Towards Practical Automated Trust Negotiation

William H. Winsborough
NAI Labs
Network Associates, Inc.
3060 Washington Road
Glenwood, MD 21738
william_winsborough@nai.com

Ninghui Li
Department of Computer Science
Stanford University
Gates 4B
Stanford, CA 94305-9045
ninghui.li@cs.stanford.edu

Abstract

Exchange of attribute credentials is a means to establish mutual trust between strangers that wish to share resources or conduct business transactions. Automated Trust Negotiation (ATN) is an approach to regulate the exchange of sensitive credentials by using access control policies. Existing ATN work makes unrealistic simplifying assumptions about credential-representation languages and credential storage. Moreover, while existing work protects the transmission of credentials, it fails to hide the contents of credentials, thus providing uncontrolled access to potentially sensitive attributes. To protect information about sensitive attributes, we introduce the notion of attribute acknowledgment policies *(Ack policies). We then introduce the trust target graph (TTG) protocol, which supports a more realistic credential language, Ack policies, and distributed storage of credentials.*

1 Introduction

Access control presents difficult problems in a decentralized collaborative environment, particularly when resources and the subjects requesting them belong to different security domains controlled by different authorities. Many commonly used access control mechanisms make authorization decisions based on the identity of the resource requester. Unfortunately, when the resource owner and the requester are unknown to one another, access control based on principal identity may be ineffective. Some trust management systems, such as KeyNote [1] and SPKI[1], address this by using credentials that delegate permissions. In these systems, a credential, or a credential chain, acts as a capability giving the subject certain permissions. However, even capability-style systems do not match well the nature of decentralized authority in collaborative environments.

Suppose Alpha and Beta enter into a coalition, and Alpha wants to give access to Beta's engineers. In the traditional identity-based approach, administrators in Alpha would explicitly authorize individual engineers in Beta. The need for Alpha to be aware of users from collaborating organizations that might access its resource is a significant administrative burden. Using a capability-like system, an administrator in Alpha typically delegates permissions to an administrator in Beta, who then further delegates those permissions to eligible users in Beta. These permissions are resource specific, and each one requires a separate delegation act by Beta's administrator, which remains a significant burden.

To better serve the needs of collaborative environments, particularly when alliances shift frequently, we aim to base access control decisions on authenticated attributes of entities (*i.e.*, organizations, users, or processes in the system), while simultaneously decentralizing attribute authority. We call this approach attribute-based access control (ABAC). Some trust management systems, such as RT [6, 7], explicitly support ABAC, and others, such as SPKI/SDSI 2.0, can be used as ABAC systems. Authorization decisions are based on attributes of the requester, which are established by digitally signed credentials through which credential issuers assert their judgments about the attributes of entities. Because these credentials are digitally signed, they can serve to introduce strangers to one another without on-line contact with attribute authorities. ABAC avoids the need for permissions to be assigned to individual requesters before the request is made. Instead, when a stranger requests access, the ABAC-enabled access mediator can make an authorization decision by combining agreements and judgments of decentralized authorities in a natural and logical way.

ABAC systems depend on credentials that specify attributes of entities and/or rules for deriving entities' attributes. These attributes (such as financial or medical data) may be sensitive. Because we are interested in establishing

[1]We use SPKI to denote the part of SPKI/SDSI 2.0 [3, 4] originally from SPKI, *i.e.*, 5-tuples, and SDSI to denote the part of SPKI/SDSI 2.0 originally from SDSI, *i.e.*, name certificates (or 4-tuples as called in [4]).

trust between entities that initially have no mutual trust, it is unlikely the requester and the access mediator will be able to agree upon a trusted third-party that might assist them in using their sensitive credentials to establish mutual trust.

The goal of a growing body of work on *automated trust negotiation (ATN)* [8, 9, 10, 11] is to enable resource requesters and access mediators to establish trust in one another through cautious, iterative, bilateral disclosure of credentials. In the existing ATN literature, access control policies are established to regulate the disclosure of credentials, in addition to the granting of system resources. The negotiation consists of a sequence of exchanges that begin by disclosing credentials that are not sensitive. As credentials flow, higher levels of mutual trust are established, and access control policies for more sensitive credentials are satisfied, enabling these credentials also to flow. In successful negotiations, credentials eventually flow that satisfy the policy of the desired resource.

The current paper addresses three shortcomings of existing ATN system designs. First, existing ATN work makes unrealistic simplifying assumptions about the language for representing credentials. Using over-simplified languages in ATN system design hides issues that need to be addressed. For instance, in [8, 10], propositional languages are used to represent credentials. In practice, the simplest form of attribute credential needs to designate an issuer, a subject, and an attribute name or id. Additional structures are needed to adequately support delegation of authority, which is essential in a decentralized environment. Even the most expressive credential/policy language [5] to be used in prior ATN work [9] does not support delegation of authority.

Second, prior ATN systems make unrealistic assumptions about credential storage. They presume that ATN participants, called negotiators, have at the outset access to all credentials they will need during negotiation. However, in a decentralized collaborative environment, credential storage is not centralized. Thus, ATN system design must consider credential discovery.

The third pitfall of existing ATN designs is that although they protect the transmission of attribute credentials, the fact that the entity does or does not possess a given credential is typically not protected. In all but the most rudimentary of existing ATN systems, a negotiator receiving an attribute query acknowledges, implicitly or explicitly, whether or not she has credentials that satisfy the query. The attribute query can be arbitrarily specific. This enables an unauthorized adversary to determine precisely which credentials—and therefore which (potentially sensitive) attributes—the negotiator does and does not have.

In this paper, we address these three issues through two innovations. First, to control transmissions that could disclose whether or not the negotiator has a given attribute, we introduce the notion of *attribute acknowledgment policies* (Ack policies). Second, we introduce the trust-target graph protocol, which supports a powerful, yet efficient ABAC credential language, Ack policies, and distributed credential storage. As part of on-going work, we are implementing an ATN system based on this protocol.

The rest of this paper is organized as follows. Section 2 discusses the requirements and desired features of credential languages in ABAC and present a language, RT_0, that meets our requirements and has most of our desired features. Section 3 introduces the notion of Ack policies, and illustrate through an example how Ack policies are used to control the flow of information about attributes. Section 4 presents the TTG protocol. Section 5 presents negotiation procedures, based on the TTG protocol and the acknowledgment policy architecture, that support the use of RT_0 and distributed credential storage. Section 6 discusses future work and section 7 concludes.

2 Language for Credentials in ABAC

We argued above that propositions are insufficient for representing credentials in ABAC. We now discuss requirements for credentials in ABAC systems. We illustrate these requirements by using an example scenario in which a bank wants to know whether an entity is a full-time student, to determine whether the entity is eligible to defer repayment on a guaranteed student loan (GSL). (The US government insures banks against default of GSLs and requires participating banks to allow full-time students to defer repayments.) We consider the following expressive features essential:

- Decentralized attributes: an entity asserts that another entity has a certain attribute. *E.g.*, a university registrar may assert that Dan is a full-time student.
- Delegation of attribute authority: an entity delegates authority over an attribute to another entity, *i.e.*, one entity trusts another entity's judgment on the attribute. *E.g.*, a university may delegate to its registrar the determination of who is a full-time student.
- Inference of attributes: an entity uses one attribute to make inferences about another attribute. *E.g.*, a university may consider a student to be full-time even without a full credit load if the student is a Ph.D. candidate.
- Attribute-based delegation of attribute authority. A key to ABAC's scalability is the ability to delegate to strangers whose trustworthiness is determined based on their own attributes. *E.g.*, a university may delegate to the graduate officers of all departments the authority to determine which students are Ph.D. candidates.

In addition, the credential language should have a clear, monotonic semantics. There must be no ambiguity as to whether a given set of credentials shows that a principal has a given attribute. Moreover, since negotiators control the availability of their own credentials, it must not be possible to obtain authorization by virtue of withholding credentials.

Other highly desirable expressive features include:

- Attribute intersection: an entity uses a combination of attributes to infer an additional attribute of another entity. *E.g.*, a university may require that Ph.D. candidates be registered for at least one credit to be considered full-time students.
- Attribute fields. Parameterizing attributes enables values, such as age and credit limit, to be documented and used in credentials.

Several well known trust management systems have some, but not all of these features. For instance, with their focus on delegation of rights, KeyNote and SPKI do not support inference of attributes or attribute-based delegation. While SDSI adds some of this to SPKI/SDSI, the combination is complex, and does not support attribute fields or intersection, reflecting SDSI's focus on naming individuals. Existing public key infrastructure (PKI) also does not meet our requirements. For instance, one cannot express inference of attributes or attribute-based delegation by using X.509 attribute certificates. TPL [5] builds a limited form of attribute inference on top of X.509 attribute certificates. However, delegations cannot be carried in credentials. Thus, in the student loan example, TPL would force every university to issue IDs directly to its full-time students, and would not enable authority over membership in this category to be delegated to the university's registrar or graduate officers. Bonatti and Samarati [2] give an attribute-based authorization language that, like TPL, does not fully support delegation of attribute authority.

In our current work on ATN, we use RT [6, 7], a family of role-based trust management languages. One language in the family, called RT_0, is presented in [7], along with an efficient execution model that is well-suited to interleaving evaluation steps with credential discovery and collection steps. Our presentation in this paper is based on RT_0, which has all the of the features listed above except attribute fields. The fact that RT_0 does not have fields has the advantage of simplifying the presentation here. Additionally, other members of the RT family, such as RT_1, do support fields [6]. We believe that the approach to ATN that we present here can be extended to RT_1, and we plan to do so. Another benefit of using RT_0 is that we can use existing work on credential typing systems [7] in our treatment of negotiating with credentials that are issued and stored in a distributed manner.

2.1 RT_0

In this section, we summarize RT_0. Our presentation of RT_0 follows that of [6, 7].

Constructs of RT_0 include entities, role names (attribute names), and roles (attributes). An *entity* is a uniquely identified individual or process. Entities are also called principals in the literature. Entities can issue credentials and make requests. RT_0 assumes one can determine which entity issued a particular credential or request, for instance, by using public/private key pairs as entities. A *role name* is an identifier, such as a string. A *role* takes the form of an entity followed by a role name, separated by a dot, *e.g.*, $A.r$ and $B.r_1$. A role has a value that is a set of entities who are members of this role. Each entity A has the authority to define who are the members of each role of the form $A.r$. In RT_0, an access control permission is represented as a role as well.

Roles in RT correspond to the notion of attributes in ABAC. An entity is a member of a role if and only if it has the attribute identified by the role. In the rest of this paper, we use the terms role and attribute interchangeably.

There are four kinds of credentials in RT_0, each corresponding to a different way of defining role membership (*i.e.*, of deriving attributes):

- *Type-1*: $A.r \longleftarrow D$

  A and D are (possibly the same) entities, and r is a role name (attribute name).

  This means D has (or *satisfies*) attribute $A.r$, or equivalently, A asserts that D has the attribute r.

- *Type-2*: $A.r \longleftarrow B.r_2$

  A and B are (possibly the same) entities, and r and r_2 are (possibly the same) role names.

  This means an entity has attribute $A.r$ if it has $B.r_2$. That is, A asserts that an entity has the attribute r if B asserts that the entity has the attribute r_2. In particular, if r and r_2 are the same, this is a delegation from A to B of authority over r.

- *Type-3*: $A.r \longleftarrow A.r_1.r_2$

  A is an entity, and r, r_1, and r_2 are role names. We call $A.r_1.r_2$ a *linked role*.

  This means an entity D has $A.r$ if there is an entity B that has $A.r_1$, and D has $B.r_2$. That is A asserts that D has the attribute r if there is a B that A asserts has attribute r_1, and B asserts that D has the attribute r_2. If r and r_2 are the same, A is delegating its authority over r to anyone that A believes to have the attribute r_1. This is attribute-based delegation: A identifies B as an authority on r not by using (or knowing) B's identity, but by another attribute of B (*viz.*, r_1).

- *Type-4*: $A.r \longleftarrow A_1.r_1 \cap A_2.r_2 \cap \cdots \cap A_k.r_k$

  $A, A_1, \ldots, A_k$ are entities, $r, r_1, \ldots, r_k$ are role names, and k is an integer greater than 1. We call $A_1.r_1 \cap A_2.r_2 \cap \cdots \cap A_k.r_k$ an *intersection*.[2]

[2]In [7], an intersection can also contain entities or linked roles. Here, we follow the definition in [6]. The restriction here does not change expressive power: one can always add additional intermediate roles.

This means A asserts an entity has the attribute r if for each $i = 1..k$, A_i asserts the entity has attribute r_i.

A *role expression* (denoted by $e, e_1, e_2, \ldots$) is an entity, a role, a linked role, or an intersection. All credentials in RT_0 take the form, $A.r \leftarrow e$, where e is a role expression. We say that this credential *defines* the role $A.r$.

2.2 Examples and Usage

This section presents two examples in which facilities to perform business transactions are protected resources.

Example 1 Referring to the student-loan deferment scenario at the top of the section, StateU may define its full-time student attribute by the following two credentials, which represent alternative ways of satisfying the attribute:

```
StateU.fulltimeStu ⟵ Registrar.fulltimeStu
StateU.fulltimeStu ⟵
        StateU.phdCand ∩ Registrar.parttimeStu
```

The following credentials, together with the above, show Bob is a full-time student:

```
StateU.phdCand ⟵ StateU.gradOfficer.phdCand
StateU.gradOfficer ⟵ Carol
Carol.phdCand ⟵ Bob
Registrar.parttimeStu ⟵ Bob
```

Now assume StateU is certified by a (fictitious) accreditation board for universities (ABU):

```
ABU.accredited ⟵ StateU
```

If universities define `fulltimeStu` appropriately (for example, as done by StateU above), BankWon can issue credentials like those below to grant loan-deferment permission (denoted by `BankWon.deferGSL`) to students like Bob.

```
BankWon.deferGSL ⟵ BankWon.univ.fulltimeStu
BankWon.univ ⟵ ABU.accredited
```

The intended meaning of an attribute, such as `fulltimeStu`, must be agreed upon, particularly when used in linked roles, like `BankWon.univ.fulltimeStu`, where all universities are assumed to be using a common vocabulary. This is supported in RT through the use of an *application domain specification document* (ADSD), which defines the intended meaning of attribute names in an application-domain specific name space. Each credential refers to such an ADSD. See [6] for details.

Example 2 In the aftermath of a large natural disaster, MedSup, a medical supply merchant, offers to sell at a discount medical supplies to be used in the official clean up, which is being organized by a coalition called ReliefNet. Alice works for MedixFund, one of several charitable organizations that use private contributions to obtain emergency medical supplies for emergency teams working the disaster site. The following four credentials show that Alice is authorized for the discount:

```
MedixFund.pA ⟵ Alice                          (1)
ReliefNet.coaMember ⟵ MedixFund               (2)
MedSup.partner ⟵ ReliefNet.coaMember          (3)
MedSup.discount ⟵ MedSup.partner.pA           (4)
```

Prior to joining the coalition, MedixFund issued credential (1), which states that Alice is a purchasing agent for the fund. One of ReliefNet's responsibilities is to identify coalition-member organizations, as it does in credential (2). MedSup recognizes these organizations as its coalition partners, as in credential (3), and offers discounted sales to the purchasing agents of those partners, as stated in credential (4). In this example, the judgments of MedixFund, ReliefNet, and MedSup are combined to authorize Alice's receiving a discount from MedSup. When MedSup enters into other coalitions, it can add an additional credential defining `MedSup.partner` to give the discount to the purchasing agents of its new partners.

2.3 Semantics of RT_0

In [7], the semantics of RT_0 is defined as a mapping from roles to the entities they contain. To suit our needs here, we extend that semantics to a binary relation consisting pairs of role expressions that are related by role containment.

Given a set $\mathcal{C}$ of RT_0 credentials, let $\mathcal{E}(\mathcal{C})$ be the set of role expressions containing the following: every entity in $\mathcal{C}$, every linked role in $\mathcal{C}$, every intersection in $\mathcal{C}$, and every role $A.r$, in which A is an entity in $\mathcal{C}$ and r is a role name in $\mathcal{C}$.

The semantics of $\mathcal{C}$ is given by the least $\twoheadleftarrow_{\mathcal{C}} \subseteq \mathcal{E}(\mathcal{C}) \times \mathcal{E}(\mathcal{C})$ that satisfies the five conditions below. (We write just $\twoheadleftarrow$, omitting the $\mathcal{C}$ when it is clear from context.)

1. For any $e \in \mathcal{E}(\mathcal{C})$, $e \twoheadleftarrow e$, *i.e.*, $\twoheadleftarrow$ is reflexive.
2. If $e_1 \twoheadleftarrow e_2$ and $e_2 \twoheadleftarrow e_3$, then $e_1 \twoheadleftarrow e_3$, *i.e.*, $\twoheadleftarrow$ is transitive.
3. For any credential $A.r \leftarrow e$ in $\mathcal{C}$, $A.r \twoheadleftarrow e$.
4. If $A.r_1.r_2 \in \mathcal{E}(\mathcal{C})$ and $A.r_1 \twoheadleftarrow B$, then $A.r_1.r_2 \twoheadleftarrow B.r_2$.
5. If $A_1.r_1 \cap \cdots \cap A_k.r_k \in \mathcal{E}(\mathcal{C})$ and $A_i.r_i \twoheadleftarrow e$ for each i in $[1..k]$, then $A_1.r_1 \cap \cdots \cap A_k.r_k \twoheadleftarrow e$.

It is straightforward to see that the relation $\twoheadleftarrow_{\mathcal{C}}$ exists and is well-defined. It can be constructed by starting from the empty relation and iteratively adding new pairs according to the requirements until they are all satisfied.

We claim that the relation $\twoheadleftarrow$ is an extension to the semantics given in [7] in that if $e_1 \twoheadleftarrow_{\mathcal{C}} e_2$, then $member_{\mathcal{C}'}(e_1) \supseteq member_{\mathcal{C}'}(e_2)$ is true for any $\mathcal{C}' \supseteq \mathcal{C}$. Here, $member_{\mathcal{C}}(e)$ is the set of entities that, according to the semantics in [7], are members of e under a given set of credentials $\mathcal{C}$. In particular, $e \twoheadleftarrow_{\mathcal{C}} D$ if and only if $member_{\mathcal{C}}(e) \ni D$.

Example 3 Referring to example 2, credentials (1) and (2) give rise to `MedSup.partner` $\twoheadleftarrow$ `MedixFund` by applying rules 3 and 2, and to `MedSup.partner.pA` $\twoheadleftarrow$ `MedixFund.pA` by additionally applying rule 4.

3 ATN and Sensitive Attributes

As discussed in the introduction, information in attribute credentials is often sensitive, and work on automated trust negotiation (ATN) aims at protecting the transmission of these credentials and their sensitive content. In the ATN literature [8, 9, 10, 11], access control (AC) policies are associated with credentials as well as with resources. When Alice requests a resource, the access mediator transmits to Alice the AC policy of that resource. This is in effect a query asking Alice whether she holds credentials that satisfy the AC policy. Suppose that Alice holds such credentials, and that they are protected by AC policies. Alice sends a counter query based on these AC policies, asking the access mediator to satisfy these policies first. However, the very act of sending back this counter query strongly suggests to the access mediator that Alice holds the credentials that satisfy the original query. This is because if Alice did not have the credentials, she would behave differently. Thus, information about Alice's possession of credentials—and therefore about attributes she may consider sensitive—is not effectively protected. Granted, an attacker does not learn with certainty that Alice holds a given attribute without first satisfying Alice's AC policies, since the attacker has not seen the actual credentials. However, this should provide Alice little comfort in using the protocols, as the attacker's unauthorized inferences are accurate just in case she adheres to the protocols faithfully.

Trust negotiation is of little value if participants must lie to one another to protect sensitive information, since this would make most negotiations fail unnecessarily. Yet most prior negotiation techniques allow a negotiator's opponent to gain advantage just in case the negotiator is honest. The only existing trust negotiation strategy that is immune from this problem is the eager strategy. In it, each party transmits all credentials whose access control policies have already been satisfied, whether these credentials are related to the eventual negotiation goal or not. In the eager strategy, when a negotiator does not receive a given credential from the opponent, it does not know whether this is because the opponent does not have the credential, or because the negotiator simply has not satisfied the opponent's AC policy for that credential. However, because the eager strategy does not focus the exchange on credentials that are relevant to the authorization decision at hand, it is impractical for most scenarios.

3.1 Acknowledgment Policies

The problem of unauthorized information flow discussed above is caused by the fact that AC policies are associated only with the credentials a negotiator holds. When the negotiator does not hold a credential, she has no associated AC policies, and consequently behaves differently from when she does hold the credential. Thus, from the way a negotiator responds to queries, one can infer whether or not she has relevant credentials.

To solve this problem, one has to protect information about not having a sensitive attribute the same way as one protects information about having one, responding in a uniform way in either case. For this, we now introduce *acknowledgment (Ack) policies*, which a participant establishes in association with attributes that she considers sensitive, whether or not she satisfies those attributes.

What exactly Ack policies should protect is an interesting question. When Alice assigns $B'.r'_1$ to be the Ack policy for the role $B.r_1$, ideally, Alice would want a guarantee that no negotiation opponent can learn by negotiating with her whether or not she satisfies $B.r_1$ without first proving to Alice that the opponent satisfies $B'.r'_1$. However, this is very difficult to guarantee because of three kinds of possible inferential breaches of Ack policy, the first two of which are deductive. In the negative case, where Alice does not satisfy $B.r_1$, suppose an attacker knows $A.r \leftarrow B.r_1$, and asks Alice whether she satisfies $A.r$. If Alice answers no, the attacker would know that Alice does not satisfy $B.r_1$. Similarly, in the positive case, suppose that the attacker knows that $B.r_1 \leftarrow D.r_2$, and asks Alice whether she satisfies $D.r_2$. If Alice answers yes, the attacker would know that Alice satisfies $B.r_1$. The third form of inferential breach arises when credential storage is distributed and negotiators have to collect credentials. In that case, whether or not Alice holds a credential $A.r \leftarrow B.r_1$ might strongly suggest whether or not Alice satisfies $A.r$ or $B.r_1$, since only in the affirmative case is the credential relevant to proving Alice's attributes.

We handle the first of these potential breaches by ensuring Alice acknowledges she does not satisfy $A.r$ only after enforcing her Ack policy for $B.r_1$. In section 5.1, we return to show how we protect against the third potential breach.

In this paper, we handle the second potential breach only partially. In the positive case, when Alice satisfies a sensitive attribute, our policy framework protects this information only to the extent that the attribute is proven directly through use of a type-1 credential. When an attribute can be proven by using other types of credentials (rules), Alice needs to use additional precautions. So, in our framework, if Alice assigns $B'.r'_1$ to be her Ack policy for $B.r_1$, Alice would acknowledge whether or not she has a credential $B.r_1 \leftarrow$ `Alice` only after the opponent (shows it) satisfies

$B'.r'_1$. When Alice considers $A.r$ sensitive and knows that $A.r \longleftarrow B.r_1$, then Alice should make sure that her Ack policy for $B.r_1$ is adequately strong to protect $A.r$ as well as $B.r_1$. The policy framework presented in this paper does not automatically guarantee this. Techniques to make the policy framework automatically provide this guarantee are the subject of future work.

Once it has satisfied Alice's Ack policy for an attribute, the opponent is authorized to know whether Alice satisfies the attribute. For instance, Alice can then disclose that she has no credentials proving she satisfies the attribute. On the other hand, if she has such credentials, even after the opponent has satisfied the Ack policy, Alice may want the opponent to satisfy additional AC policies before she transmits the credentials. This is because knowing that Alice has a credential is different from receiving a copy of it: the credential itself can be used subsequently by the opponent to document facts about Alice to other parties.

Both Ack policies and AC policies are given by attributes that the negotiation opponent must satisfy to gain authorization. We use the following structures to represent these policies. A negotiator A defines a set of sensitive attributes, $\mathsf{sensitiveRoles}_A$. For each $B.r_1 \in \mathsf{sensitiveRoles}_A$, A defines $\mathsf{Ack}_A[B.r_1]$ to be A's Ack policy for $B.r_1$, *e.g.*, $B'.r'_1$. A also defines a set of sensitive credentials, $\mathsf{sensitiveCreds}_A$, which is a subset of the type-1 credentials that prove A satisfies attributes in $\mathsf{sensitiveRoles}_A$. For each $B.r_1 \leftarrow A \in \mathsf{sensitiveCreds}_A$, A defines $\mathsf{AC}_A[B.r_1 \leftarrow A]$ to be A's AC policy for that credential, *e.g.*, $B''.r''_1$.

3.2 An Extended Example Scenario

Example 4 We extend example 2 to include access control policies and acknowledgment policies. We repeat the credentials in example 2 here to be self-contained.
Alice possesses the following credentials:

$$\texttt{MedixFund.pA} \longleftarrow \texttt{Alice} \tag{5}$$
$$\texttt{ReliefNet.coaMember} \longleftarrow \texttt{MedixFund} \tag{6}$$

We assume Alice considers the `MedixFund.pA` attribute to be sensitive, and acknowledges it only to her employer's business partners. She wishes to further protect credential (5) with an AC policy, providing it only to organizations whose security practices are adequate to provide reasonable privacy. For this, we assume that the Better Business Bureau provides a security process auditing service. These two policies are given as follows:

$$\mathsf{Ack}_{\texttt{Alice}}[\texttt{MedixFund.pA}] = \texttt{MedixFund.partner} \tag{7}$$
$$\mathsf{AC}_{\texttt{Alice}}[(5)] = \texttt{BBB.goodSecProcess} \tag{8}$$

MedSup holds the following credentials:

$$\texttt{ReliefNet.coaMember} \longleftarrow \texttt{MedSup} \tag{9}$$
$$\texttt{BBB.goodSecProcess} \longleftarrow \texttt{MedSup} \tag{10}$$

The following are also defined. (We discuss which negotiator holds them in sections 3.3 and 3.4.)

$$\texttt{MedSup.partner} \longleftarrow \texttt{ReliefNet.coaMember} \tag{11}$$
$$\texttt{MedSup.discount} \longleftarrow \texttt{MedSup.partner.pA} \tag{12}$$
$$\texttt{MedixFund.partner} \longleftarrow \texttt{ReliefNet.coaMember} \tag{13}$$

In general, MedSup would also consider some attributes and credentials to be sensitive. However, for the purposes of this example, we assume not.

When Alice requests a discounted sale from MedSup, MedSup responds with the AC policy for that resource, `MedSup.discount`. We formalize this response by what we call a trust target, or just a target:

$$\langle \texttt{MedSup}:\texttt{MedSup.discount} \overset{?}{\twoheadleftarrow} \texttt{Alice} \rangle \tag{14}$$

This target states MedSup's request for a proof that Alice satisfies `MedSup.discount`. We call it the primary target because satisfying it is the central goal of negotiation.

3.3 A (Not So Realistic) Negotiation Process

Let us first consider the credential and trust-target flow in a negotiation where Alice starts with all the credentials she needs to satisfy (14), and MedSup starts with all it needs to satisfy (7), and (8). In other words, Alice holds (5), (6), (11), and (12), and MedSup holds (9), (13), and (10). The negotiation starts after Alice transmits her request for the discount, and after MedSup transmits (14).

1. Alice observes that she could satisfy the primary target by using credentials (5), (6), (11), and (12). However, (5) defines an attribute that Alice considers sensitive. Based on her Ack policy for this attribute, Alice transmits a new target that must be satisfied before she acknowledges that she has the `MedixFund.pA` attribute:
$$\langle \texttt{Alice}:\texttt{MedixFund.partner} \overset{?}{\twoheadleftarrow} \texttt{MedSup} \rangle \tag{15}$$
2. MedSup provides the credentials (9) and (13), thus satisfying target (15).
3. Alice's Ack policy for `MedixFund.pA` now being satisfied, Alice transmits a target based on her AC policy for credential (5):
$$\langle \texttt{Alice}:\texttt{BBB.goodSecProcess} \overset{?}{\twoheadleftarrow} \texttt{MedSup} \rangle \tag{16}$$
4. MedSup provides (10), satisfying target (16).
5. Now MedSup is authorized to receive the credentials that satisfy the primary target. Alice transmits them, and the negotiation terminates with an affirmative authorization decision regarding the discounted sale.

When asked about a sensitive attribute, either explicitly or implicitly, Alice must enforce her Ack policy, whether or not she satisfies the sensitive attribute. So, if she only had available credentials (6), (11), and (12), she would still need to enforce target (15) before doing anything to indicate she does not have credential (5).

3.4 Towards A More Realistic Negotiation Process

In practice, negotiators may both have to contribute credentials to the proof that a given target is satisfied. For instance, Alice should not have to obtain copies of credentials (11) and (12) before the negotiation, since they are local policies of MedSup. Similarly, MedSup should not be expected to obtain credential (13). In this more realistic scenario, Alice holds (5), (6), and (13), while MedSup holds (9), (10), (11), and (12). As part of showing $\texttt{MedSup.discount} \twoheadleftarrow \texttt{Alice}$, (6) and (11) must be combined to show $\texttt{MedSup.partner} \twoheadleftarrow \texttt{MedixFund}$.

It might seem tempting to allow MedSup to use credentials (12) and (11) to reduce (14) to $\langle\texttt{MedSup} : \texttt{ReliefNet.coaMember.pA} \overset{?}{\twoheadleftarrow} \texttt{Alice}\rangle$. However, we avoid this approach when handling targets that involve linked roles: reducing such targets by using type-3 credentials could lead to targets having an unbounded number of linked attributes. In an alternative design, MedSup simply provides credentials (11) and (12) when it establishes the target. However, this requires credential flow that could be avoided.

Consequently, we develop here a design based on a third approach. We allow MedSup to ask Alice to provide any $\texttt{pA}$ credential she may hold. For this, MedSup establishes what we call a linking goal, $\langle\texttt{MedSup} : ?X.\texttt{pA} \overset{?}{\twoheadleftarrow} \texttt{Alice}\rangle$. A linking goal is like a target in which the attribute authority is undetermined. From there, Alice and MedSup negotiate the disclosure of the fact that Alice has the attribute $\texttt{MedixFund.pA}$, as well as the transmission of credential (5), which proves this fact. This proceeds as above, except that Alice uses credential (13) to reduce target (15), posing instead of (15), $\langle\texttt{Alice} : \texttt{ReliefNet.coaMember} \overset{?}{\twoheadleftarrow} \texttt{MedSup}\rangle$. Then MedSup poses $\langle\texttt{MedSup} : \texttt{ReliefNet.coaMember} \overset{?}{\twoheadleftarrow} \texttt{MedixFund}\rangle$, eliciting credential (6) from Alice. At that point MedSup can verify that the primary target, (14), is satisfied.

To ensure that Alice's response to the linking goal does not make unauthorized, implicit disclosures of whether or not she satisfies a sensitive attribute, Alice negotiates disclosure of whether or not she satisfies all attributes of the form $B.\texttt{pA}$ that she considers sensitive.

4 The Trust Target Graph Protocol

In this section, we introduce the trust-target graph protocol. In the this protocol, a trust negotiation process involves the two negotiators working together to construct a *trust-target graph* (TTG). A TTG is a directed graph. Each node is either a trust target or a linking goal. When a requester requests access to a resource, the access mediator and the requester enter into a negotiation process. The access mediator creates a TTG containing one target, which we call the *primary target*. The access mediator then tries to process the primary target, and sends the partially processed TTG to the requester. In each following round, one negotiator receives from the other new information about changes to the TTG, verifies that the changes are legal, and updates its local copy of the TTG accordingly. The negotiator then tries to process some nodes, making its own changes to the graph, which it then sends to the other party, completing the round. The negotiation succeeds when the primary target is satisfied; it fails when the primary target is failed, or when a round occurs in which neither negotiator changes the graph.

This protocol is similar to the Disclosure Tree protocol in [11]. However, it supports a realistic ABAC language, while the Disclosure Tree protocol supports only propositional language. Also, in the next section we show how the TTG protocol supports the use of Ack policies to protect credential possession information.

4.1 Nodes in a Trust-Target Graph

A node in a TTG is one of the three kinds of targets or a linking goal, defined as follows. Nodes are unique.

- A *standard target* takes the form $\langle V : f \overset{?}{\twoheadleftarrow} S\rangle$, in which V is one of the negotiators, f is a role or a linked role, and S is an entity. S is often $opp(V)$, the negotiator opposing V, but it can be any entity. This target means that V wants to see the proof of $f \twoheadleftarrow S$.
- An *intersection target* takes the form $\langle V : (A_1.r_1 \cap \cdots \cap A_k.r_k) \overset{?}{\twoheadleftarrow} S\rangle$. This means that V wants to see the proof of $A_1.r_1 \cap \cdots \cap A_k.r_k \twoheadleftarrow S$.
- A *trivial target* takes the form $\langle V : S \overset{?}{\twoheadleftarrow} S\rangle$, in which V is one of the negotiators, and S is an entity. Trivial targets provide placeholders for edges in the TTG.
- A *linking goal* takes the form $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$, meaning V wants to see a proof of $B.r_2 \twoheadleftarrow S$ for every entity B for which it holds. (See section 3.4.)

In each of the above forms of targets or linking goal, we call V the *verifier*, and S the *subject* of this node.

Each target has a *satisfaction state*, which has one of three values: *satisfied*, *failed*, or *unknown*. Each linking goal has a *completion state*, which has one of two values: *complete* or *incomplete*. Being complete means that all solutions to this goal have been determined.

Each node (a target or a goal) also has a *processing state*, which is a pair of boolean states: verifier-processed and opponent-processed. A node is *verifier-processed* when the verifier cannot process the node any further, *i.e.*, the verifier cannot add any new child to the node. A node is *opponent-processed* when the opponent cannot process the node any further. When a node is both verifier-processed and opponent-processed, we say that it is *fully processed*.

4.2 Edges in a Trust-Target Graph

Six kinds of edges are allowed in a trust-target graph, listed below. Edges are unique. Each kind of edge has its own requirements for being justified. We use $\hookleftarrow$ to represent edges in TTG's.

- A *standard implication edge* takes the form $\langle V : f \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : e \overset{?}{\twoheadleftarrow} S\rangle$, in which e is any role expression and f is a role or a linked role. A standard implication edge has to end at a standard target, but can start from any target. We call $\langle V : e \overset{?}{\twoheadleftarrow} S\rangle$ a standard implication child of $\langle V : f \overset{?}{\twoheadleftarrow} S\rangle$. (We use similar "child" terminology for other kinds of edges.) An edge always points from the child to the parent.

  A standard implication edge is *justified* if the edge is accompanied by a credential chain that proves $f \twoheadleftarrow e$.

- A *linking-monitor edge* takes the form $\langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$.

  This edge means that in order for V to determine whether $A.r_1.r_2 \twoheadleftarrow S$, V wants to know all the B's that make $B.r_2 \twoheadleftarrow S$ true.

  A linking-monitor edge is always justified.

- A *linking-solution edge* takes the form $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$. This is the only kind of edge going into a linking goal.

  A linking-solution edge is always justified.

- A *linking implication edge* takes the form $\langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : A.r_1 \overset{?}{\twoheadleftarrow} B\rangle$.

  This linking implication edge is justified when $\langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ has a linking-monitor child $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$, and $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ has a link-solution child $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ that is satisfied.

- An *intersection edge* takes the form $\langle V : (A_1.r_1 \cap \cdots \cap A_k.r_k) \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : A_i.r_i \overset{?}{\twoheadleftarrow} S\rangle$, where i is in $1..k$.

  An intersection edge is always justified.

- A *control edge* takes the form $\langle V : f \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle opp(V) : f' \overset{?}{\twoheadleftarrow} V\rangle$. Control edges are used for handling acknowledgment and access control policies.

  A control edge is always justified.

4.3 Messages in the Protocol

As described above, negotiators cooperate through use of the protocol in constructing a shared TTG, a copy of which is maintained by each negotiator. Negotiators alternate transmitting messages that each contains a sequence of TTG update operations and a set of credentials to be used in justifying implication edges. On receiving a update operation, a negotiator verifies it is legal before updating its local copy of the shared TTG. The following are *legal* TTG update operations:

- Initialize the TTG to contain a given primary TT, specifying a legal initial processing state for this node. (See below.)
- Add a justified edge (not already in the graph) from a TT that is not yet in the graph to one that is, specifying a legal initial processing state for the new node. The new TT is added to the graph as well as the edge.
- Add a justified edge (not already in the graph) from an old node to an old node.
- Mark a node processed. If the sender is the verifier, this marks the node verifier-processed; otherwise, it marks it opponent-processed.

The legal initial processing state of a trivial target is fully-processed. A linking goal must be verifier-processed. An intersection target must be verifier-processed or opponent-processed. A standard target can take any state.

These operations construct a connected graph. Satisfaction state of trust targets and completion state of linking goals are not transmitted in messages; instead, each negotiation party infers them independently. The satisfaction-state rules presented in the next section ensure that negotiators using the protocol always reach the same conclusions regarding node satisfaction.

4.4 Trust Target Satisfaction State Propagation

We now describe how to determine the satisfaction state of targets and the completion state of linking goals.

Standard target. The initial satisfaction state a standard target is unknown. It becomes satisfied when one of its implication children is satisfied. It becomes failed when it is fully processed and either it has no implication child, or all of its implication children are failed.

Intersection target. The initial satisfaction state of an intersection target is unknown. It becomes satisfied when it is fully processed and all of its children are satisfied. It becomes failed when one of its children is failed.

Trivial target. A trivial target is always satisfied.

Linking goal. The initial completion state of a linking goal is incomplete. It becomes complete when it is fully processed and all of its linking-solution children are either satisfied or failed.

The legal update operations do not remove nodes or edges once they have been added, and once a node is fully processed, it remains so thereafter. Consequently, once a target becomes satisfied or failed, it retains that state for the duration of the negotiation.

5 Node Processing

In the previous section, we described the TTG negotiation protocol, in which two negotiators exchange update messages. The protocol defines what updates are legal, and the receiver of a message can verify that the updates in the message is legal. This section describes procedures for *correct processing*, which update the TTG in a manner designed to satisfy the primary target whenever this is possible, while enforcing each negotiator's Ack and AC policies. Correct processing continues until either the primary target is satisfied (negotiation success), it is failed (negotiation failure), or neither negotiator can perform a correct update (also negotiation failure). The latter happens if a negotiator is not a member of a necessary role, or if there is a cyclic dependence in the policies of the two negotiators with regard to a necessary role. (Such a cyclic dependence manifests itself in the TTG as a cycle involving at least two control edges.)

Note that a negotiator cannot be forced to follow the correct procedures, and when it does not, the other negotiator may not be able to tell. The protocol and the correct processing procedures are intended to guarantee that a misbehaving negotiator can never gain advantage (either learn information or gain access without satisfying relevant policies first) over a faithful negotiator who follows the protocol and the correct procedures. Therefore, a normal negotiator has no incentive to misbehave. Still, it is always within the power of either negotiator to behave incorrectly, and doing so may prevent the negotiation from succeeding. For instance, either negotiator can simply abort the negotiation at any time.

5.1 Negotiating with Credentials Whose Storage is Distributed

Prior ATN systems have simply assumed that negotiators have all the credentials they will need during negotiation. However, in a realistic environment that uses ABAC and ATN, credential storage is not centralized, and credential discovery should be considered. In this section we consider the following basic problem: which negotiator can be expected to have which credentials, allowing them to take responsibility for adding corresponding edges to the TTG?

We base our approach on the results in [7], where a storage type system and a notion of well-typed credentials were presented to guarantee that credential chains can be discovered even when credentials are stored in a distributed way. Each credential is assumed to be stored by its issuer (an *issuer-stored* credential) or by its subjects (a *subject-stored* credential), where subject and issuer are defined as follows. For a credential, $A.r \leftarrow e$, we call A the *issuer*, and each entity in *base*(e) a *subject* of this credential, where *base*(e) is defined as follows: *base*$(D) = \{D\}$, *base*$(B.r_2) = \{B\}$, *base*$(A.r_1.r_2) = \{A\}$, *base*$(A_1.r_1 \cap \cdots \cap A_k.r_k) = \{A_1, A_2, \ldots, A_k\}$. Each role name has two types: an issuer-side type and a subject-side type. On the issuer side, the possible types are issuer-(traces-)none, issuer-(traces-)def, and issuer-(traces-)all, in order of increasing strength. If r is issuer-def or issuer-all, each credential of the form $A.r \leftarrow e$ is required to be issuer-stored. If r is issuer-all, the well-typing rule for credentials additionally requires e to be issuer-all. (The types for role expressions are determined from role names.) This means that from A, one can retrieve the credential and discover its subject, allowing one to find all issuer stored credentials issued by that subject, and to repeat the process to discover all members of $A.r$. On the subject side, the possible types are subject-(traces-)none and subject-(traces-)all. If r is subject-all, each credential of the form $A.r \leftarrow e'$ is required to be subject-stored, and e' must also be subject-all. To ensure that credentials are either issuer-stored or subject-stored, the well-typing rule also requires that no role expression is both issuer-none and subject-none. A linked role $A.r_1.r_2$ is issuer-all when both r_1 and r_2 are; same is true for subject-all. $A.r_1.r_2$ is issuer-def if r_1 is issuer-def and r_2 is subject-all, or r_1 is issuer-all and r_2 is issuer-def. For more information on the type system, refer to [7].

Here we assume that the above typing system is in place. We additionally assume here that role names are either issuer-none or subject-none. Ack policies can be assigned only to roles that are issuer-none and subject-all, and AC policies can be assigned only to credentials defining such roles. This makes sense because normally an entity D only has control over credentials of the form $A.r \leftarrow D$ when $A.r$ is subject-traces-all.

The typing system ensures that every chain proving $A.r \twoheadleftarrow S$ can be partitioned into a set of issuer-stored credentials that are reachable from A, and a set of subject-stored credentials that are reachable from S. This justifies assuming that each negotiator, V, can discover and use all issuer-stored credentials that can be traced from V's policies. We call this set of credentials $\mathcal{C}^I(V)$. Similarly, we assume that any negotiator, O, can discover and use any subject-stored credential that can be traced from O. We call this set $\mathcal{C}^S(O)$. These two sets, $\mathcal{C}^I(V)$ and $\mathcal{C}^S(O)$, are designed to ensure that every credential relevant to the negotiation is available to either V or O, respectively. In order for O to behave uniformly when O does not satisfy one of the roles in $\mathsf{sensitiveRoles}_O$, $\mathcal{C}^S(O)$ also must also contain each subject-stored credential reachable from a sensitive role. This addresses the third and helps address the first potential inferential breach of Ack policy identified in section 3.1.

5.2 Node Processing State Initialization

When a new node is added to a TTG, its processing state should be initialized as follows:

- A trivial target is fully processed.
- For a standard target, $\langle V : A.r \overset{?}{\twoheadleftarrow} S\rangle$, if r is subject-traces-all, it is verifier-processed, which means that the verifier cannot process it any further; otherwise, it is opponent-processed.
- For a standard target, $\langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle$, if both r_1 and r_2 are subject-traces-all, then it is verifier-processed; otherwise, it is opponent-processed.
- An intersection target is initially opponent-processed, if created by the verifier; it is verifier-processed if created by the opponent of the verifier.
- A linking goal is initially verifier-processed.

5.3 Verifier-Side Processing

We now describe how a negotiator V process a node when it is the verifier of the node. This case needs to consider only those nodes not yet marked verifier-processed.

For V to process a standard target $\langle V : A.r \overset{?}{\twoheadleftarrow} S\rangle$, if V has a credential $A.r \leftarrow B.r_2$ in $\mathcal{C}^I(V)$ (discussed in section 5.1), then V could add $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ as an implication child to $\langle V : A.r \overset{?}{\twoheadleftarrow} S\rangle$, since satisfying the former suffices to satisfy the latter. However, unless $B.r_2$ is subject-traces-all, the opponent cannot process the target $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$, and so there is no point adding it; instead, V can continue its backward search for role expressions e such that $B.r_2 \twoheadleftarrow e$, until it either finds an e that the opponent can process, or it finds an e that affects the logical structure of the TTG, namely an entity, a linked role, or an intersection.

We construct a function that can be used by the verifier to skip over targets that need not be created, and to construct instead a set of targets whose satisfaction would imply the satisfaction of the omitted target. This function is $\mathsf{oppoFrontier} : \mathcal{E}(\mathcal{C}^I(V)) \rightarrow 2^{\mathcal{E}(\mathcal{C}^I(V))}$, where $\mathcal{E}(\mathcal{C}^I(V))$ is the set of role expressions defined in section 2.3. We define $\mathsf{oppoFrontier}$ to be $\mathsf{frontier}[oppo, \mathcal{C}^I(V)]$, in which $oppo$ is a predicate on role expressions such that $oppo(e)$ is true when e is an entity, a subject-traces-all role, a linked role, or an intersection, and $\mathsf{frontier}$ is defined as follows.

We define $\mathsf{frontier}[pred, \mathcal{C}]$, in which $pred$ is a predicate on role expressions and $\mathcal{C}$ is a set of credentials, to be the pointwise-minimal function satisfying the following system of equalities and inequalities:

1. If $pred(e)$ is true, $\mathsf{frontier}[pred, \mathcal{C}](e) = \{e\}$,
2. If $pred(e)$ is not true, and $e \leftarrow e' \in \mathcal{C}$, then $\mathsf{frontier}[pred, \mathcal{C}](e) \supseteq \mathsf{frontier}[pred, \mathcal{C}](e')$.

This least-solution construction is well-defined because of two factors. First, the function space is a finite lattice under the pointwise subset ordering. Second, the system of equalities and inequalities can easily be converted into a monotonic function-valued function whose least fixpoint, which is known to exist, is the system's least solution. By extending the graph-based approach in [7], we have implemented an efficient, goal-oriented search algorithm that computes $\mathsf{frontier}[pred, \mathcal{C}]$ for any given $pred$ and $\mathcal{C}$.

We now describe how a verifier correctly processes a node. Remember that a node is processed according to the following procedure only when it is not already verifier-processed. What we present here are constraints for correct processing. One strategy is to carry out correct processing steps as soon as one can, but other strategies are possible.

Processing $T = \langle V : A.r \overset{?}{\twoheadleftarrow} S\rangle$
(1) For each $A.r \leftarrow e \in \mathcal{C}^I(V)$ and each $e' \in \mathsf{oppoFrontier}(e)$, V can add an implication edge $T \hookleftarrow \langle V : e' \overset{?}{\twoheadleftarrow} S\rangle$.

(2) V can mark T as verifier-processed only after (1) is *done*, meaning that all edges that can be added according to (1) have been added.

Processing $T = \langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ **when** r_1 **is issuer-traces-all**
(1) For each entity B such that $A.r_1 \twoheadleftarrow B$, and for each e such that $e \in \mathsf{oppoFrontier}(B.r_2)$, V can add a standard implication edge, $T \hookleftarrow \langle V : e \overset{?}{\twoheadleftarrow} S\rangle$.
(2) V can mark T as verifier-processed only after (1) is done.

Processing $T = \langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ **when** r_1 **is issuer-traces-def**
(1) V can add the (unique) linking-monitor edge, $T \hookleftarrow \langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$.
(2) After (1) is done, V can add a linking-implication edge $T \hookleftarrow \langle V : A.r_1 \overset{?}{\twoheadleftarrow} B\rangle$ for each linking-solution edge $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ such that $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ is satisfied.
(3) V can mark T as verifier-processed only after both (1) and (2) are done (*i.e.,* all $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ are added and satisfied).

Processing $T = \langle V : A_1.r_1 \cap \cdots \cap A_k.r_k \overset{?}{\twoheadleftarrow} S\rangle$

(1) V can add the k intersection edges, $T \hookleftarrow \langle V : A_j.r_j \overset{?}{\twoheadleftarrow} S\rangle$, $1 \leq j \leq k$
(2) V can mark T verifier-processed only after (1) is done.

5.4 Opponent-Side Processing

We now describe how a negotiator O should process a node when O is the opponent of the verifier of the node. Here, $\mathcal{C}^S(O)$ is taken to be the set of credentials introduced in section 5.1.

We define $\mathsf{simpFrontier}$ to be $\mathsf{frontier}[simp, \mathcal{C}^S(O)]$, in which $simp(e)$ is true when e is not a role, and $\mathsf{frontier}$ is defined in section 5.3. We define $\mathsf{sensFrontier}$ to be $\mathsf{frontier}[sens, \mathcal{C}^S(O)]$, in which $sens(e)$ is true when e is not a non-sensitive role.

O uses the following rules to process nodes that are not yet marked opponent-processed.

Processing $T = \langle V : A.r \overset{?}{\twoheadleftarrow} O\rangle$ when $A.r$ is not sensitive
(1) If $O \in \mathsf{sensFrontier}(A.r)$, then O can add an implication edge $T \hookleftarrow \langle V : O \overset{?}{\twoheadleftarrow} O\rangle$.
(2) When $O \notin \mathsf{sensFrontier}(A.r)$, for each e in $\mathsf{sensFrontier}(A.r)$ that is not an entity, O can add an implication edge $T \hookleftarrow \langle V : e \overset{?}{\twoheadleftarrow} O\rangle$.
(3) O can mark T as opponent-processed if T is satisfied, or both (1) and (2) are done.

Processing $T = \langle V : A.r \overset{?}{\twoheadleftarrow} O\rangle$ when $A.r$ is sensitive
(1) O can add a control edge $T \hookleftarrow \langle O : e_{Ack} \overset{?}{\twoheadleftarrow} V\rangle$, where $e_{Ack} = \mathsf{Ack}_O[A.r]$.
(2) After (1) is done and $\langle O : e_{Ack} \overset{?}{\twoheadleftarrow} V\rangle$ is satisfied, if O has the credential $A.r \leftarrow O \in \mathcal{C}^S(O)$, and if $e_{AC} = \mathsf{AC}_O[A.r \leftarrow O]$ is defined, O can add the control edge $T \hookleftarrow \langle O : e_{AC} \overset{?}{\twoheadleftarrow} V\rangle$.
(3) After (2) is done and $\langle O : e_{AC} \overset{?}{\twoheadleftarrow} V\rangle$ (if it exists) is satisfied, O can add the implication edge $T \hookleftarrow \langle V : O \overset{?}{\twoheadleftarrow} O\rangle$.
(4) For each e in $\mathsf{sensFrontier}(A.r)$ that is not an entity, O can add an implication edge $T \hookleftarrow \langle V : e \overset{?}{\twoheadleftarrow} O\rangle$.
(5) O can mark T as opponent-processed if T is satisfied, or all of the above steps are done.

Processing $T = \langle V : A.r \overset{?}{\twoheadleftarrow} S\rangle$ when $S \neq O$
(1) If $S \in \mathsf{simpFrontier}(A.r)$, then O can add an implication edge $T \hookleftarrow \langle V : S \overset{?}{\twoheadleftarrow} S\rangle$.
(2) If $S \notin \mathsf{simpFrontier}(A.r)$, for each $e \in \mathsf{simpFrontier}(A.r)$ that is not an entity, O can add an implication edge $T \hookleftarrow \langle V : e \overset{?}{\twoheadleftarrow} S\rangle$.
(3) O can mark T as opponent-processed if T is satisfied, or both (1) and (2) are done.

Processing $T = \langle V : A.r_1.r_2 \overset{?}{\twoheadleftarrow} S\rangle$
Note that S may or may not be O.
(1) O can add the linking-monitor edge $T \hookleftarrow \langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$.
(2) After (1) is done, O can add a linking-implication edge $T \hookleftarrow \langle V : A.r_1 \overset{?}{\twoheadleftarrow} B\rangle$ for each linking-solution edge $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ such that $\langle V : B.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ is satisfied.
(3) O can mark T opponent-processed when T has a linking-monitor child $\langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$ whose completion state is complete and (2) is done.

Processing $G = \langle V : ?X.r_2 \overset{?}{\twoheadleftarrow} S\rangle$:
(1) O can add a linking-solution edge, $G \hookleftarrow \langle V : A.r_2 \overset{?}{\twoheadleftarrow} S\rangle$, for any $A.r_2 \in \mathsf{sensitiveRoles}_O$ or $A.r_2$ defined by a credential in $\mathcal{C}^S(O)$.
(2) O can mark G opponent-processed only after (1) is done.

Processing $T = \langle V : A_1.r_1 \cap \cdots \cap A_k.r_k \overset{?}{\twoheadleftarrow} S\rangle$
(1) O can add the k intersection edges, $T \hookleftarrow \langle V : A_j.r_j \overset{?}{\twoheadleftarrow} S\rangle$, $1 \leq j \leq k$.
(2) O can mark T opponent-processed after (1) is done.

Notice that if O considers $B.r_1$ sensitive and does not satisfy $B.r_1$, the above processing rules protect that information against the first form of deductive breach of Ack policy, identified in section 3.1 (*i.e.*, the negative case). For instances, if O should have $A.r \leftarrow B.r_1$, and V should establish the target, $\langle V : A.r \overset{?}{\twoheadleftarrow} O\rangle$, O does not mark this target processed until it has added the implication child, $\langle V : B.r_1 \overset{?}{\twoheadleftarrow} O\rangle$. O will then add a control child to the latter target using O's Ack policy for $B.r_1$. Consequently, $\langle V : A.r \overset{?}{\twoheadleftarrow} O\rangle$ will not become failed until the Ack policy for $B.r_1$ has been satisfied.

6 On-going and Future Work

We are in the process of implementing a trust negotiation system that uses the TTG protocol.

6.1 Termination, Correctness, and Completeness

We have designed the TTG protocol and processing rules with termination, correctness, and completeness concerns in mind. We plan to formalize and prove the following properties in an extended version of this paper:

Termination. The number of nodes in the TTG is clearly polynomial in the number of credentials held by the two negotiators. No node or edge is added more than once, and each negotiator marks each node as processed at most once. So, because negotiators limit the number of empty messages they allow, negotiation will terminate.

Correctness. If a negotiator follows the TTG protocol and the processing rules presented here, other entities cannot unduly gain information through negotiation with the negotiator.

Completeness. Consider any pair of negotiators. Suppose there is a sequence of credential exchanges with the following safety property: for each credential that is transmitted, if it is governed by Ack or AC policies, those policies have been satisfied by credentials available to the sender through prior transmissions in the same negotiation. Then, if both negotiators follow the TTG protocol and the node processing rules, the negotiation process will succeed.

Strong Completeness. In addition to the above, if well-typed credentials exist that, if available to two negotiators, would enable them to negotiate successfully, then, between them, the negotiators can collect such credentials prior to exchanging any information with one another.

6.2 Strategy

The rules presented in section 5 above constrain, but do not determine the total order in which the nodes of the TTG are constructed, processed, and marked fully-processed. Many different orders are possible and the selection among them may reflect the disposition of the negotiators, such as their eagerness to negotiate efficiently versus their conservatism about disclosing their own credentials. The TTG implementation we are building as this paper goes to press uses an eager strategy.

6.3 Breaches of Ack Policy by Positive Inference

As part of on-going work, we are seeking solutions to the second form of inferential breach identified in section 3.1. Preliminary study suggests that we may need negotiators to collect credentials from third parties during the negotiation process to prevent such breaches.

7 Conclusion

We have presented the trust-target graph protocol. Unlike previous ATN work, our protocol is based on a credential language that is realistic, by criteria we have discussed. We have presented an acknowledgment policy architecture that solves a problem common to prior ATN systems, that negotiators control the flow only of credentials, and not the sensitive attribute information contained in these credentials. We have also presented negotiation procedures, based on the TTG protocol and the acknowledgment policy architecture, that support a realistic credential language and distributed storage of credentials, and that protect negotiators' sensitive attributes from unauthorized disclosure.

Acknowledgments

This work is supported by DARPA through SPAWAR contracts N66001-01-C-8005 and N66001-00-C-8015. We thank the anonymous reviewers for their helpful suggestions.

References

[1] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.

[2] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-7)*, pages 134–143. ACM Press, November 2000.

[3] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[4] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.

[5] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 2–14. IEEE Computer Society Press, May 2000.

[6] 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*. IEEE Computer Society Press, May 2002.

[7] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management (extended abstract). In *Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 156–165. ACM Press, November 2001.

[8] Kent E. Seamons, Marianne Winslett, and Ting Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS'01)*, February 2001.

[9] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*. IEEE Press, January 2000.

[10] Ting Yu, Xiaosong Ma, and Marianne Winslett. Prunes: An efficient and complete strategy for trust negotiation over the internet. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-7)*, pages 210–219, November 2000.

[11] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 146–155. ACM Press, November 2001.