary functions used by the algorithm.

6.2.1 Data Structures and Strategy

The overall strategy of the algorithm is based on graph search techniques. The central data structures of the algorithm are *nodes*, each of which are uniquely identified by a role expression f. Every node contains the following mutable component structures:

Solution. A risk assessment (Definition 4.3) for membership in the identifying role expression f. That is, a set of elements of the form (B, κ) denoting that B has been determined to be a member of the role expression f, with risk κ. Note that this use of the term "solution" is slightly abusive given Definition 4.8, since here membership is assessed for role expressions f in general and not just roles.

Search Risks. The aggregate risk-so-far that it took to search from particular

roles to the identifying role expression f. Search risks differ from solutions, in that they reflect the cost of search, not the cost of membership, and are in essence a running under-approximation of how risky it would be to establish membership in a role along the path traversed by search from that role.

Search Risk Propagaters. Functions for propagating search risks along paths already searched. Each risk propagation function living at a node f is defined with respect to a local node parameter f', such that a search edge from f to f' has been explored. Future search risks are propagated along this edge via invocation of the function.

Solution Monitors. Functions for propagating newly-discovered solutions along paths already searched. Each solution monitor function living at a node f is defined with respect to a local node parameter f', such that a credential graph edge from f to f' has been discovered. Future solutions are propagated along this edge via invocation of the function.

Search risks, propagaters, and solution monitors are discussed in more detail below. In essence, during a run of the algorithm, new nodes are created for role expressions discovered along credential paths. The components of an *initialized* node are all empty. After initialization, node solutions and search risks are updated to reflect flow through the graph structure known at that point in time. The same nodes are later mutated to reflect newly discovered graph structure, via propagater and monitor functions, that invoke each other in chains reflecting discovered graph edge structure. Nodes are specified by the type `node` in Appendix A.1.

6.2.2 Search Risks

Thresholds allow per-role specification of tolerable membership risks. Since backwards search proceeds backwards along credential graph edges, search reconstructs partial risk-weighted credential paths. Furthermore, the aggregate search risk along these paths is always a conservative approximation of role membership along these paths. Hence, as search proceeds away from a role node $A.r$, search can be aborted when the aggregation of risks encountered exceeds $\Theta(A.r)$ for given threshold Θ, since any role membership along that path is sure to exceed threshold.

Nodes f therefore maintain a set of search risks of the form $(A.r, \kappa)$ representing the aggregation of risks along a search path from $A.r$ to f. A node is searchable only if its search risk are below-threshold, in the following sense.

Definition 6.5 *A* search risk *is a tuple of the form $(A.r, \kappa)$. We let S range over sets of search risks. We say that $A.r$ is* included *in a set of search risks S iff there exists κ such that $(A.r, \kappa) \in S$. Given some Θ, a set of search risks S is* below threshold *iff for all $A.r$ included in S, there exists κ such that $(A.r, \kappa) \in S$ and $\kappa \preccurlyeq \Theta(A.r)$.*

The type of search risks is specified as `search_risk` in Appendix A.1.

6.2.3 Search Risk Propagaters and Solution Monitors

Any given node in the credential graph can be discovered more than once during search, but to ensure termination we require that any given node can only be searched once, as is usually the case in graph search algorithms. But it is important to maintain discovered graph structure, so that newly discovered information about the graph can be propagated to already-searched nodes. We use functions called *search risk propagaters* and *solution monitors*, to maintain graph structure and propagate information, specified as types `propagater` and `monitor` in Appendix A.1. These functions propagate search risks and solutions along edges. Each function lives at a node f, which is the source of the edge, and is always defined with respect to a node f' that is the sink of the edge.

To propagate search risks backward along search paths, we define side-effecting functions called search risk propagaters. This functionality is necessary since less risky search paths may be discovered to already-visited nodes. Whenever a node f is notified to add a new risk $(A.r, \kappa)$ to its search risks (as defined by function `risk_notify` in Appendix A.3), if there does not exist $(A.r, \kappa')$ already in f's search risks such that $\kappa' \preccurlyeq \kappa$, then $(A.r, \kappa)$ is added, and all of f's search risk propagaters are invoked on $(A.r, \kappa)$. Each propagater is defined with respect to a node f' denoting the node to which the search risk should be propagated.

> A *search risk propagater* for a node f' and a risk κ' is a function abstracted on search risks $(A.r, \kappa)$ that notifies f' to add $(A.r, \kappa' \oplus \kappa)$ to its search risks. The function `make_propagater` in Appendix A.3 generates search risk propagaters.

Solution monitors propagate solution elements (A, κ) forward along discovered edges, aggregating edge risks as they go; their control flow structure mimics the discovered graph structure. At the same time, they also enforce the threshold Θ supplied as a parameter to `checkmem`. Whenever a node f is notified to add a solution element (A, κ) (as defined by function `soln_notify` in Appendix A.3), the element is added to f's solution and all of f's solution monitors are applied to it, on two conditions: (1) there does not exist $\kappa' \preccurlyeq \kappa$ such that (A, κ') is already in f's solution (in which case we say it is *canonically new*), and (2) if f is a role $A.r$ then $\kappa \preccurlyeq \Theta(A.r)$. There are three classes of solution monitors, generated by functions `make_nmonitor`, `make_lmonitor`, and `make_imonitor` as specified in Appendix A.3. Each monitor form is defined with respect to a given role expression, denoting the node to which the given solution should be propagated:

1. A *node monitor* for a given node f and edge risk κ is a function abstracted on solution elements (B, κ'), that notifies f to add $(B, \kappa' \oplus \kappa)$ to its solutions.

2. A *linking monitor* for a given linked role $A.r_1.r_2$ is a function abstracted on solution elements (B, κ), that creates a node monitor for $A.r_1.r_2$ and κ, applies it to each known element of $B.r_2$'s solution, and adds it to $B.r_2$'s

solution monitors to propagate solutions yet to be discovered. Also, given all search risks $(C.s, \kappa')$ of $A.r_1.r_2$, $B.r_2$ is notified to add $(C.s, \kappa \oplus \kappa')$ to its search risks, a search risk propagater for $B.r_2$ and κ is added to $A.r_1.r_2$'s search risk propagaters, and $B.r_2$ is added to the queue if it hasn't already been.

3. An *intersection monitor* for a given intersection role $f_1 \cap \cdots \cap f_n$ is a function abstracted on solution elements (B, κ), that applies a node monitor for $f_1 \cap \cdots \cap f_n$ and $\bot$ to each element (B, κ') in the the assessment $R_1 \otimes \cdots \otimes R_n$, where each R_i is the assessment of f_i in the current solution.

6.2.4 Node Processing

Given entity A, role $B.r$ and threshold Θ, the algorithm `checkmem` reconstructs a proof graph, to check membership of A in role $B.r$ within a given threshold Θ. The algorithm maintains two mutable global data structures: a list of nodes created during execution of the algorithm, and a queue of nodes to be searched. These are specified as `nodes` and `q` in Appendix A.2. Whenever a role expression f is first encountered during search, an initialized node identified by f is added to `nodes`.

Upon invocation, the algorithm `checkmem`, defined in Appendix A.4, clears `q` and `nodes`. The node $B.r$ is initialized and added to `q` and `nodes`. While the queue contains at least one element whose search risks are below threshold Θ, such below threshold nodes are taken from the queue individually for searching. Above threshold nodes are not explored, since any solution paths that encounter them are sure to overrun the risk threshold specified for the node. But neither are they eliminated, since future search may find new below threshold paths to them. The algorithm runs until there are no below threshold nodes left in the queue, or until an element (A, κ) is added to $B.r$'s solutions with $\kappa \preccurlyeq \Theta(B.r)$, signalling that a Θ-constrained path $A \xrightarrow{\kappa\!\!\rightarrow} B.r$ has been discovered.

Whenever nodes are taken from the queue, they are processed depending on their form:

1. To process an entity A, the node A is notified to add $(A, \bot)$ as a solution to itself.

2. To process a role $A.r$, the credentials defining $A.r$ are retrieved. For each such credential $A.r \xleftarrow{\kappa} f$, a node monitor for $A.r$ and κ is created, is applied to all of f's known solutions, and is added to f's solution monitors for propagating solutions still to be discovered. Also, $A.r$ is notified to add $(A.r, \bot)$ to its search risks, and given all search risks $(B.s, \kappa')$ of $A.r$, f is notified to add $(B.s, \kappa' \oplus \kappa)$ to its search risks. A search risk propagater for f and κ is added to $A.r$'s search risk propagaters, and f is added to the queue if it hasn't already been. The auxiliary function `process_creds` defined in Appendix A.4 implements much of this.

3. To process a linked role $A.r_1.r_2$, a linking monitor for $A.r_1.r_2$ is created, is applied to all of $A.r_1$'s known solutions, and is added to $A.r_1$'s solution

monitors. The node $A.r_1$ is notified to add the search risks of $A.r_1.r_2$ to its own search risks, $A.r_1.r_2$ acquires a search risk propagater for $A.r_1$ and $\perp$, and $A.r_1$ is added to the queue if it hasn't already been.

4. To process an intersection role $f_1 \cap \cdots \cap f_n$, an intersection monitor for $f_1 \cap \cdots \cap f_n$ is created, and added to each f_i. For each f_i and every search risk $(A.r, \kappa)$ of $f_1 \cap \cdots \cap f_n$, the node f_i is notified to add the search risk $(A.r, \kappa)$ and the node $f_1 \cap \cdots \cap f_n$ acquires a search risk propagater for f_i and $\perp$, and each f_i is added to the queue if it hasn't already been. The auxiliary function `process_isect` defined in Appendix A.4 implements much of this.

The algorithm `checkmem` defined in Appendix A raises an exception `Solved` if the node $A.r$ is notified to add a solution (B, κ) such that $\kappa \preccurlyeq \Theta(A.r)$. At the top-level, we write $\texttt{checkmem}(A, B.r, \Theta)$ to denote invocation of a function that calls `checkmem` on our OCaml representation of arguments, and that returns true if this call raises a `Solved` exception and false if it terminates unexceptionally. Hence, given a set of distributed credentials $\mathcal{C}$, when an invocation $\texttt{checkmem}(A, B.r, \Theta)$ terminates, the algorithm returns true iff there exists κ such that $(A, \kappa) \in \mathcal{S}_C^\Theta$.

6.2.5 Properties

Assuming that defining credentials can always be obtained for any role, we assert that `checkmem` satisfies the following properties, demonstrating that it correctly reconstructs credential graphs. Since credential graphs are full abstractions of the RT^R semantics as discussed in Sect. 6.1, these results demonstrate that `checkmem` is a correct implementation of RT^R. The proofs, omitted here for brevity, are straightforward extensions of results in [21], since our `checkmem` algorithm is an extension of the `backward` algorithm described in that paper. Note that these results presuppose that credentials are stored in a manner that allows backwards chain reconstruction: we say that a set of credentials $\mathcal{C}$ is *distributed* if they are stored in a manner that allows lookup given the credential issuer.

Theorem 6.5 (Soundness) *Given a set of distributed credentials $\mathcal{C}$, if an invocation $\texttt{checkmem}(A, B.r, \Theta)$ holds then there exists κ such that $(A, \kappa) \in \mathcal{S}_\mathcal{C}^\Theta$.*

Theorem 6.6 (Completeness) *Given a set of distributed credentials $\mathcal{C}$, if there exists κ such that $(A, \kappa) \in \mathcal{S}_\mathcal{C}^\Theta$ then $\texttt{checkmem}(A, B.r, \Theta)$ holds.*

For our algorithm, an issue highly relevant to completeness is how search risks are computed. Recalling that discovery will not proceed along paths where search risks are over-threshold, it is essential to observe that search risks are indeed under-approximations of role membership risk. Othermise, search could be aborted along paths that would otherwise discover solutions within the given threshold constraint, resulting in incompleteness. This property is expressed via the following lemma.

Lemma 6.1 *Given a set of distributed credentials $\mathcal{C}$, at any point during execution of* `checkmem`$(C, B.s, \Theta)$ *for arbitrary C, $B.s$, and Θ, if a node identified by a role expression f contains $(A.r, \kappa)$ in its search risks, then $f \xrightarrow{\kappa'\!\!\twoheadrightarrow} A.r \in \mathcal{G}_{\mathcal{C}}$ such that $\kappa' \preccurlyeq \kappa$.*

We also observe that the algorithm terminates given a finite set of credentials, regardless of the given threshold. This is because nodes are never visited more than once, and solution monitors will not traverse any graph cycle, and hence are guaranteed to terminate. Solution monitors only propagate canonically new members, but traversal of a cycle necessarily causes a monotonic increase in a solution's risk assessment, hence canonical containment in an existing solution. Search risk propagaters are similarly guaranteed to terminate.

Theorem 6.7 (Termination) *Given a finite set of distributed credentials $\mathcal{C}$, for all B, $A.r$, and Θ,* `checkmem`$(B, A.r, \Theta)$ *terminates.*

6.2.6 Discussion: Example

We now provide a graphical example that illustrates how the algorithm works. While the algorithm possesses a number of technical details, its basic operation is fairly straightforward as the graphic shows. Assume given the natural number risk ordering defined in Example 4.1, where aggregation is addition. Modifying the Example in Sect. 2.1, assume that the following credentials are stored with their issuers, numbered here (i) through (v) for later reference:

$$H.discount \xleftarrow{15} H.preferred \quad (i) \qquad H.discount \xleftarrow{5} H.orgs.members \quad (ii)$$

$$H.orgs \xleftarrow{10} AAA \quad (iii) \qquad H.preferred \xleftarrow{7} AAA.members \quad (iv)$$

$$AAA.members \xleftarrow{4} Mary \quad (v)$$

Finally, define Θ as the threshold that maps $H.discount$ to 20 and every other role to ω. In Fig. 2, we show how a credential graph evolves during a run of:

$$\texttt{checkmem}(H.discount, Mary, \Theta)$$

in subfigures (a) through (f). Nodes in the graph are represented by circles, and edges along which solution monitors will propagate solutions by arrows, labeled by the risk aggregated by traversing that edge. Each node is labeled with three possibly blank lines: its role expression identifier on the first line, its search risk from role $H.discount$ on the second, and its solution on the third. Only search risks from $H.discount$ are denoted since all others are unconstrained by Θ and therefore irrelevant to the algorithm. We now describe each subfigure (a) through (f) in order.

(a) Initially, the node $H.discount$ is added to the graph, with a search risk of 0 from itself.

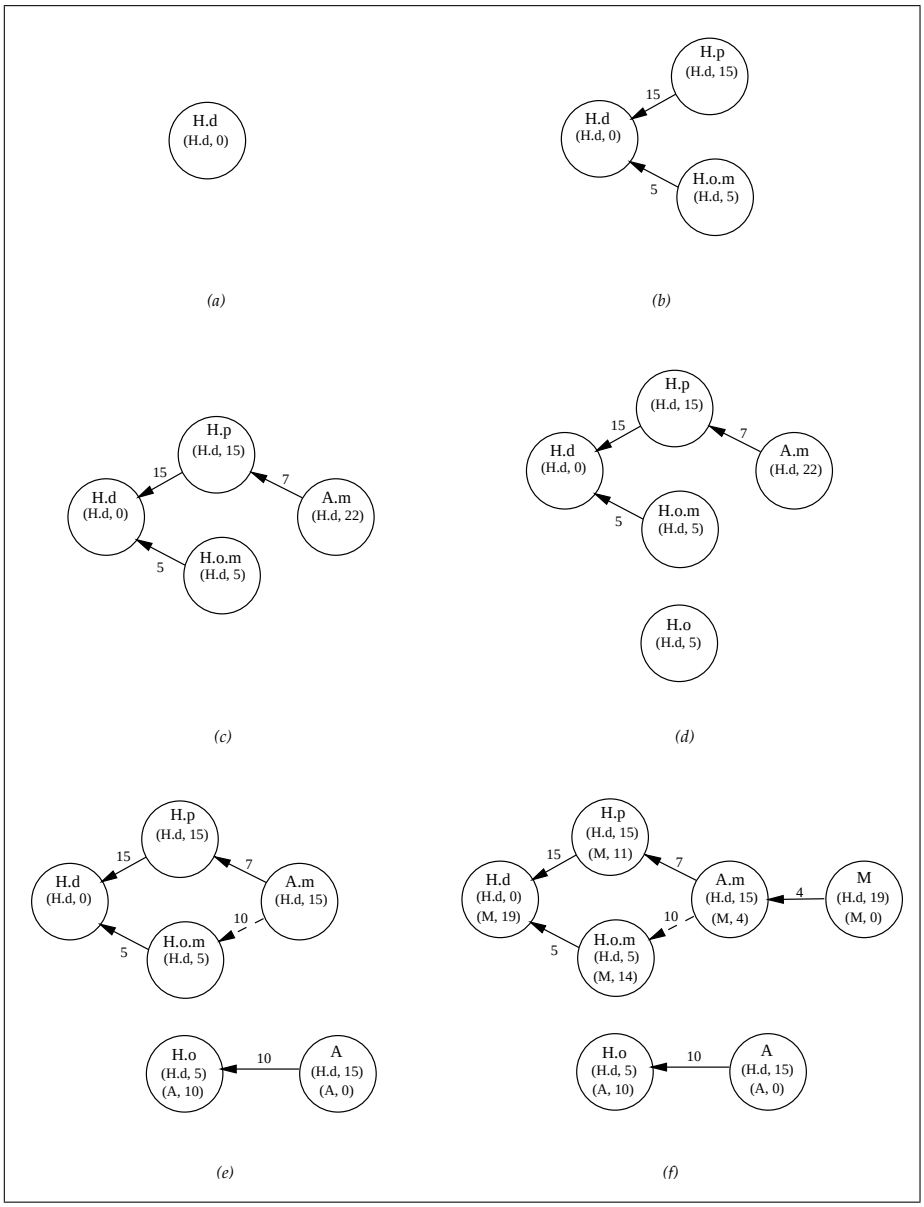


Figure 2: Graphical Example of `checkmem` Execution

(b) After retrieval of credentials (i) and (ii) defining role $H.discount$, the nodes $H.preferred$ and $H.orgs.members$ are added to the graph with the appropriate search risks from $H.discount$.

(c) Assuming that node $H.preferred$ is added before $H.preferred$, the former is the next to be searched, resulting in retrieval of credential (iv) and the addition of node $AAA.members$ to the graph with its search risk from $H.discount$ being the aggregate of preceding risks.

(d) Since the current search risk of $AAA.members$ from $H.discount$ is above the threshold specified by Θ, the search will proceed by exploring the next below-threshold node in the queue, which is $H.orgs.members$. This results in the addition of the node $H.orgs$ to the graph.

(e) Since $H.orgs$ is now the next unsearched below-threshold node in the queue, credential (iii) will be retrieved, so that node AAA is added to the graph. Also, for the first time, solutions are added to the graph: $(AAA, 0)$ is added to AAA's solution, as is $(AAA, 10)$ to $H.orgs$'s, the latter by a solution monitor.

(f) The previous step reduced the search risk of $AAA.members$ from $H.discount$ to 15, so the former node is now below-threshold and searchable. This results in retrieval of credential (v), and the addition of node `Mary` to the graph. Since $Mary$ is an entity, $(Mary, 0)$ is added to $Marys$'s solution, and propagated backwards with aggregated risk by solution monitors. This results in the addition of $(Mary, 19)$ to node $H.discount$, so the algorithm terminates with success.

6.2.7 Discussion: Refinements

There are two particular instances where the definition of `checkmem` could be enhanced, for more eager short-circuiting of chain discovery in case risk thresholds are exceeded along discovery paths. First, observe that credentials are retrieved before being checked to see if their risks will force the discovery threshold to be exceeded. However, risks such as expected wait time suggest that it is more practical for credentials to be retrieved after ensuring they won't overrun the threshold. A number of minor variations on `checkmem` can be imagined that will address this.

A more interesting enhancement is relevant to the propagation of search risks along discovery paths leading from intersection nodes. Observe that from any intersection role $f_1 \cap \cdots \cap f_n$, the search risks of $f_1 \cap \cdots \cap f_n$ are propagated to each f_i. However, this could be a under-approximation of search risks for any given f_i. For example, suppose that A is being checked for authorization and (A, κ) is known to be the only possible assessment of A in f_1's solution. When checking f_n, the search risks of f_n inherited from $f_1 \cap \cdots \cap f_n$ could be aggregated with κ, since κ is certain to be a component risk of any authorization supported by discovery from f_n.

7 Applications

In this section we discuss interesting applications of RT^R. Details of these applications are avenues for future work.

7.1 Authorization Caching and Certificate Validity

Since authorization decisions can be expensive to compute, and it might be the case that a decision needs to be computed again in the future, caching of decision results can be a useful tactic. Indeed, since some decisions involve subdecisions, a number of results can be cached, or vice-versa a number of cached results can be used, for many authorization decisions. Recall that in the example in Sect. 6.2.6, it is determined that *Mary* is a member of several roles, and that *AAA* is a member of $H.orgs$. All of these facts could be cached, and if in the future it is necessary to determine e.g. that *Mary* is a member of $H.preferred$, the fact will be on hand.

Previous systems have leveraged this idea. In particular, the ConChord system for SDSI certificate storage and name resolution [3] maintains a *closure* of the certificate database, where all name bindings derivable from a certificate are computed and cached upon certificate insertion. As a consequence, authorization decisions are $O(1)$. However, the introduction of certificate validity measures, such as expiration dates, pose a problem for this strategy: cached computations may depend on certificates that have expired. How to determine that this is or is not the case, without re-computation of the authorization decision?

The system RT^R provides a solution to this problem. Imagine that authorization facts discovered during `checkmem` computation are cached as weighted paths $B \xrightarrow{\kappa}\!\!\!\!\!\rightarrow A.r$. Let $\mathcal{K}$ be the set of timestamps, and define $\preccurlyeq$ as the "more recent than" relation between timestamps. Thus, credentials can be labeled with their expiration date. By defining $\oplus$ and $\oplus$ as operations that return the older of their operands, any membership decision will be labeled with the expiration date of the oldest certificate involved in the decision. Hence, by caching computed authorization facts as weighted paths of the form $B \xrightarrow{\kappa}\!\!\!\!\!\rightarrow A.r$ in this risk model and by defining a threshold Θ mapping every role to the current date/time, cached facts can be reused in a manner that allow those reliant on expired certificates to be immediately rejected.

7.2 Trust but Verify

The Trust but Verify (TbV) framework [25] provides a setting for distributed trust management that takes into account a notion of *trust* for online authorization decisions, backed up by offline *verification*. Many realistic authorization decisions require "softening" of security in the online phase; this amounts to trusting the validity of certain assertions in this phase, that would otherwise be too expensive to verify. However, online trust should be specified so that

sound offline verification is well-defined, providing formal certainty that offline verification supports online trust.

Any authorization decision in the TbV framework is abstractly specified as derivability of a target privilege **priv** given a security context s, written $s \vdash \mathbf{priv}$. Any instance of the TbV framework comprises a *trust transformation*, that formalizes the definition of trust in terms of a function, mapping initial security contexts s to contexts $[\![s]\!]$, that contain assertions that are trusted solely for efficient online verification. Furthermore, the trust transformation should be reversible, via an audit technique that is required to reconstruct a security context that is at least as strong as the pre-image s of any trust-transformed security context $[\![s]\!]$. The audit technique is the implementation of offline verification. In [25], the TbV framework is developed using ABLP logic [1]. However, the RT framework is a more modern trust management system, with a variety of implementation techniques and variations [21]. The RT^R variation offers a unique dimension of support for TbV, since trust can be encoded using definitions of risk in RT^R.

The TbV framework is characterized by three conditions, that we recount here. We show how RT^R can be used to instantiate the framework in a system that satisfies these conditions. The first condition requires that authorization decisions are decidable:

Condition 7.1 *Let s be an authorization context; then validity of $s \vdash \mathbf{priv}$ is decidable.*

In RT^R, authorization decisions are implemented as role membership decisions with an assessed risk, and security contexts are sets of credentials $\mathcal{C}$. That is, if the role $A.r$ represents a target privilege and B is a privilege requester, then authorization amounts to discovery of whether there exists κ such that $(B, \kappa) \in \mathcal{S}_{\mathcal{C}}^{\Theta}$ for some specified threshold Θ. We have shown that this relation is decidable.

The second condition specifies that auditing reverses trust transformation, though since trust transformations can be many-to-one, the context returned by auditing need not be the exact preimage of trust transformation:

Condition 7.2 *Let s be a trusted context. Then success of audit(s) implies that $[\![\mathit{audit}(s)]\!] = s$.*

The last condition sufficiently strengthens the requirements of auditing to formally establish that any auditing is a sound verification of trust injected by the trust transformation:

Condition 7.3 *Let s be a trusted context; then if audit(s) succeeds, for all* **priv** *it is the case that audit$([\![s]\!]) \vdash \mathbf{priv}$ implies $s \vdash \mathbf{priv}$.*

The condition requires that auditing of a trust-transformed context must reconstruct a context that is at least as strong as the initial context, prior the trust transformation. In RT^R, since authorization contexts are credentials $\mathcal{C}$, and the authorization decision includes a risk threshold, trust transformations

may be implemented via an increase in the tolerable risk thresholds in chain discovery. Assuming the same credentials $\mathcal{C}$ as the example in Sect. 5.1, the initial authorization decision could be to determine whether there exists κ such that $(Ed, \kappa) \in \mathcal{S}_{\mathcal{C}}^{\Theta}(Store.buyer)$, where $\Theta(Store.buyer) = medium$ and $\Theta(A.r) = high$ for all other roles $A.r$. An online trust transformation could be implemented by using the threshold $\Theta[Store.buyer : high]$, allowing Ed's credential $Acme.purchaser \stackrel{high}{\longleftarrow} Ed$ to be used for immediate authorization. Auditing in this case would just consist of authorization under threshold Θ. The following Lemma is trivial and establishes that this trust transformation satisfies the last condition enumerated above.

Lemma 7.1 *Define* $\Theta_1 \preccurlyeq \Theta_2$ *iff* $\Theta_1(A.r) \preccurlyeq \Theta_2(A.r)$ *for all roles* $A.r$. *Then* $\mathcal{S}_{\mathcal{C}}^{\Theta_1} \preccurlyeq \mathcal{S}_{\mathcal{C}}^{\Theta_2}$ *for arbitrary* $\mathcal{C}$.

7.3 Cost/Benefit Analysis

Risk in RT^R is defined in an abstract manner. Although the examples in this paper have used atomic risk values, it is possible to define a risk ordering on compound risk values. For example, suppose both levels of "trustability" and expected wait times for retrieval of specific credentials are considered components of risk. The set $\mathcal{K}$ could then contain elements of the form (κ, t), where $\kappa \in \{low, medium, high\}$ as in Sect. 5.1 and t is a wait time represented as an integer, and:

$$(\kappa_1, t_1) \preccurlyeq (\kappa_2, t_2) \iff \kappa_1 \preccurlyeq \kappa_2 \wedge t_1 \leq t_2$$

reflecting that lower wait times, as well as higher confidence in validity, define lower risk. Maximum risk in chain discovery would then specify both a tolerable level of trust, and a tolerable wait time for any particular credential.

This suggests an *interactive* procedure for chain discovery, where the costs of raising the level of one component of risk could be balanced against benefits in another risk dimension. In the above scenario, if chain discovery in some instance fails given a threshold Θ such that $\Theta(A.r) = (\kappa, t)$ for the governing role $A.r$, chain discovery could be re-run with a higher threshold, but notice there is a choice of which element(s) of risk to raise. The cost of raising κ can then be balanced against the benefits in the time dimension, by re-running chain discovery with the threshold $\Theta[A.r : (\kappa', t)]$ with $\kappa \preccurlyeq \kappa'$. The opposite is also clearly the case. This cost/benefit analysis would be further enhanced by optimizing chain discovery. The backward chain discovery algorithm presented in this paper ensures that risks are kept below a certain threshold, but does not attempt to optimize risk. By extending chain discovery with optimization techniques, in the presence of compound risk, benefit dimensions could be optimized within a fixed cost dimension. For example, optimal wait times could be sought given a *high* level of trust risk. Development of optimizing algorithms is a topic for future work.

8 Conclusion

We now conclude with comments on related work and a short summary of the paper.

8.1 Related Work

Many trust management systems have been developed by previous authors. In any such system resource owners write policy statements using a suitable policy language that describes the attributes of authorized users. When a request is made, the requesting entity provides signed credentials that prove the requester complies with the policy. Proofs are constructed automatically, and implement a formal semantics. Previous systems include BAN [10] and ABLP logic [1], PolicyMaker [9], KeyNote [8], SDSI/SPKI [23], [13], and RT [20], [21], [19], to name a few. However, our focus is not on trust management in general, but trust management extended with risk assessment.

Proof carrying carrying authorization (PCA) [6, 5] is a framework for specifying and enforcing webpage access policies. It is based on ABLP logic, but includes primitives for detecting timestamp expiration. While this capability reflects some sense of risk assessment, it is not as general as the notion of risk expressed in our system.

In [4], semantics for a number of RT variants are obtained via embedding in constraint datalog. An implementation of "confidence levels", similar to our notion of risk assessment, is suggested via the use of constraints, though not developed in detail. While it is possible that many interesting risk assessment schemes can be defined using RT_1 or RT_2, we believe that defining a new RT variant to explicitly capture the notion of risk assessments is appealing in various respects. In particular, we are able to define risk in a general manner, and isolate issues related to online authorization with components of risk.

In prior work, we developed preliminary foundations for the system RT^R [11]. However, the current presentation has a more rigorously developed metatheory and chain discovery algorithm, and a more general theory of authorization thresholds. New applications are also presented here, in particular delegation width and depth control, and certificate validity.

Dealing with trustworthiness in distributed systems has been an active research area (see, e.g., [14]). In [18], an algebra is provided for reasoning about trust in certificate chains. Our notion of risk is related to the notion of trust, and some relevant operators of [18] may be directly incorporated into our framework. Comparative expressiveness of risk and trust operators is an interesting research topic, but is beyond the scope of this paper.

A framework for characterizing generalized authorization problems has been developed for SPKI/SDSI [24] that is closely related to our risk assessment scheme for RT. In their system, *weights* chosen from a collection of values with certain algebraic properties label certificates, and certificate chains are endowed with a weight that is a combination of the component certificate weights. However, their weighting framework is more general than ours, and is fundamentally

based on *witness sets*– sets of certificate chains that establish a particular authorization, rather than individual role membership decisions as in our scheme. Also, their focus is not on algorithms that use proof direction based on risk considerations for efficiency gains, as in our work.

8.2 Summary

In this paper we have defined RT^R, a role-based trust management framework with formal risk assessment. This system is a variation on RT [19], and includes the capability to associate credentials with risk, and to assess risk levels of authorization as the aggregated risks of authorization components. Risks are defined in an abstract manner, under the requirement that the set of risks be a complete lattice, with a monotonic aggregation operator. A formal semantics has been given, that associates role membership with risk levels. An algorithm has also been defined for implementation of this semantics, providing an automatic risk assessed authorization procedure. The algorithm is specialized for functionality in a distributed environment, and can be parameterized by risk thresholds, specifying a maximum tolerable risk for authorization. The algorithm is directed, to avoid proof paths whose aggregate risks exceed the given threshold, hence to risk as little as possible during the course of authorization. To illustrate the usefulness of the system, we have discussed a number of applications, including delegation depth and width control, a trust-but-verify approach to authorization, and authorization decision caching in the presence of certificate expiration.

Acknowledgements

The work of Wang and Chapin was partly supported by the NSF grant IIS-0430165.

The work of Skalka was partly supported by the AFOSR grant USAF-9550-06-1-0313.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[2] Abdul-Rahman. The PGP trust model. EDI-Forum: the Journal of Electronic Commerce, 1997.

[3] Sameer Ajmani, Dwaine E. Clarke, Chuang-Hue Moh, and Steven Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *International Workshop on Peer-to-Peer Systems*, January 2002.

[4] Scot Anderson. Constraint datalog in trust management. Master's thesis, University of Nebraska, 2003.

[5] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In G. Tsudik, editor, *Proceedings of the 6th Conference on Computer and Communications Security*, Singapore, November 1999. ACM Press.

[6] Lujo Bauer. *Access Control for the Web via Proof-carrying Authorization.* PhD thesis, Princeton University, 2003.

[7] Andrew Birrell, Butler W. Lampson, Roger M. Needham, and Michael D. Schroeder. A global authentication service without global trust. In *IEEE Symposium on Security and Privacy*, pages 223–230, 1986.

[8] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. *RFC-2704: The KeyNote Trust-Management System Version 2.* IETF, September 1999.

[9] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. Technical Report 96-17, DIMACS, June 28 1996.

[10] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[11] Peter Chapin, Christian Skalka, and X. Sean Wang. Risk assessment in distributed authorization. In *Proceedings of the ACM Workshop on Formal Methods in Security Engineering*, 2005.

[12] D. Denning. A lattice model of secure information flow. In *Communications of the ACM*, pages 236–243. ACM, May 1976.

[13] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, Sept. 1999.

[14] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials*, 4th Quarter, 2000.

[15] Carl A. Gunter and Trevor Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, 2000.

[16] Fan Hong, Xian Zhu, and Shaobin Wang. Delegation depth control in trust-management system. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 411–414, Washington, DC, USA, 2005. IEEE Computer Society.

[17] International Telecommunications Union. *Information Technology - Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks*, 2000.

[18] Audun Jøsang. An algebra for assessing trust in certification chains. In J. Kochmar, editor, *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99)*. The Internet Society, 1999.

[19] Ninghui Li and John C. Mitchell. Rt: A role-based trust-management framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition*, pages 201–212. IEEE Computer Society Press, April 2003.

[20] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[21] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[22] Michael K. Reiter and Stuart G. Stubblebine. Authentication metric analysis and design. *ACM Transactions on Information Systems Security*, 2(2):138–158, 1999.

[23] R. Rivest and B. Lampson. SDSI — a simple distributed security infrastructure, 1996. `http://theory.lcs.mit.edu/rivest/sdsi11.html`.

[24] Stefan Schwoon, Somesh Jha, Thomas W. Reps, and Stuart G. Stubblebine. On generalized authorization problems. In *CSFW*, pages 202–. IEEE Computer Society, 2003.

[25] Christian Skalka and X. Sean Wang. Trust but verify: Authorization for web services. In *ACM Workshop on Secure Web Services*, October 2004.

A Definition of `checkmem`

In this appendix we describe our OCaml implementation of the `checkmem` algorithm. As described in the text, the implementation is in a procedural style, but exploits higher order functions and types available in OCaml.

A.1 Type Definitions

To abstract the definition of risk orderings in the implementation, we parameterize the implementation by a module `Risk`, that satisfies the following type signature `RISK`.

```
module type RISK =
  sig
    type risk
    val lt : risk -> risk -> bool
    val plus : risk -> risk -> risk
    val times : risk -> risk -> risk
    val bot : risk
    val top : risk
  end
```

Any concrete instance `Risk` of `RISK` implements the risk ordering provided in an instance of RT^R, insofar as each component of the module implements a component of a given risk ordering, as follows.

`Risk.risk`: the set of risk values $\mathcal{K}$.

`Risk.lt`: the risk ordering $\preccurlyeq$.

`Risk.plus`: risk aggregation $\oplus$.

`Risk.times`: intersection aggregation $\otimes$.

`Risk.bot`: the bottom element $\bot$.

`Risk.top`: the top element $\top$.

Roles, role expressions, credentials, risk assessments, and thresholds are then specified as follows. Let types `entity` and `role_name` be arbitrary identifiers with equality operations, e.g. `string`s. We define:

```
type role = entity * role_name
type role_expr =
    Ent of entity
  | Role of role
  | Link of entity * role_name * role_name
  | Isect of role_expr list
type cred = Cred of role * Risk.risk * role_expr
type risk_assessment = (entity * Risk.risk) list
type threshold = role_expr -> Risk.risk
```

Now we specify types that are specific to the `checkmem` algorithm: search risks, solution monitors, and search risk propagaters.

```
type search_risk = role * risk
type monitor = (entity * risk) -> unit
type propagater = (role * risk) -> unit
```

Nodes are specified as records identified by constant role expressions, with mutable fields for maintaining data structures relevant to the algorithm, and a visited flag.

```
type node = {
    id : role_expr;
    mutable monitors : monitor list;
    mutable propagaters : propagater list;
    mutable search_risks : search_risk list;
    mutable solution : risk_assessment;
    mutable visited : bool
  }
```

```
let risk_notify nd (r,k) =
  if not (subsumed (r,k) nd.search_risks)
  then
    begin
      add_search_risks [(r,k)] nd;
      apply_all nd.propagaters (r,k)
    end

let make_propagater nd k' = (fun (r,k) -> risk_notify nd (r, Risk.plus k k'))
```

Figure 3: Search Risk Propagation Functions

A.2 Auxiliary Functions and Data Structures

The `checkmem` algorithm uses two global mutable data structures, identified by global variables in our implementation:

`q`: global queue data structure for storing nodes to be searched.

`nodes`: global list of all nodes created during the course of authorization.

The `checkmem` algorithm uses a library of auxiliary functions whose semantics we describe below. The definitions are not particularly relevant and are omitted for brevity, but we describe their argument lists and semantics as follows. Recall that in OCaml, argument lists can be written in a "curried" style, e.g. `f x y` instead of `f(x,y)`.

`make_target a r`: sets membership checking of entity `a` in role `r` as main target of the authorization decision.

`issuer_retrieve r`: retrieves all certificates in $\mathcal{C}$ for which `r` is the issuer.

`add_monitor m nd`: adds a solution monitor `m` to `nd.monitors`.

`add_propagater p nd`: adds a search risk propagater `p` to `nd.propagaters`.

`add_search_risks risks nd`: adds `risks` to `nd.search_risks`.

`add_new_solution (b,k) nd`: given solution element `(b,k)` that is not subsumed by `nd.solutions`, returns canonical union of $\{$`(b,k)`$\}$ and `nd.solutions`.

`apply_all fs x`: apply all functions in the list `fs` to `x`.

`apply_to_all f l`: apply `f` to all elements of list `l`.

`aggregate_all rs k`: transforms search risks or assessment `rs` so that every element `(x,k')` becomes `(x,Risk.plus k' k)`

`subsumed (x,k) risks`: a predicate that holds iff there exists `(x,k')` in `risks` such that `Risk.lt k' k`.

`produce_node f`: if `nodes` contains a node `nd` identified by `f`, returns `nd`, otherwise returns an initialized node identified by `f`, and adds it to `nodes`.

`enqueue_unvisited nd`: adds `nd` to `q` and sets the flag `nd.visited` to true if `nd.visited` is false, otherwise a noop.

```
let soln_notify nd (b,k) thresh =
  if (Risk.lt k (thresh nd.id)) && not (subsumed (b,k) nd.solution)
  then
    begin
      add_new_solution (b,k) nd;
      apply_all nd.monitors (b,k)
    end

let make_nmonitor nd k thresh =
  (fun (b,k') -> soln_notify nd (b, Risk.plus k k') thresh)

let make_lmonitor nd thresh =
  match nd.id with
    Link(a,r1,r2) ->
      (fun (b,k) ->
        let m = make_nmonitor nd k thresh in
        let nd' = produce_node (Role(b,r2)) in
          begin
            apply_to_all m nd'.solution;
            add_monitor m nd';
            add_search_risks (aggregate_all nd.search_risks k) nd';
            add_propagater (make_propagater nd' k) nd;
            enqueue_unvisited nd';
          end
      )
  | _ -> raise BadNode
```

Figure 4: Solution Monitor Functions

`below_thresh thresh risks`: a predicate that holds iff search risks `risks` are below threshold `thresh` (Definition 6.5).

`searchable thresh q`: a predicate that holds iff queue `q` contains at least one node whose search risks are below threshold `thresh`.

`getnext thresh q`: gets the next searchable node from queue `q`.

A.3 Propagaters and Role Monitors

To propagate search risks along search paths, we define functions for notifying nodes of new risks, and for making propagaters for a given node and risk. These functions are defined in Fig. 3.

`risk_notify nd (r,k)`: notifies the node `nd` to add a new search risk `(r,k)`, and applies all of `nd.propagaters` to `(r,k)`.

`make_propagater nd k'`: returns search risk propagater for node `nd` and risk `k'`.

To propagate solutions along solution paths, we define functions to notify nodes of solutions, and functions to generate solution monitors. Note that node, linking, and intersection monitors are not defined explicitly, but are returned

```
let process_cred nd thresh (Cred(_,k,f)) =
  let m = make_nmonitor nd k thresh in
  let nd' = produce_node f in
  begin
    apply_to_all m nd'.solution;
    add_monitor m nd';
    apply_to_all (risk_notify nd') (aggregate_all nd.search_risks k);
    add_propagater (make_propagater nd' k) nd;
    enqueue_unvisited nd'
  end

let process_isect nd m nd' =
  begin
    add_monitor m nd';
    add_search_risks nd.search_risks nd';
    add_propagater (make_propagater nd' Risk.bot) nd;
    enqueue_unvisited nd'
  end

let rec checkmem b r thresh =
  begin
    Queue.clear q;
    nodes := [];
    make_target b r;
    enqueue_unvisited (produce_node (Role r));
    while (searchable thresh q) do
      let nd = getnext thresh q in
      match nd.id with
        Ent(a) -> soln_notify nd (a, Risk.bot) thresh
      | Role(r) ->
          begin
            add_search_risks [(r,Risk.bot)] nd;
            apply_to_all (process_cred nd thresh) (issuer_retrieve r)
          end
      | Link(a,r1,r2) ->
          let nd' = produce_node (Role(a,r1)) in
          let m = make_lmonitor nd thresh in
          begin
            apply_to_all m nd'.solution;
            add_monitor m nd';
            add_search_risks nd.search_risks nd';
            add_propagater (make_propagater nd' Risk.bot) nd;
            enqueue_unvisited nd'
          end
      | Isect(fs) ->
          let m = make_imonitor nd thresh in
          let nds = List.map produce_node fs in
          apply_to_all (process_isect nd m) nds
    done
  end
```

Figure 5: Node Processing (`checkmem`) Function Definitions

by higher order functions for making them. All of these functions are defined in Fig. 4 except for `make_imonitor` which is omitted for brevity.

`soln_notify nd (b,k) thresh`: notifies node `nd` to add a new solution element `(b,k)`, and applies all of `nd.monitors` to `(b,k)`. Raises a `Solved` exception if membership of `b` in `nd.id` is the target of authorization as set by `make_target`.

`make_nmonitor nd k thresh`: returns a node monitor for node `nd` and risk `k`.

`make_lmonitor nd thresh`: returns a linking monitor for node `nd`.

`make_imonitor nd thresh`: returns an intersection monitor for node `nd`.

A.4 Node Processing

Putting all the pieces together, we can now define the `checkmem` function that implements node processing. To make the code more readable, we have factored out some blocks as auxiliary functions. These functions are defined in Fig. 5. The invocation of `make_target` sets the goal of the authorization decision; if it is discovered before the queue is empty, an exception is raised and computation is aborted.

`process_cred nd thresh c`: does all the work for processing a credential `c` that defines a role identifying `nd` in the role case of `checkmem`.

`process_isect nd m nd'`: given `nd` identified by an intersection role, does most of the work of processing a node `nd'` identified by a component of the intersection.

`checkmem b r thresh`: implementation of the `checkmem` algorithm.