TRUST-X: A PEER-TO-PEER FRAMEWORK FOR TRUST ESTABLISHMENT

by E. Bertino, E. Ferrari, A.C. Squicciarini

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Trust-$\mathcal{X}$: A Peer-to-Peer Framework for Trust Establishment

Elisa Bertino, *Fellow*, *IEEE*, Elena Ferrari, *Member*, *IEEE*, and Anna Cinzia Squicciarini

Abstract—In this paper, we present Trust-$\mathcal{X}$, a comprehensive XML-based [12] framework for trust negotiations, specifically conceived for a peer-to-peer environment. Trust negotiation is a promising approach for establishing trust in open systems like the Internet, where sensitive interactions may often occur between entities at first contact, with no prior knowledge of each other. The framework we propose takes into account all aspects related to negotiations, from the specification of the profiles and policies of the involved parties to the selection of the best strategy to succeed in the negotiation. Trust-$\mathcal{X}$ presents a number of innovative features, such as the support for protection of sensitive policies, the use of trust tickets to speed up the negotiation, and the support of different strategies to carry on a negotiation. In this paper, besides presenting the language to encode security information, we present the system architecture and algorithms according to which negotiations can take place.

Index Terms—Security and protection, access controls, and trust negotiation.

1 Introduction

The extensive use of the Web for exchanging information and requesting or offering services requires extensively redesigning the way access control is usually performed. In a conventional system, the identity of all possible requesting subjects is known in advance and can be used for access control. This simple paradigm is, however, not suitable for an environment like the Web, where most of the interactions occur between strangers that do not know each other before interaction takes place. A promising approach to trust establishment is represented by *trust negotiation* [5], according to which trust is established through an exchange of digital credentials. Because, however, credentials may contain sensitive and private information, disclosure of credentials also must be protected through the use of policies that specify which credentials must be received before the requested credential can be disclosed.

Several approaches to trust negotiation have been proposed so far [3], [4], [8], [9], [11]. We survey work closest with ours in Section 7. However, all these proposals mainly focus on one of the aspects of trust negotiation, such as, for instance, policy and credential specification, or the selection of the negotiation strategy, but none of them provide a comprehensive solution to trust negotiation, taking into account all phases of a negotiation process. To overcome such drawback, in this paper, we propose Trust-$\mathcal{X}$, an XML-based system addressing all the phases of a negotiation and providing novel features with respect to existing approaches. Trust-$\mathcal{X}$ has been specifically designed for a peer-to-peer environment in that both the negotiating parties are equally responsible for negotiation management and can both drive the negotiation process, by selecting the strategy that better fits their needs. Additionally, each party is equipped with the same functional modules and, thus, it can alternatively act as a requester or resource controller during different negotiations.

The first component of Trust-$\mathcal{X}$ is an XML-based language, named $\mathcal{X}$-TNL, for specifying certificates and policies. We believe the availability of a standard and expressive language for expressing security information is essential for providing a comprehensive and widely-usable environment for carrying on trust negotiations. Trust-$\mathcal{X}$ certificates are either credentials or declarations, where a credential states personal characteristics of its owner, certified by a Credential Authority (CA), whereas declarations collect personal information about its owner that do not need to be certified (such as, for instance, specific preferences) but may help in better customizing the offered service. A novel aspect of $\mathcal{X}$-TNL is the support for *trust tickets*. Trust tickets are issued upon the successful completion of a negotiation and can be used to speed up subsequent negotiations for the same resource. Additionally, $\mathcal{X}$-TNL allows the specification of a wide range of policies and provides a mechanism for policy protection, based on the notion of *policy preconditions*. A Trust-$\mathcal{X}$ negotiation consists of a set of phases to be sequentially executed. A relevant aspect of Trust-$\mathcal{X}$ is that it provides a variety of strategies for trust negotiations, which allow one to better trade off between efficiency and protection requirements. The motivation behind this choice is that, since trust negotiations can be executed for several types of resources and by a variety of entities having various security requirements and needs, a single approach to perform negotiation processes may not be adequate in all the circumstances. As a result, Trust-$\mathcal{X}$ is very flexible and can support negotiations in a variety of scenarios, involving entities like business, military and scientific partners, or companies and their cooperating partners or customers. To the best of our knowledge, Trust-$\mathcal{X}$ is the first attempt of providing such comprehensive solution to trust negotiation.

- *E. Bertino and A. Squicciarini are with the Dipartimento di Scienze dell'Informazione, Universita' degli Studi di Milano. E-mail: {bertino, squicciarini}@dico.unimi.it.*
- *E. Ferrari is with the Dipartimento di Scienze Chimiche, Fisiche e Matematiche, Universita' degli Studi dell'Insubria, Como. E-mail: elena.ferrari@uninsubria.it.*

Manuscript received 24 May 2003; revised 11 Sept. 2003; accepted 10 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-0069-0503.

A preliminary version of the language provided by Trust-$\mathcal{X}$ appeared in [1]. However, [1] presents only a limited subset of the language proposed in this paper, mainly focusing on certificate encoding, whereas trust tickets and policy preconditions are novel aspects that have not been presented in [1]. Additionally, the work reported in [1] does not deal with negotiation management, but it focuses only on the negotiation language.

The remainder of this paper is organized as follows: Next, Section 2 presents the basic structure and components of the Trust-$\mathcal{X}$ framework. Section 3 introduces the running example we use throughout the paper. Section 4 summarizes the XML language we have developed for encoding certificates and disclosure policies. Section 5 outlines the various phases of a Trust-$\mathcal{X}$ negotiation process. Section 6 focuses on the key phase of a Trust-$\mathcal{X}$ negotiation, that is, *Policy evaluation phase*. Section 7 surveys related work, whereas Section 8 concludes the paper and outlines future research directions. The paper also contains two Appendices (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm): Appendix A reports XML encoding of disclosure policies, whereas Appendix B reports formal proofs.

2 THE Trust-$\mathcal{X}$ FRAMEWORK

The reference scenario for Trust-$\mathcal{X}$ negotiations is a network composed of different entities that interact with each other in order to obtain and/or offer resources controlled by the entities themselves. The notion of resource comprises both sensitive information and services, whereas the notion of entity includes users, processes, and servers. Entities are identified through credentials typically issued by CAs. Moreover, in our system, "private" entities can also issue credentials (signed by their private key) to third trusted parties to delegate part of their authority. Each entity can be the controller of one or more resources, a third-party credential issuer, or a requester. Typically, a negotiation involves two entities: the entity providing negotiated resources, referred to as the *controller*, and the entity wishing to access the resources, referred to as *requester*. Note that the *controller* does not necessarily coincide with the *owner* of the resource, it may be the manager of the resource entitled by the real owner. Each entity, characterized by a Trust-$\mathcal{X}$ profile of certificates,[1] can act as a requester in one negotiation and as a controller in another. During a negotiation, mutual trust might be established between the controller and the requester: The requester has to show its certificates to obtain the resource, and the controller, whose honesty is not always assured, submits certificates to the counterpart in order to establish trust before receiving sensitive information. Release of information is regulated by disclosure policies, which are exchanged to inform the other party of the trust requirements that need to be satisfied to advance the state of the negotiation. Trust-$\mathcal{X}$ participants are both considered equally important, therefore, each party has an associated system managing negotiation and always has a complete view of the state of the negotiation process.

1. We assume that both parties are Trust-$\mathcal{X}$ compliant, but it is possible to enforce negotiations even between parties that do not adopt the same negotiation language, simply by adding a translation mechanism to guarantee semantic conversion of possibly heterogeneous certificates.

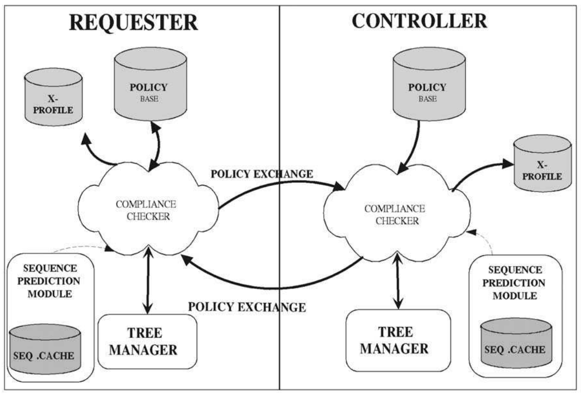


Fig. 1. Architecture for Trust-$\mathcal{X}$ negotiation.

The main components of the Trust-$\mathcal{X}$ architecture are shown in Fig. 1. The architecture is symmetric and peer to peer. The main functions supported by the architecture are: supporting policy exchange, testing whether a policy might be satisfied, supporting certificate and trust ticket[2] exchange, and caching of sequences. Each of these functions is executed by a specific module of the Trust-$\mathcal{X}$ system. Facets modules may also be added to make the negotiation easier and faster. The system is composed of a *Policy Base*, storing disclosure policies, the $\mathcal{X}$*-Profile* associated with the party, a *Tree Manager*, storing the state of the negotiation, and a *Compliance Checker*, to test policy satisfaction and determine request replies. Note that the *Compliance Checker* also includes a credential verification module, which performs a validity check of the received credentials in order to verify the document signature, check for credential revocation, and discovery credential chain, if necessary. Finally, Trust-$\mathcal{X}$ system has a complimentary module named *Sequence prediction module*, for caching and managing previously used trust sequences.

3 RUNNING EXAMPLE

In this section, we briefly introduce a scenario we will use throughout the paper to demonstrate Trust-$\mathcal{X}$ application.

The scenario we refer to is that of an online rental car agency, named *Cars*, providing rental cars of vehicles. *Cars* adopts different policies for the renting of vehicles, based on the characteristics of the requesting user. More precisely, *Cars* allows free rental for the employees of `Corrier` company, which belongs to the same holding of *Cars*. By contrast, rental service is available on payment for unknown requesters, who have to submit, before being allowed to rent a vehicle, a digital copy of their driving licence and then a valid credit card. `Corrier` company employees do not have to submit any document since *Cars* maintains a digital copy of their driving licences. We assume that *Cars* company and all the entities interacting with *Cars* are characterized by a profile of certificates. Additionally, service requesters may optionally posses some declarations (for instance, recording their car preferences). As the paper proceeds, we will use this scenario as a running example to show Trust-$\mathcal{X}$ main functionalities.

2. We elaborate on the notion of trust ticket in Section 4.2.

```
<Corrier_Employee credID='12ab',
  SENS= 'NORMAL' >
<Issuer HREF='http://www.Corrier.com'
  Title=Corrier_Employees_Repository/>
  <name>
    <Fname> Olivia </Fname>
    <lname > White </lname>
  </name>
  <address>
    Grange Wood 69 Dublin
  </address>
  <employee_number code=34ABN/>
  <position> driver   </position>
</Corrier_Employee>
```

(a)

```
<car_preferences >
  <name >
    <Fname> Olivia </Fname>
    <lname > White </lname>
  </name>
  <car_category> compact   </car_category>
    < vehicle>
      <model> TOYOTA COROLLA 1.4 </model>
      <model> FIAT PUNTO 1.2 </model>
    </vehicle>
      <description >
        <option > Air Conditioning </option>
        <option > ABS </option>
        <option > car radio </option>
      </description>
</car_preferences>
```

(b)

Fig. 2. Examples of (a) an $\mathcal{X}$-TNL credential and (b) a declaration.

4 $\mathcal{X}$-TNL

$\mathcal{X}$-TNL is the XML-based language we have developed for specifying Trust-$\mathcal{X}$ certificates and disclosure policies. Expressing credentials and security policies using XML has several advantages. First, the protection of Web data and their security related information is uniform, in that credentials and policies are XML documents and, thus, can be protected using the same mechanisms developed for the protection of conventional XML documents. Furthermore, the use of an XML formalism for specifying credentials facilitates credential submission and distribution, as well as their analysis and verification by use of a standard query language such as XQuery [12].

4.1 Certificates and $\mathcal{X}$-$Profiles$

$\mathcal{X}$-TNL *certificates* are the means to convey information about the profile of the parties involved in the negotiation. A certificate can be either a *credential* or a *declaration*. A credential is a set of properties of a party certified by a CA and digitally signed by the issuer, according to the standard defined by W3C for XML Signatures [12]. Note that, incorporating digital signatures into credentials allows verification of entity credentials against forgery. Fig. 2a shows an example of a credential, containing the basic information about an employee of `Corrier` company. By contrast, declarations are a set of data without any certification. These kind of certificates are stated by the owner and provide auxiliary information that can help the negotiation process. Intuitively, declarations will be used only for unsensitive information exchange. For instance, a declaration named `car_preferences` (see Fig. 2b) describes the car preferences of a given subject. This declaration can be used to communicate Olivia's personal preferences during negotiation with $Cars$ company.

Both credentials and declarations are encoded using XML. More precisely, in $\mathcal{X}$-TNL, a credential type is modeled as a DTD and a credential as a valid document with respect to the corresponding DTD. The same formalism is used for declarations. Furthermore, $\mathcal{X}$-TNL addresses the problem of vocabulary agreement, by using XML Namespaces [12]. The use of namespaces combined with the certificate type system helps trust negotiation software to correctly interpret different credentials, even if they are issued by different entities which do not share a common ontology.

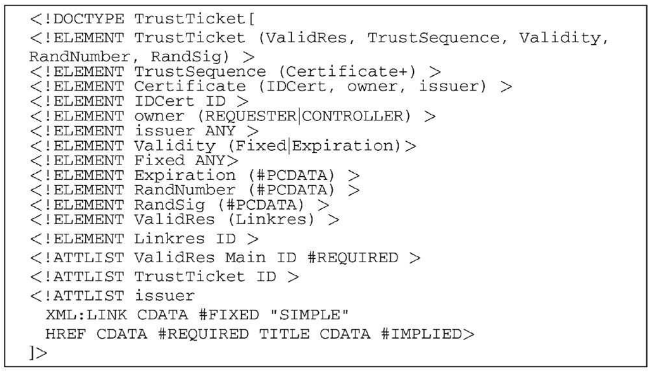
All certificates associated with a party are collected into a unique XML document, called $\mathcal{X}$-$Profile$. To better structure credentials and declarations into an $\mathcal{X}$-$Profile$, each $\mathcal{X}- - Profile$ is organized into $Data\ sets$. Each data set collects a class of credentials and declarations referring to a particular aspect of the life of their owner. For instance, `Demographic_Data`, `Working Experience` are examples of possible data sets.[3] Data sets can then be used to refer to a set of logically related certificates as a whole, and this can facilitate their evaluation and exchange during negotiation.

4.2 Trust Tickets

Although the scenario we refer to is an open environment, it is really likely that in many cases similar negotiations will be executed several times between the same parties. To enforce both trust and efficient negotiations, $\mathcal{X}$-TNL supports the notion of *trust ticket*. Trust tickets are a powerful means to reduce as much as possible the number of certificates and policies that need to be exchanged during negotiations. Trust tickets are generated by each of the involved parties at the end of a successful negotiation and issued to the corresponding counterpart. Like conventional certificates, trust tickets are locally stored by their owners into their $\mathcal{X}$-$Profile$, in a specific data set. A trust ticket certifies that parties have already successfully ended a negotiation for that resource, that is, they possess the necessary certificates to acquire the resource. The idea is that, when a party asks for a resource, it can first be asked for a trust ticket for that resource. In this way, parties can easily prove that they are trusted entities and, thus, they have all the requirements for successfully negotiating the requested resource.

According to our XML compliant formalism, a trust ticket is a valid XML document conforming to the DTD shown in Fig. 3. The resource identifier is stored into a mandatory attribute of element `ValidRes`. The ticket can also be used to facilitate negotiations of resources related to the one it was issued for. The identifiers of such resources are stored in the children elements of element `ValidRes`. For instance, suppose that a flight customer contacts a travel agency where he/she has recently booked a trip to buy an additional service (the rental of a car, say) for the same trip. The requester, instead of executing a complete negotiation process for the rental, can just show his/her valid trust ticket to the agency, proving that he is a known customer and that he/she has the requirements to get the service. The core of a trust ticket is the sequence of certificates that lead to negotiation success, which is specified in the `TrustSequence` element. A trust sequence

3. As for credentials, we assume that data set names are unique, and they are registered through some central organization.



```
<!DOCTYPE TrustTicket[
<!ELEMENT TrustTicket (ValidRes, TrustSequence, Validity,
RandNumber, RandSig) >
<!ELEMENT TrustSequence (Certificate+) >
<!ELEMENT Certificate (IDCert, owner, issuer) >
<!ELEMENT IDCert ID >
<!ELEMENT owner (REQUESTER|CONTROLLER) >
<!ELEMENT issuer ANY >
<!ELEMENT Validity (Fixed|Expiration)>
<!ELEMENT Fixed ANY>
<!ELEMENT Expiration (#PCDATA) >
<!ELEMENT RandNumber (#PCDATA) >
<!ELEMENT RandSig (#PCDATA) >
<!ELEMENT ValidRes (Linkres) >
<!ELEMENT Linkres ID >
<!ATTLIST ValidRes Main ID #REQUIRED >
<!ATTLIST TrustTicket ID >
<!ATTLIST issuer
  XML:LINK CDATA #FIXED "SIMPLE"
  HREF CDATA #REQUIRED TITLE CDATA #IMPLIED>
]>
```

Fig. 3. Trust ticket DTD.

determines a list of certificates where the disclosure of each certificate in the list represents a condition for a trusted release of one of the certificates following it in the list.[4] This element has as many subelements, named `Certificate`, as the number of certificates in the sequence. Each certificate is represented by its unique ID, its owner (one of the two parties involved in the negotiation) and the link to the issuer repository, to allow the counterpart to check the ticket validity (i.e., that it has not been revoked). Each trust ticket has a validity time denoting the time-interval within which the given ticket can be used. The duration depends on the parties preferences and needs. To make the specification of the validity period as flexible as possible, we provide two different alternatives. The first is to establish with the counterpart a validity period, typically 24/48 hours.[5] Alternatively, since the corresponding trust sequence is valid until one of its certificates expires, it is possible to fix the expiration date to the first expiration date of all the certificates of the trust sequence.

Trust is established between parties employing a challenge-response protocol [7]. Each trust ticket contains a random number, stored in the element `Randnumber`. Such number is digitally signed by the issuer of the trust ticket, and the signature is stored in the `RandSig` element. To authenticate each other, each party first sends the `Randnumber` stored in its trust ticket and then verifies if the number signature coincides with the signature in the `RandSig` element. Note that a trust ticket can be used several times before it expires. However, to better guarantee trust, once the random number is exchanged, the certificate is updated generating (and signing) a new number for each party, eventually, also updating the expiration date.

Note that the aim of trust tickets is completely different from that of tickets used by authentication services (e.g., Kerberos [7]). Trust tickets are used to certify that two parties have already successfully negotiated a resource and, thus, subsequent negotiations for the same resource can be simplified. By contrast, the ticket mechanism exploited by authentication services is not used to grant accesses to a resource. Rather, it is only used to denote that a client has been authenticated by the authentication server.

4. With the term trust, we refer to the fact that the sequence is composed by certificates that can be disclosed according to the specified policies. Trust is not related, in this context, with certificate validity or their effective content.

5. Note that to assure the consistency of the ticket, its expiration must happen before the first certificate of the list expires.

4.3 Disclosure Policies

Trust-$\mathcal{X}$ disclosure policies state the conditions under which a resource can be released during a negotiation. Conditions are expressed as constraints on the certificates/trust tickets possessed by the parties involved in the negotiation and on their attributes. Additionally, protection of policies is obtained by introducing the notion of *prerequisites* associated with a policy, i.e., a set of alternative disclosure policies that must be disclosed before the disclosure of the policy they refer to. Prerequisite policies are also useful to obtain a fine-grained protection of the involved certificates. Each party adopts its own Trust-$\mathcal{X}$ disclosure policies to regulate release of local information (that is, credentials or policies) and access to services. Similar to certificates, disclosure policies are encoded using XML. Additionally, Trust-$\mathcal{X}$ policies can also be formalized as logical rules. In the following, we present this logical representation, since it makes explaining the compliance checker mechanisms and runtime system algorithms easier. XML encoding is reported in Appendix A (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm).

4.3.1 Preliminary Definitions

Disclosure policies regulate accesses to resources. A Trust-$\mathcal{X}$ resource can be either a certificate or a service. By service, we mean either an application that the requesting party can execute, for instance, for purchasing goods, or an access to protected data, such as, for instance, medical data. Additionally, a resource can be characterized by a set of attributes, specifying relevant characteristics of the resource that can be used when specifying disclosure policies. Trust-$\mathcal{X}$ resources are formalized by the notion of *R-Term*, introduced in what follows.

Definition 4.1: (R-Term). *An R-Term is an expression of the form Resource_name(attribute_list) where: $Resource\_Name$ is the resource name and can denote either a service or a certificate type, whereas $attribute\_list$ is a possible empty set of attribute names characterizing the resource.*

A resource can thus be considered as a structured object identified by a name and by some attributes. If the resource is a certificate type, the list of attributes consists of the attribute and tag names contained in its XML encoding. Resource attributes are used to express constraints on the resource release when specifying disclosure policies. We use the dot notation to refer to a specific attribute of a resource, that is, we use $R.a$ to denote attribute a of a resource R. Expressions of the form $R.a$ are called *resource expressions*.

Example 1. Examples of R-Terms are:

1. *Rental_Car(requesterCode, name, car_category, pick-upDate, ReturnDate)*: it denotes an online vehicle-rental service. The service is characterized by a set of attributes to customize the service release, such as the requester identity code, the name, the category of the requested car, and the duration of the rental (pick-up date and return date).
2. *Corrier_Employee()*: it denotes the certificate type Corrier_Employee.[6]

6. The list of attributes is not specified since it is implied by the certificate type it refers to.

4.3.2 Disclosure Policies

Disclosure policies are expressed in terms of logical expressions, called *policy conditions*, which can specify either simple or composite conditions against certificates/trust tickets.

Definition 4.2: (Policy condition). *Let* ***P*** *be either a* Trust-$\mathcal{X}$ *certificate type or a trust ticket. A policy condition C on* ***P*** *is an expression of the form* **a op expr**, *where:*

- ***a*** *denotes an element tag or an attribute name of* ***P***.
- ***op*** *is a comparison operator, such as* $\neq, <, >, =, \leq$.
- ***expr*** *can be either a constant or a resource expression, compatible with the type of* **a**.

Example 2. The following are examples of policy conditions:

1. *position = driver;*
2. *Release_year > 1998;*
3. *code = Rental_Car.requesterCode.*

Definition 4.3: (Term). *A term* $\mathcal{T}$ *is an expression of one of the following forms:*

- ***P(C)****: where* ***P*** *is either a* Trust-$\mathcal{X}$ *certificate type or a trust ticket, and* ***C*** *is a possibly empty list of policy conditions* $C_1 \ldots C_n$ *on* ***P***.
- ***X(C)****: where X is a variable and C is a nonempty list of policy conditions.*

The form X(C) denotes a list of conditions that may be contained in a generic certificate/trust ticket, without imposing any condition on the entity that contains them. Terms of the form X(C) can be used to express constraints on the counterpart properties without specifying where such properties should be collected. This option makes policy writers able to specify policies without having to be aware of all the certificate types/trust tickets that contain the required property. It also gives the receiver of the policy the flexibility of choosing which certificate to send as a reply. By contrast, the form P(), where the list of policy conditions is empty, denotes a term where the only constraint is given by the name of the certificate type/trust ticket and no conditions are specified. Finally, the form P(C) specifies both the certificate type/trust ticket and conditions against it. In the remainder of the paper, we say that a certificate/trust ticket *CT satisfies* a term *P(C)*, if *CT* is of type *P* and satisfies all the conditions specified in *C*. By contrast, we say that *CT satisfies X(C)*, if all the conditions specified in *C* are satisfied by *CT*. Additionally, given a term $\mathcal{T}$, we use the notation $P(\mathcal{T})$ to denote the certificate type/trust ticket (if any) in $\mathcal{T}$, and $C(\mathcal{T})$ to denote the policy conditions, if any, in $\mathcal{T}$.

Example 3. Consider the following terms:

$$\mathcal{T}_1 = IDCard(City = Milan, Country = Italy),$$
$$\mathcal{T}_2 = X(City = Milan, Country = Italy).$$

$\mathcal{T}_1$ denotes an ID card issued in Italy in the city of Milan. $\mathcal{T}_2$ denotes an unspecified certificate type, collecting the same information of $\mathcal{T}_1$. Note that $\mathcal{T}_1$ represents a possible binding for $\mathcal{T}_2$.

We are now ready to formally define disclosure policies. In the definition and throughout the paper, we use the dot notation to denote the component elements of a tuple. Thus, given a tuple t, we use $t.c$ to denote the value of the component c of tuple t.

Definition 4.4: (Disclosure policy). *Let* R *be a resource, a disclosure policy* ***p*** *for* R *is a pair (****pol_prec_set****,* ***rule****), where:*

1. ***rule*** *is an expression of one of the following forms:*
   - $\mathbf{R} \leftarrow \mathcal{T}_1, \mathcal{T}_2, , \mathcal{T}_n$, $n \geq 1$, *where* $\mathcal{T}_1, \mathcal{T}_2, , \mathcal{T}_n$, *are terms and R is the* $Resource\_name$ *component of an R-Term;*
   - $\mathbf{R} \leftarrow \mathbf{DELIV}$, *where* ***R*** *is the* $Resource\_name$ *component of an R-Term. When a policy contains this kind of rule, it is called* $delivery\ policy$.
2. ***pol_prec_set*** *is a set of policy identifiers,*[7] *named policy precondition set, where* $\forall p \in pol\_prec\_set, p.rule$ *is of the form* $\mathbf{R} \leftarrow \mathcal{T}_1, \mathcal{T}_2, ., \mathcal{T}_n$.

A disclosure policy is thus specified by a possibly empty set of policy preconditions and a rule. The rule specifies which properties a party should possess either for obtaining access to a resource managed by the other party or for letting the other party disclose the subsequent policy for the same resource. A delivery policy is a policy stating that no further information is requested for disclosing the requested resource. Additionally, since a policy may contain sensitive information, we introduce the concept of *policy preconditions*, that is, a set of policies such that at least one of the policy needs to be satisfied before the disclosure of the policy with which the precondition set is associated. Policies belonging to a precondition set are related to the same resource R of the policy with which they are associated. A further relevant use of precondition policies is that protection needs for a resource R can be organized in several policies logically linked (one policy is a precondition for the following and so on) and gradually disclosed during the policy evaluation phase of a negotiation. This can be useful in those contexts where the negotiation requires disclosing sensitive certificates. For instance, during an electronic transaction, credit card data are usually requested at the end of the negotiation when all the other information have been submitted and successfully verified. Moreover, since the precondition set of a policy may contain more than one policy, a party can specify a set of alternative possibilities for the disclosure of the corresponding policy/certificate. For instance, suppose that an entity requires three different certificates for a resource R, namely, $Cert_1$, $Cert_2$, and $Cert_3$ and suppose that $Cert_3$ must be disclosed only after $Cert_1$ and $Cert_2$ have been disclosed. Suppose moreover that information conveyed by $Cert_1$ and $Cert_2$ can be equivalently obtained by disclosing certificate $Cert_4$. The entity can organize these requirements through three different policies pol_1, pol_2, pol_3, where pol_1 asks for $Cert_1$ and $Cert_2$, pol_2 asks for $Cert_4$, and pol_3 asks for the most sensitive certificate, that is, $Cert_3$. $Cert_3$ can be disclosed whether the counterpart has already granted either $Cert_1$ and $Cert_2$ or $Cert_4$. According with Definition 4.4, this requirement can be expressed by

7. We assume that each policy is identified by a unique identifier.

including both pol_1 and pol_2 in the policy precondition set of pol_3. If no preconditions are needed for a rule, (e.g., pol_1 and pol_2), the related prerequisite set is empty and the rule can be disclosed without any preliminary check.

Example 4. Consider the Rental Car service scenario introduced in Section 3. As mentioned, the service is free for the employees of `Corrier` company, which belongs to the same holding of the Rental Car Company. Moreover, the Company already knows `Corrier` employees and has a digital copy of their driving licences. Thus, it only asks the employees the company badge and a valid copy of the ID card, to double check the ownership of the badge. By contrast, rental service is available on payment for unknown requesters, who have to submit first a digital copy of their driving licence and then a valid credit card. These requirements can be formalized as follows:

$$\begin{aligned}
pol_1 = (\{\}, &Rental\_Car \leftarrow Corrier\_Employee\\
&(code = Rental\_Car.requesterCode,\\
&position = driver), Id\_Card\\
&(name = Corrier\_Employee.name));\\
pol_2 = (\{\}, &Rental\_Car \leftarrow Driving\_Licence\\
&(name = Rental\_Car.name, issuer = EU));\\
pol_3 = (\{pol_2\}, &Rental\_Car \leftarrow Credit\_Card\\
&(name = Rental\_Car.nameRental\_Car.ReturnDate\\
&< ExpirationDate));\\
pol_4 = (\{pol_3, &pol_1\}, Rental\_Car \leftarrow DELIV).
\end{aligned}$$

Policy pol_2 requires the driving licence of the requester and is a precondition to proceed on the rental process. Intuitively, there is no reason to ask for a credit card if the requester cannot drive a car. Thus, pol_3 can be disclosed whether policy pol_2 specified in its precondition set is satisfied. The resource is thus deliverable (pol_4) when either policy pol_3 or pol_1 are satisfied.

To avoid loop among policies possibly introduced by policy prerequisites, the set of policies for a resource R must be *well-formed*. Well-formed chain of policies are formally defined as follows:

Definition 4.5: (Well-formed chain). *Let R be a resource, $\{p_1, \ldots, p_k\}$ be a set of disclosure policies for R. $[p_1, \ldots, p_k]$ is a well-formed chain of disclosure policies for R if the following conditions hold:*

- $p_1.pol\_prec\_set = \emptyset$;
- $\forall\, i \in [1, k-1], p_i \in p_{i+1}.pol\_prec\_set$;
- $p_k.rule = R \leftarrow \text{DELIV}$.

Example 5. Suppose that NBG Bank offers special Student loan for promising students. To check the eligibility of the requester, the Bank asks the student to present the student card, ID card, Social Security card, and current bank statements. Current bank statements can be presented by disclosing either the Federal Income Tax Returns or the requester statement account. Since some of the listed certificates contain high sensitive information about the student and his/her credit standing, the certificate requests can be organized in different policies to be gradually disclosed.[8]

$$\begin{aligned}
&p1 = (\{\}, Student\_Loan \leftarrow Student\_Card());\\
&p2 = (\{p1\}, Student\_Loan \leftarrow Social\_Security\_Card());\\
&p3 = (\{p2\}, Student_Loan \leftarrow\\
&\qquad Federal\_Income\_Tax\_Returns());\\
&p4 = (\{p2\}, Student\_Loan \leftarrow Bank\_Statement\_Account());\\
&p5 = (\{p3, p4\}, Student\_Loan \leftarrow DELIV);\\
&p6 = (\{p1, p4\}, Student\_Loan \leftarrow\\
&\qquad Bank\_Statement\_Account());
\end{aligned}$$

Policies $\{p1, p2, p3, p4, p5\}$ result in two distinct well-formed chains of policies, namely, $[p1, p2, p3, p5]$ and $[p1, p2, p4, p5]$. By contrast, p_6 creates a loop and makes the specification incorrect.

Each time a policy for a resource R is specified, the system checks whether it generates a chain of policies for R that is not well-formed and, in this case, it asks the revision of the policy. In the following, given a resource R and its set of policies, we use the term *policy chain* to denote a well-formed chain of policy disclosures for R. In the remainder of the paper, we say that the preconditions of a policy are *satisfied* if at least one of the policies in the precondition set has been previously disclosed and satisfied by the counterpart. Additionally, we say that the rule component of a policy is *satisfied* if the right side elements of the rule are all satisfied by the counterpart $\mathcal{X}$-Profile.

4.3.3 Introductory Policies

Introductory policies are used in the introductory phase of a negotiation and express general conditions that need to be satisfied in order to enter the negotiation process. Such policies are modeled as the rule component of disclosure policies, where the left side part of the policy represents a resource and right side elements model certificates/trust tickets. To distinguish introductory policies from ordinary ones, we simply add a subscription on the left side part of the policy, valueing p. The aim of introductory policies is twofold. On the one hand, they are used for establishing a first level of trust between negotiation participants and for speeding up the subsequent phases of the negotiation. One of the most interesting use of introductory policies is thus to check whether parties already know each other, and this can be done by asking for trust tickets. On the other hand, introductory policies are used to better customize the requested service (by asking for declarations). The following example clarifies the concept.

Example 6. Consider again the Rental Car Service scenario. Possible prerequisites that the agency may require before starting negotiations are modeled by the following introductory policies:

1. $Car\_Rental_p \leftarrow Trust\_ticket(ValidRes = Rental)$;
2. $Car\_Rental_p \leftarrow Car\_Preferences()$.

8. In describing the policies, we omit for simplicity the condition specification.

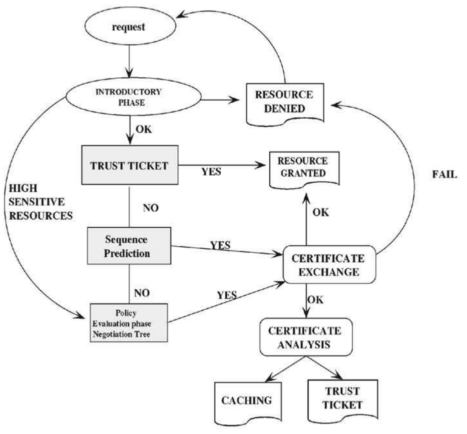


Fig. 4. Phases of a Trust-$\mathcal{X}$ negotiation.

The first policy checks whether parties have a trust ticket that can speed up the negotiation process, whereas, with the second policy, the Agency asks the requester to submit the `car_preferences` declaration, if any, in order to satisfy requester preferences.

5 The Negotiation Process

We believe that, since trust negotiations can be executed for several types of resources and by a huge variety of entities having various security requirements and needs, an effective negotiation system must provide several negotiation strategies. In order to address this issue, Trust-$\mathcal{X}$ includes different approaches to carry on trust negotiations, which differ with respect to efficiency and protection. By protected negotiation, we mean a negotiation executed by maximizing protection for all the resources involved. However, there can be cases where efficiency is the most crucial need or where the requested resource does not have very strict security requirements. In such cases, it is preferable to adopt strategies that speed up the negotiation, even if they do not maximize the protection of the involved resources. To better trade off among all the possible requirements of a negotiation, Trust-$\mathcal{X}$ thus supports a variety of strategies to carry on a negotiation that can be useful for different contexts and domains.

A Trust-$\mathcal{X}$ negotiation is organized according to the following phases: introductory phase, the sequence generation phase, certificate exchange phase, and caching of trust sequences. As illustrated in Fig. 4, the introductory phase is always executed, whereas the execution of the other phases depend on parties needs and on the evolution of the process. The sequence generation phase is the core of the negotiation process and can be carried out according to three different strategies: by performing the policy evaluation phase, by exchanging trust tickets, and by using the sequence prediction module. The first method is the most complete one and is the one that assures the maximum degree of protection to the involved resources. The use of trust tickets supports fast interaction for those entities who have previously completed a similar trust establishment process. These first two ways are strictly related since the use of trust tickets depends on previously executed negotiations between the same parties for the same (or a related) resource. Finally, the sequence prediction module is used when the negotiation that is being carried on shows similarities with previously executed negotiations. Similarity is estimated on the basis of information collected during previous negotiations and/or during the introductory phase. Practically, this way of carrying on the sequence generator phase consists of suggesting to the counterpart potentially valid trust sequences, in order to find the one that better matches the counterpart profile and security needs. The concluding step of a Trust-$\mathcal{X}$ successful negotiation is an analysis of the certificates and information exchanged and results either in caching the generated sequence, or in issuing trust tickets, or none of them. Indeed, each entity can cache the sequence of certificates that lead to negotiation success, to eventually reuse them in the following negotiations through the mentioned sequence prediction module. In what follows, we illustrate each of the above-mentioned phases in more detail. Note that articulating the negotiation into different distinct phases results in a multilevel protection that avoids release of unnecessary or unwanted information. Each phase is executed only if the previous ones succeed and sensitivity of the exchanged information increases during negotiation. Indeed, the disclosure of certificates, that are more sensitive with respect to policies, is postponed at the end of the whole negotiation; only if there is any possibility to succeed in the negotiation, certificates are effectively disclosed.

5.1 Introductory Phase

The introductory phase begins as a requester contacts a controller asking for a resource $\mathcal{R}$. This initial phase is carried out to exchange preliminary information that must be satisfied in order to start processing resource requests. Such exchange of information is regulated by the introductory policies of both the parties. Introductory policies are used to verify properties of the counterpart that are mandatory to continue in the negotiation. For instance, a server providing services only to registered clients, before evaluating the requirements for the requested service, can first ask the counterpart for the login name. If the requester is not registered, there is no reason to further proceed. Introductory policies are therefore essentially used to establish whether to enter into the core of the negotiation process or not, but they also can help in driving the subsequent phases of the process. Indeed, another important goal of the introductory phase is to check whether negotiation participants are not total strangers. Parties can prove that they have already had a previous successful negotiation establishing mutual trust through trust tickets. The key point is that each party should be able to determine who the counterpart is and what type of interactions they had in the past simply by trust ticket exchanges. Once the tickets have been exchanged and checked, it is easy to enter into the core of the negotiation process adopting a reduced version of the policy evaluation phase. Finally, introductory policies may also be used to collect information about the requester preferences and/or needs. For instance, in the Rental Car Service example, the agency may ask the requester to submit the `car_preferences` declaration, if any, in order to satisfy customer preferences.

5.2 Sequence Generation Phase

The ultimate goal of a negotiation is to establish mutual trust between the involved parties. Trust can be approached in different ways, on the basis of parties security needs and application domains. Trust-$\mathcal{X}$ supports three different ways to carry on the process, which are illustrated in what follows:

5.2.1 Policy Evaluation

During this phase, both the requester and the controller communicate disclosure policies adopted for the involved resources. The goal is to determine a sequence of requester and controller certificates that, when disclosed, allows the release of the requested resource, in accordance with the disclosure policies of both parties. This phase is carried out as an interplay between the requester and the controller. During each interaction, one of the two parties sends a set of disclosure policies to the other. The receiver party verifies whether its $\mathcal{X}$-Profile satisfies the conditions stated by the policies, and whether its local Policy Base contains policies regulating the disclosure of the certificates requested by the policies sent by the other party. If the $\mathcal{X}$-Policy of the receiver party satisfies the conditions stated by at least one of the received policies, the receiver can adopt one of two alternative strategies. It can choose to maximize the protection of its local resources replying only for one policy at a time, hiding the real availability of the other requested resources or, alternatively, it can reply for all the policies to maximize the number of potential solutions for negotiation. Additionally, when selecting a policy each party determines whether its preconditions are verified by the policies disclosed until that point and, only in this case, is the policy selected. By contrast, if the $\mathcal{X}$-Profile of the receiver party does not satisfy the conditions stated by the received policies, the receiver informs the other party that it does not possess the requested certificates. The counterpart then sends an alternative policy, if any, or it halts the process, if no other policies can be found. The interplay goes on until one or more potential solutions are determined, that is, whenever both parties determine one or more set of policies that can be satisfied for all the involved resources. The policy evaluation phase is mostly executed by the *Compliance Checker* whose goal is the evaluation of remote policies with respect to local policies and certificates (certificates can be locally available in the $\mathcal{X}$-Profile or can be retrieved through certificate chains), and the selection of the strategy for carrying out the remainder of the negotiation. To simplify the process, a tree structure is used (details are explained in Section 6.1) which is managed and updated by the *Tree Manager*. Note that no certificates are disclosed during the policy evaluation phase. The satisfaction of the policies is only checked to communicate to the other party the possibility of going on with the process and how this can be done.

5.2.2 Use of Trust Tickets

If during the introductory phase parties have determined that a trust ticket can be used for negotiation, the policy evaluation phase might be skipped. Each participant only has to analyze the validity and content of the received ticket, eventually retrieving the remote certificates specified into the `TrustSequence` element of the ticket again. This further check can be executed to ensure that the certificates that previously granted negotiation success are actually still valid. If all these checks end successfully, the negotiated resource is immediately granted to the requester and the process successfully ends.

5.2.3 Sequence Prediction Module

It is really likely that the same trust sequence will be used several times to perform similar negotiations. With the notion of similarity, we mean negotiations for the same resource that are executed by an entity with different counterparts having a similar profile. In such cases, it might be useful to keep track of the sequences of certificates more often exchanged, instead of recalculating them for each negotiation. For instance, some books of a digital library might be often asked during exam sessions. The procedure to establish trust will be very similar for different students. Or, better, the required information will be exactly the same and the only differences will concern the type of certificates disclosed. A student attending University X might have a card issued by the main secretary office of the university, whereas students of a branch department might have only a badge issued by the departmental secretary office. Suppose that the properties required to obtain access can be proved by presenting either the card issued by the main secretary office or by presenting both the badge issued by the departmental secretary office and the student library badge. Moreover, suppose that students usually require privacy guarantees before disclosing certificates and that the library proves its honesty by a set of proper certificates. Different sequences can therefore be generated to disclose the same resource, but all of them are quite intuitive and easy to determine. The controller can cache and suggest them upon receiving a request from a student. The student can cache the most widely used sequences for negotiating access to digital libraries as well as the library and suggest them upon sending a request to a library.

Intuitively, suggesting a sequence of certificates to solve the negotiation before knowing whether the counterpart is really trusted does not ensure a complete protection of policies. However, in many contexts, protection of policies or information is not really necessary. Indeed, we expect that in many scenarios, there will be standard, off-the-shelf policies available for common, widely used resources (e.g., VISA cards, passports). In case of negotiations involving these common resources, the sequences of certificates to be disclosed will be only regulated by such standard and predictable policies. In this case, certificates represent only a means to easily establish trust, and it is not unsafe to suggest the sequences at the beginning of the process. If the counterpart cannot satisfy the proposed sequence, the negotiation can continue by executing the policy evaluation phase or by suggesting another sequence.

The module of Trust-$\mathcal{X}$ in charge of caching and suggesting the sequences is the *sequence prediction module*. It consists of two main components: the cache memory and the prediction algorithm, which verifies whether this strategy can be applied to a specific negotiation. The cache memory collects all trust sequences recently executed that can be directly reused. To better manage sequence reuse and storage, each cached sequence is complemented by a set of additional information to be used to determine the sequence usage.

Definition 5.1: (Cached sequence). *A cached sequence cts is a tuple **(ts, R, rel_cond, count, date)**, where **ts** is a trust sequence, i.e., a sequence of certificate identifiers that make it possible the successful end of a negotiation; **R** is the resource for which **ts** was originally generated; **rel_cond** is a set of information for **ts** usage, i.e., conditions and/or certificates that characterize **ts**; **count** counts the number of times **ts** has been successfully used; **date** stores the last date **ts** was successfully used.*

A cached sequence is thus a structured object containing the list of certificate types that grant a successfully disclosure of R, the resource originally requested. Set **rel_cond** is a set of certificate names and conditions characterizing sequence **ts**. Its content usually strongly depends on the policies associated with the certificates in **ts**. The idea is that, when a cached sequence is selected by a party, information in rel_cond are used to verify whether the counterpart properties match the remote certificates associated with sequence. For instance, a trust sequence can be usable only if the other party has an age greater than a given threshold. In such a case, rel_cond will contain such constraint which has to be verified before the sequence can be successfully selected. Parameter $count$ is a counter keeping track of the number of times the sequence is used to solve the negotiation. Finally, $date$ memorizes the last date ts was successfully used. Such parameters can be used to extract the better sequence that may lead to negotiation success.

Example 7. With respect to the scenario introduced in Example 5, consider a negotiation between Bank NBG and a student applying for a student loan. Suppose that the negotiation successfully ends. The corresponding trust sequence will contain certificates that prove the honesty and the credit standing of the requester as well as the reliability of the bank. A possible sequence cached by the Bank will thus contain the corresponding certificate names, such as student identity card, student residence certificate, student work certificates, Federal income tax returns, debt obligation information (if applicable), and also Bank obligations certificates, and Bank Guarantees. rel_cond will store relevant conditions that need to be verified before suggesting the sequence, such as, for instance, if the age of the student is less than 21, if the yearly income is less than a fixed amount, and if there are any guarantees.

Intuitively, controllers of resources will often incur in similar negotiations and, therefore, they will likely have available a large number of cached sequences to handle resource requests. However, since Trust-$\mathcal{X}$ entities are considered as peers, requesters may use all of the same sequence prediction module to reuse previously used sequences as much as possible.

In what follows, we consider the controller behavior. During a negotiation, upon the end of introductory phase, the party verifies whether it has cached any sequence for the requested resource R. If one or more sequences for R are in the cache, the controller first extracts the most commonly used sequences, which can be easily determined using the parameter $count$. Then, if the information collected during the introductory phase are not inconsistent with respect to the information contained into set rel_cond, the controller asks to the counterpart for the remaining information. Note that the aim of this phase is not to immediately obtain the certificates, but simply to query the counterpart to verify the applicability of a sequence. Upon counterpart reply, the party extracts the sequences that better match the requester profile and the negotiation can proceed with the subsequent phase.

5.3 Certificate Exchange

This phase begins when the sequence generator phase determines one or more trust sequences either by successfully executing the policy evaluation process or by using the sequence prediction module. Several trust sequences can be determined to succeed in the same negotiation and several criteria can then be used by the parties to select one of the possible trust sequences. Examples of these criteria include the number of involved certificates, the sensitivity of their content, or the expected length of negotiation. Once the parties have agreed on a sequence, the certificate exchange begins. Each party discloses its certificates, following the order defined in the trust sequence, eventually retrieving those certificates that are not immediately available through certificate chains. To avoid man in the middle [7] attacks, each certificate is wrapped into a specific package, named $\mathcal{X}$-Package. The package consists of a Trust-$\mathcal{X}$ certificate, the random number received from the counterpart, and the signature computed on both the certificate and the number, using the party private key. When an $\mathcal{X}$-Package is transmitted, the recipient verifies the signature using the sender's public key. If the signature is valid, the $\mathcal{X}$-Package is accepted. The idea is to embed the Trust-$\mathcal{X}$ certificate in a signed package so that, if an intruder intercepts the packet, it will not be able to reuse the certificates pretending to be the real owner. Furthermore, this structure allows parties to encrypt the whole package to achieve confidentiality of Trust-$\mathcal{X}$ certificates. Upon receiving a certificate, the counterpart verifies the satisfaction of the associated policies, checks for revocation, checks validity dates, and authenticates the ownership (for credentials). Eventually, if further information is needed for establishing trust, it is the receiver responsibility to check for new certificates using credential chains. The receiver then replies with an acknowledgment, and asks for the subsequent certificate in the sequence, if any. Otherwise, a certificate belonging to the subsequent set of certificates in the trust sequence is sent. The process ends with the disclosure of the requested resource or, if any unforeseen event happens, an interruption. If the failure is related to trust, for instance, a party is using a revoked certificate, the negotiation fails. Otherwise, if it is due to events that are independent from trust, for instance, interruption of connection, the negotiation is restarted, executing the same sequence or, if it is not possible, one of the alternative trust sequences determined at the beginning of this phase.

In the following section, we focus on the key and most complex phase of a Trust-$\mathcal{X}$ negotiation, that is, the policy evaluation phase.

6 Policy Evaluation Phase

The policy evaluation phase consists of a bilateral and ordered policy exchange. As previously introduced, this method is the most complete one and is the one that assures the maximum degree of protection of all the involved resources. Certificates and services are disclosed only after a complete counterpart policies evaluation, that is, only when the parties have found a trust sequence of certificate disclosures that makes it possible the release of the requested resource, according to the disclosure policies of both parties. Even disclosure of sensitive policies is protected since policies are disclosed gradually according to the degree of trust established. The policy evaluation phase is executed either when the parties are total strangers and it is not possible to reuse the result of previous negotiations, or when the negotiated resources require high protection. The Compliance Checker module of each party, upon receiving a disclosure policy, determines if it can be satisfied by any certificate of the local $\mathcal{X}$-Profile. Then, it checks in its Policy Base the protection needs associated with the certificates satisfying the policy, if any. The progress of a negotiation is recorded into a specific data structure, called $Negotiation\ tree$, managed by the $Tree\ Manager$, illustrated in the following section.

6.1 Negotiation Tree

A negotiation tree is a data structure used to maintain the progress of a negotiation. The goal of the tree is to identify at least one *view*, where a view denotes a possible trust sequence that can lead to the negotiation success. The view keeps track of which certificates may contribute to the success of the negotiation, and of the correct order of certificate exchange.

More formally, a negotiation tree is a labeled tree where each node corresponds to a term, whereas edges correspond to policy rules. More precisely, a negotiation tree is characterized by two different kinds of edges: *simple edges* and *multiedges*. A simple edge denotes a policy having only one term on the left side component of the rule. By contrast, a multiedge links several simple edges to represent policy rules having more than one term on their left side component. Nodes belonging to a multiedge are thus considered as a whole during the negotiation.

The tree is dynamically built during the policy evaluation phase and grows up as the phase proceeds. Upon the end of the introductory phase, each party triggers an instance of a negotiation tree, rooted at the requested resource R. The tree is then consequently updated each time a set of policies are exchanged and evaluated. The operations executed on the tree consist of adding new edges (to keep track of both the policies and the resources involved) and updating the state of a node when a corresponding set of policies is satisfied. Each level of the resulting tree is always composed by nodes belonging alternatively to the requester or the controller.

Before formally defining a negotiation tree, we need to introduce a function called $Eval$ which is used by the $Compliance\ Checker$. It receives as input a term and an $\mathcal{X}$-Profile and determines whether the $\mathcal{X}$-Profile satisfies the conditions stated by the term.

Definition 6.1: (Eval). *Function*

$$Eval(\mathcal{T}, Prof) \rightarrow \{true, false\}.$$

*Let $\mathcal{T}$ be a term and Prof an $\mathcal{X}$-*Profile*:*

$$Eval(\mathcal{T}, Prof) = \begin{cases} TRUE & \text{if } (\mathcal{T}\text{=P()} \wedge P \text{ belongs to } Prof) \vee (\mathcal{T}\text{=}P(C) \\ & \wedge\, P \text{ belongs to } Prof \wedge \text{ satisfies the} \\ & \text{conditions in } C \vee (\mathcal{T}\text{=}X(C) \wedge \exists\, P \text{ in } Prof \\ & \text{such that P satisfies the conditions in } C) \\ FALSE & \text{otherwise.} \end{cases}$$

Additionally, since a negotiation tree is a labeled tree where the labels give information on the order in which the nodes should be considered, we need to introduce some notations that we use in the edge labeling function. Let $Terms_{rq}$ and $Terms_{cn}$ be the set of terms appearing in the policies of $\mathcal{PB}_{rq}$ and $\mathcal{PB}_{cn}$, respectively; and let $\sum(Terms^*_{rq})$ and $\sum(Terms^*_{cn})$ be the set of strings build on $Terms_{cn}$ and $Terms_{rq}$. Let "+" be a special symbol denoting the concatenation operator of strings. We define set label as follows:

$$Label = \{t_1 + \ldots + t_{k-1} + t_k | k \geq 1[t_i \in \sum(Terms^*_{cn}) \vee t_i \in \sum(Terms^*_{rq})]\}$$

and $Label^*$ as the set of possible edge labels obtained by concatenating strings in $Label$ with the ending symbol "l."

We are now ready to formally define a negotiation tree. The most widely used symbols are reported in Fig. 5a.

Definition 6.2: (Negotiation tree). *Let $\mathcal{CN}$ be a controller and $\mathcal{RQ}$ be a requester. Let $\mathcal{PB}_{cn}$ and $\mathcal{PB}_{rq}$ be the policy bases associated with $\mathcal{CN}$ and $\mathcal{RQ}$, respectively. Let X-Prof$_{cn}$ and X-Prof$_{rq}$ be the $\mathcal{X}$-*Profiles* associated with $\mathcal{CN}$ and $\mathcal{RQ}$, respectively. Let R be the resource requested by $\mathcal{RQ}$ to $\mathcal{CN}$. A Negotiation Tree $NT = \langle \mathcal{N}, \mathcal{R}, \mathcal{E}, \phi \rangle$ for R, $\mathcal{CN}$, and $\mathcal{RQ}$ is a finite tree satisfying the following properties:*

- *$\mathcal{N}$ (the set of nodes) is a set of triples*

  $$n =< \mathcal{T}, state, party >,$$

  where:

  - *$\mathcal{T}$ is a term;*
  - *state denotes the current state of the node;*
  - *party $\in$ {$\mathcal{RQ}$, $\mathcal{CN}$} denotes if the node belongs to $\mathcal{RQ}$ or $\mathcal{CN}$;*
- *$\mathcal{R} =< R, state, \mathcal{CN} >$ is the root of the tree;*
- *$\mathcal{E}$ (the set of edges), where each $e \in \mathcal{E}$ has one of the following forms:*

  - ***simple edge** ($\mathcal{SE}$): $e = (n_1, n_2), n_1, n_2 \,\epsilon\, \mathcal{N}$ belongs to $\mathcal{SE}$ if both the following conditions hold:*

    ◆

    $$[\exists\, p \,\epsilon\, \mathcal{PB}_{rq/cn} | p.rule = \mathcal{T}(n_1) \leftarrow \mathcal{T}(n_2) \wedge Eval(\mathcal{T}(n_1), X\text{-}Prof_{rq/cn}) = TRUE]$$

    or

    $$[\mathcal{T}(n_1) = R \wedge \exists\, p \,\epsilon\, \mathcal{PB}_{cn} | p.rule = R \leftarrow \mathcal{T}(n_2)] :$$

    ◆

    $$[(\mathcal{T}(n_1) \epsilon X\text{-}Prof_{rq/cn}) \wedge (\mathcal{T}(n_2) \,\epsilon\, X\text{-}Prof_{cn/rq})].$$

  - ***multiedge** ($\mathcal{ME}$): $e = \{(n, n_1), \ldots, (n, n_k)\}$, $n, n_1 \ldots n_k \,\epsilon\, \mathcal{N}$, belongs to $\mathcal{ME}$ if both the following conditions hold:*

    ◆

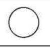
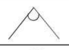
    $$[Eval(\mathcal{T}(n), X\text{-}Prof_{rq/cn}) = TRUE \wedge \exists\, p \,\epsilon\, \mathcal{PB}_{rq/cn} | p.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_1) \ldots \mathcal{T}(n_k)]$$

    or

    $$[\mathcal{T}(n) = R \wedge \exists\, p \,\epsilon\, \mathcal{PB}_{cn} | p.rule = R \leftarrow \mathcal{T}(n_1) \ldots \mathcal{T}(n_k)].$$

VARIABLE	DESCRIPTION
T	term
n	node
T(n)	term in a node
state(n)	state of the term in *n*
party(n)	owner of the term in *n*
e=(n, n₁)	simple edge
e={(n,n₁), ...,(n,nₖ)}	multi edge
p	policy
p.pol_prec_set	precondition set of p
p.rule	rule of a policy p
R<-T₁,T₂,..,Tₖ	rule

(a)

GRAPH NOTATION	MEANING
	open node
	simple edge
	multi edge
	deliv node
	linked nodes

(b)

Fig. 5. (a) Negotiation tree symbols and (b) negotiation tree graph notation.

◆

$$[(\mathcal{T}(n)\ \epsilon\ X\text{-}Prof_{rq/cn}) \wedge (\mathcal{T}(n_1)\dots\mathcal{T}(n_k)\ \epsilon\ X\text{-}Prof_{cn/rq})];$$

- $\phi : \mathcal{E} \rightarrow Label \cup Label^*$ *is the edge labeling function. Edge labels are generated by the algorithm presented in Fig. 8.*
- *The state of a node n can assume one of the following two values:*[9]

  - ***DELIV**, if $Eval(\mathcal{T}(n), X\text{-}Prof_{rq/cn}) = TRUE$ and one of the following conditions hold:*

    1. *n is nonleaf and $\exists\, \mathcal{E}' = \{e_1, \dots, e_k\} \subseteq \mathcal{E}, k \geq 1$ such that all the following conditions hold:*

       a. $\forall e \in \mathcal{E}', e = \{(n, n_1), \dots, (n, n_k)\}$, $\forall i \in [1, k]$, *$state(n_i) = DELIV$;*

       b. $\exists\ e = \{(n, n_1), \dots, (n, n_k)\} \in \mathcal{E}' | \phi(e) ='' $ *and* $\exists\ p \in \mathcal{PB}_{rq/cn}$ *such that*

       $$p.pol\_prec\_set = \emptyset \wedge p.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_1), \dots, \mathcal{T}(n_k).$$

       c. $\forall\ e \in \mathcal{E}', e = \{(n, n_1), \dots, (n, n_k)\}, \phi(e) \neq'' \exists p \in \mathcal{PB}_{cn/rq}, \exists p \in \mathcal{PB}_{cn/rq}$ *such that*

       $$p.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_1), \dots, \mathcal{T}(n_k), \wedge [\exists\ p' \in\ p.pol\_prec\_set | p'.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n'_1), \dots, \mathcal{T}(n'_k)] \wedge [\phi(e) =' .. + \mathcal{T}(n'_1) | \dots | \mathcal{T}(n'_k) + ..'].$$

       d. $\exists e = \{(n, n_1), \dots, (n, n_k)\} \in \mathcal{E}'$, $\phi(e) =' *l'$[10] $\wedge \exists p, p'' \in \mathcal{PB}_{rq/cn}$ *such that*

       $$p.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_1), \dots, \mathcal{T}(n_k) \wedge [p \in p''.pol\_prec\_set \wedge p''.rule = \mathcal{T}(n) \leftarrow DELIV].$$

    2. *n is a leaf node and*

       $$[\exists p\ \epsilon\ \mathcal{PB}_{cn/rq} | p.pol\_prec\_set = \emptyset \wedge p.rule = \mathcal{T}(n) \leftarrow DELIV];$$

  - ***OPEN**, if neither condition 1 nor condition 2 hold.*

9. We give the definition by referring to mutliedges. In this context, simple edges can be considered as a particular case of multiedges.
10. * is a metacharacter to denote a string of variable length.

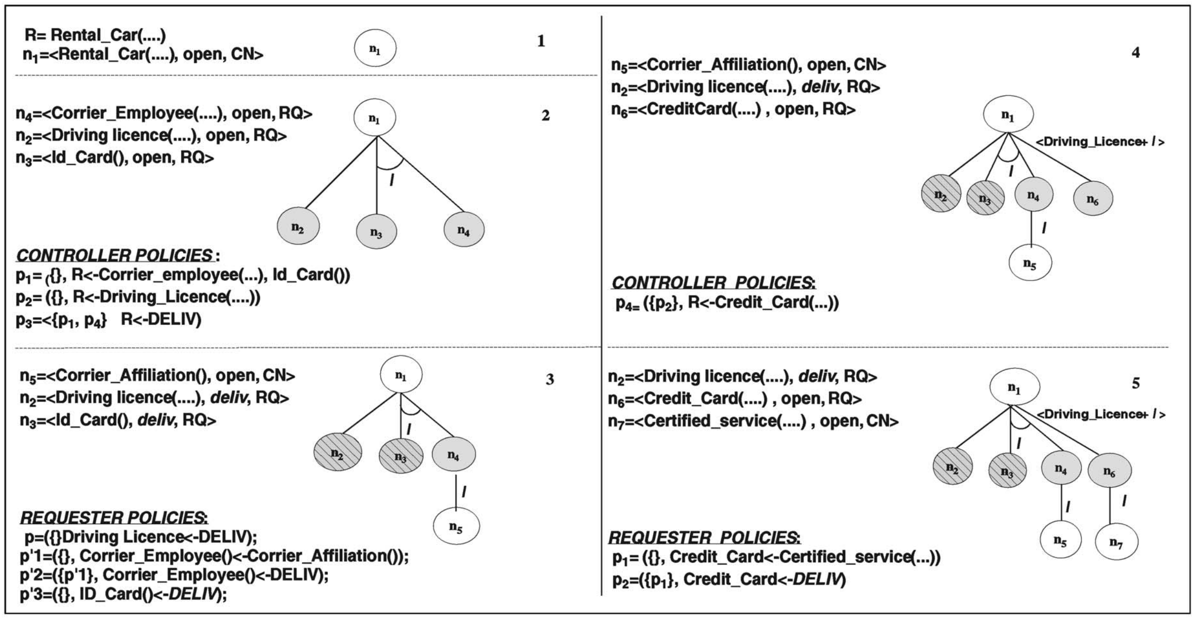
6.2 Negotiation Tree Building

A negotiation tree is thus a particular tree, which evolves during the policy evaluation phase by adding disclosure policies of one of the parties. Graphically, a negotiation tree is built according to the notation introduced in Fig. 5b.

Fig. 6 shows the initial steps of the generation of a negotiation tree. The example shows a negotiation between a requester and a controller and is based on the case study proposed in Section 3.[11] Multiedges are the result of policies having the left side elements of the rule component with more than one term. In Fig. 6, at step 2, the evaluation of policy $R \leftarrow Corrier\_Employee(\dots), Id\_Card(\dots)$ results in a multiedge connecting node n_1 with node n_3 and n_4. By contrast, a simple edge is generated by a rule having only one term on its left side. Examples of simple edges are the edge connecting n_1 and n_2, and the one, appended at the step 3, connecting n_4 to n_5.

A negotiation tree is built by repeatedly invoking function *Build* shown in Fig. 7. The function receives as input the negotiation tree constructed so far and a node n and verifies whether a new edge should be added originating at n. More precisely, function *Build* starts by querying the local $\mathcal{X}$-Profile in order to find whether there is a certificate that satisfies the term associated with the input node. Note that by $\mathcal{X}$-Profile we now refer to all the certificates related with the local party, that is, the certificates that are physically locally stored and those that are stored in remote sites but that can be retrieved by the party through certificates chains, if necessary. Upon the retrieving of the certificate, the local policy base is then queried, to extract the corresponding policies. The tree is then consequently updated, either by adding one or more edges, or by updating the state of the nodes, if the policy is a

11. For simplicity, we focus on the most relevant policies and related terms.



1
R= Rental_Car(....)
n1=<Rental_Car(....), open, CN>
2
n4=<Corrier_Employee(....), open, RQ>
n2=<Driving licence(....), open, RQ>
n3=<Id_Card(), open, RQ>
CONTROLLER POLICIES :
p1= ({}, R<-Corrier_employee(...), Id_Card())
p2= ({}, R<-Driving_Licence(....))
p3=<{p1, p4} R<-DELIV)
3
n5=<Corrier_Affiliation(), open, CN>
n2=<Driving licence(....), deliv, RQ>
n3=<Id_Card(), deliv, RQ>
REQUESTER POLICIES:
p=({}Driving Licence<-DELIV);
p'1=({}, Corrier_Employee()<-Corrier_Affiliation());
p'2=({p'1}, Corrier_Employee()<-DELIV);
p'3=({}, ID_Card()<-DELIV);
4
n5=<Corrier_Affiliation(), open, CN>
n2=<Driving licence(....), deliv, RQ>
n6=<CreditCard(....) , open, RQ>
<Driving_Licence+ l >
CONTROLLER POLICIES:
p4= ({p2}, R<-Credit_Card(...))
5
n2=<Driving licence(....), deliv, RQ>
n6=<Credit_Card(....) , open, RQ>
n7=<Certified_service(....) , open, CN>
<Driving_Licence+ l >
REQUESTER POLICIES:
p1= ({}, Credit_Card<-Certified_service(...))
p2=({p1}, Credit_Card<-DELIV)

Fig. 6. An example of negotiation tree building.

delivery policy. Each of the added edges is the result of a policy for the resource associated with n, and is appended only if its precondition set is verified. The verification of this last condition can be simply done by checking the state of the children of n. In particular, the precondition set is verified if there exists at least an edge corresponding to one of the policies in the precondition set whose entering nodes state is set to DELIV. With respect to step 4 of Fig. 6, the edge connecting n_1 with n_4 related to policy $p3$ is appended as the node n_2, whose edge corresponds to a policy of the precondition set, is set to DELIV. Preconditions associated with a policy imply an order in the disclosure of the corresponding certificates. Such order is kept into account in building the tree by using an ad hoc labeling system. Intuitively, the label associated with each edge in the tree gives information on the order in which the certificates associated with the corresponding nodes must be disclosed to successfully end the negotiation process.

The labeling function is presented in Fig. 8. Given an edge e, its label $\phi(e)$ is a possibly empty string of terms $\phi(e) =' t_0 + \ldots + t'_h$, where each substring t_i is obtained by concatenating the terms of a satisfied policy in the corresponding precondition set. Symbol "+" is used to group the set of terms associated with the same policy. As an example, consider the case of a multiedge $e = \{(n, n_1)(n, n_2)\}$ corresponding to a satisfied policy p, and a policy p_1 having p in its precondition set. If p_1 is not a delivery policy, p_1 will be modeled as an edge e_1 labeled with "$\mathcal{T}(n_1)|\mathcal{T}(n_2)$." Otherwise, e label will be updated with

```
Function Build($NT,\ n$)
$Input$ :
        $n$: the node to be processed;
        $NT = \langle \mathcal{N}, \mathcal{R}, \mathcal{E}, \phi \rangle$ a negotiation tree for $\mathcal{R}$;
$Output$ :
        $NT' = \langle \mathcal{N}', \mathcal{R}, \mathcal{E}', \phi \rangle$ such that $\mathcal{N} \subseteq \mathcal{N}'$ and $\mathcal{E} \subseteq \mathcal{E}'$ or a failure message;
$Precondition$:
        $State(n) \neq DELIV$

begin
   $\mathcal{N}' := \mathcal{N}$;
   $\mathcal{E}' := \mathcal{E}$;
Let $\mathcal{X}\text{-}Prof_L$ and $\mathcal{PB}_{\mathcal{L}}$ be the local $X$-Profile and policy base, respectively
If Eval($\mathcal{T}(n)$, $\mathcal{X}\text{-}Prof_L$) = TRUE then
   Let $P = \{p_1, .., p_k\}$ be the subset of $\mathcal{PB}_l$ such that $\forall\ i\ \in\ [1, k]$ the
   head of $p_i.rule = \mathcal{T}(n)$;
   Let $E$ be the edges in $\mathcal{E}$ rooted at $n$;
      For each $p_j\ \in\ P\ |\ [p_j.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_g)..\ \wedge\ \nexists\ e \in E$
        $|\ e = \{(n, n_g)..\}] \vee\ [p_j.rule = \mathcal{T}(n) \leftarrow DELIV]$
        If $[\exists\ p_i\ \in\ p_j.pol\_prec\_set\ |\ p_i.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n'_1), .., \mathcal{T}(n'_k)\ \wedge$
        $\exists\ e = \{(n, n'_1)...(n, n'_k)\}\ \in\ E\ \wedge\ state(n'_1) = ... = state(n'_k) = DELIV]$
          or $[E = \emptyset \wedge p_j.pol\_prec\_set = \emptyset]$
            Switch($p_j.rule$)
              case: $p_j.rule$ is of the form $\mathcal{T}(n) \leftarrow DELIV$
                $state(n) = DELIV$;
                $UpdateState(NT, n)$; % Update the tree state
                Return($\emptyset$);
              case: $p_j.rule$ is of the form $\mathcal{T}(n) \leftarrow T_1, ..T_k$
                For each $T_1...T_k$  create a node  $n_1, ...n_k$;
                Add $\{n_1, .., n_k\}$ to $\mathcal{N}'$;
                Add $e_i = \{(n, n_1), (n, n_2), ..(n, n_k)\}$ to $\mathcal{E}'$;
                Labeling(e$_i$, p$_j$, E, P);
              case: $p_j.rule$ is of the form $\mathcal{T}(n) \leftarrow T_2$
                Create node $n_2$;
                Add $e_i = \{(n, n_2)\}$ to $\mathcal{E}'$;
                Labeling(e$_i$, p$_j$, E, P);
                Add $\{n_2\}$ to $\mathcal{N}'$;
      endfor
   Return($\langle \mathcal{N}', \mathcal{R}, \mathcal{E}', \phi \rangle$);
else
   Return($n$, FAILURE) % Failure message
end
```

Fig. 7. Function Build().

```
Function Labeling($e,\ p,\ E, P$)
$Input$ :
      $e$: the edge to be labeled;
      $E$: the set of sibling edges of $e = \{(n, n_1)..(n, n_k)\}$ ;
      $p$: the corresponding policy used for building $e$;
      $P$: the set of policies for $\mathcal{T}(n)$;
$Output$ :
      The label for edge $e$, namely $\phi(e)$
begin
Let $p.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_1), ..., \mathcal{T}(n_k)$
If $p.pol\_prec\_set = \emptyset$ then:    $\phi(e) ='' $    % The label is the empty string
else
   For each $p_i\ \in\ p.pol\_prec\_set$ such that $p_i.rule = \mathcal{T}(n) \leftarrow \mathcal{T}(n_{i1}), ..., \mathcal{T}(n_{ik})\ \wedge$
      $e_i = \{(n, n_{i1}), .., (n, n_{i1})\} \in E\ \wedge\ state(n_{i1}) = ..state(n_{ik}) = DELIV$ do
            $\phi(e) =' \phi(e) + \mathcal{T}(n_{i1})|...|\mathcal{T}(n_{ik})'$;
   endfor
Let $\overline{p}$ be the delivery policy for $\mathcal{T}(n)$ in $P$;
   If $\overline{p}.pol\_prec\_set \subseteq p$ then:    $\phi(e) =' \phi(e) + l'$;
RETURN($\phi(e)$);
end.
```

Fig. 8. Function Labeling().

```
Function UpdateState($NT, n$)
$Input$ :
    $NT = \langle \mathcal{N}, \mathcal{R}, \mathcal{E}, \phi \rangle$ is a negotiation tree for $\mathcal{R}$:
$Output$ :
    A special message signaling either that a view has been found in $NT$ or the executed updating
$Precondition$:
    $state(n)$=DELIV;
begin
  While $\mathcal{T}(n) \neq \mathcal{R}$    % Backtracking the update of the state along the tree
  Let $n_0$ be the parent node of $n$ and $\overline{e}$ the edge connecting $n_0$ and $n$;
    If $[\overline{e} = (n_0, n) \in \mathcal{SE}]$ or $[\overline{e} = \{(n_0, n), (n_0, n_1), .., (n_0, n_k)\} \in \mathcal{ME}$ and
      $state(n) = state(n_1) = .. = state(n_k) = DELIV]$ and $\phi(\overline{e}) =' *l'$ then
        $state(n_0) = DELIV$
        $n := n_0$          % continue the update
    else
      $RETURN(Deliv(NT, n))$;       % the state cannot be propagated
  endwhile
RETURN($FoundView(NT)$);
end.
```

Fig. 9. Function UpdateState().

the ending symbol "l" and no other edges will be added. As a further example, consider two edges $e_1 = (n, n'_1)$ and $e_2 = (n, n'_2)$ corresponding to two satisfied policies p_1 and p_2 for the resource in n. Suppose that p_1 and p_2 are in the precondition set of a third policy for the resource in n, say p_3. p_3 will be modeled as an edge labeled by the string "$\mathcal{T}(n'_1) + \mathcal{T}(n'_2)'$. With reference to Fig. 6, at step 4, upon the disclosure of the delivery policy for the Driving Licence, a new edge is appended, labeled with the corresponding satisfied term. Note that, since policy preconditions for a resource R are not disjoint, a single satisfied policy may allow the simultaneous disclosure of several subsequent policies (all the policies that have the satisfied policy in their precondition set). Furthermore, since an edge representing a policy for R is appended when at least one of its precondition policies is satisfied, the satisfaction of the remaining policies of the set may happen several steps after having added the edge. However, to avoid redundancy, a policy is mapped onto an edge only once and the label is no more updated, even if new policies of the preconditions become satisfied. The edge label then keeps track of the order followed in appending the policies, with respect to the constraints implied by the policy preconditions of the policy chain traversed. When the last policy for a resource is processed and the corresponding edge is then created, its label also contains an ending symbol, that is, l. In addition, if some policies give rise to a cycle the tree is pruned to remove the cycle, using the strategy illustrated in Section 6.4. The $state$ associated with a node denotes the possibility of finding a trust sequence of certificates containing the term associated with the node. When a new node n is appended to the tree its state is set to OPEN, meaning that the tree may evolve through addition of children to the given node. The state of a nonleaf node n is set to DELIV when exist a set of edges $\mathcal{E}'$ exiting from n, each corresponding to a satisfied policy of a well-formed chain of policies (cfr. Definition 4.5) as stated by subcondition 1.(a) of Definition 6.2. To denote a well-formed chain of policies the set $\mathcal{E}'$ must contain one edge having an empty label, denoting the first policy of the chain (condition 1.(b) of Definition 6.2) and one edge whose label contains the ending symbol denoting the last policy (condition 1.(d) of Definition 6.2) of the chain. The remaining edges in $\mathcal{E}'$ must be properly labeled to keep track of the order of the policies in the chain, as stated by subcondition 1.(c) of Definition 6.2.

By contrast, a leaf node is immediately set to DELIV if there is only one policy associated with the resource in the node that is a delivery policy. As an example, with reference to Fig. 6, nodes n_2 and n_3 are labeled as OPEN until the evaluation of the delivery policies: $Driving\_Licence \leftarrow DELIV$ (whose corresponding term is in node n_2) and $Id\_Card \leftarrow DELIV$ (whose corresponding term is in node n_3) is completed. Note that in the example, the edge connecting n_1 and n_3, n_4 is labeled by the ending symbol since the resources for which the corresponding policies are specified are not protected by any further condition. Each time function $Build$ updates the state of the input node into DELIV, a further function is invoked to determine if the evaluation phase can successfully end. More precisely, the function $UpdateState$ (presented in Fig. 9) verifies whether the DELIV state can be propagated up to the tree root and, if this is not the case, it propagates as much as possible the DELIV state.

If the state of the root becomes DELIV, the function outputs a special message, otherwise, the policy evaluation phase goes on by sending the $Deliv(NT, n)$ message. The $Deliv(NT, n)$ message just means that the input negotiation tree has been updated, but the resource R originally requested is not yet disclosable. Consider again the example of Fig. 6. Suppose that the server certification, named $Certified\_Service$ is not protected by any policies and, thus, the corresponding node is tagged DELIV, as shown in Fig. 10a. The DELIV state is then propagated until the root (cfr. Fig. 10b).

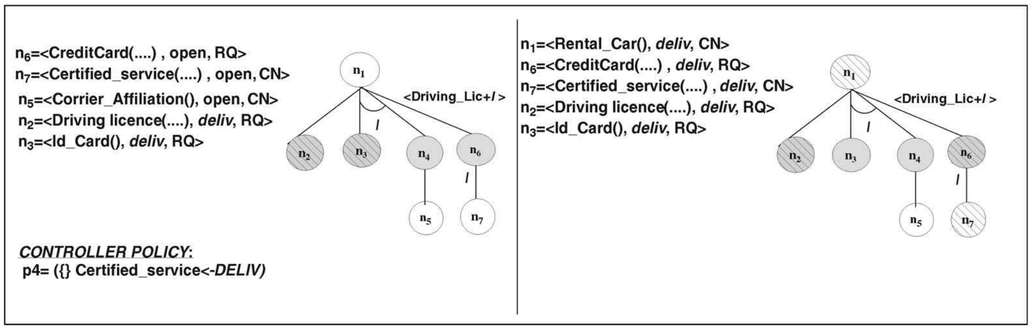
6.3 Valid Views and Trust Sequences

The successful end of the policy evaluation phase occurs when the DELIV state is propagated up to the root of a negotiation tree. When such a condition holds, it is possible to determine a trust sequence for the considered negotiation. The trust sequence can be built by traversing a portion of the negotiation tree according to a specified order. Such portion consists only of nodes with a DELIV state and is denoted in the following as *valid view*. The notion of a valid view is formally introduced by next definition. In defining a view, we use the terms $multipath$ to denote a path in the negotiation tree, containing multiedges.[12]

Definition 6.3: (Valid view). *Let $\mathcal{NT} = \langle \mathcal{N}, \mathcal{R}, \mathcal{E}, \phi \rangle$ be a negotiation tree. A valid view $\mathcal{WT}$ on $\mathcal{NT}$ is a subtree of $\mathcal{NT}$, $\mathcal{W} = \langle \mathcal{N}', \mathcal{R}, \mathcal{E}', \phi \rangle$, where $\mathcal{N}' \subseteq \mathcal{N}$ and $\mathcal{E}' \subseteq \mathcal{E}$ such that:*

- *The state of each node in $\mathcal{WT}$ is $DELIV$;*

12. Recall that in graph theory, a path in a tree is a sequence of nodes where there is a directed edge connecting any node to the following one.


n$_6$=<CreditCard(....) , open, RQ>
n$_7$=<Certified_service(....) , open, CN>
n$_5$=<Corrier_Affiliation(), open, CN>
n$_2$=<Driving licence(....), *deliv*, RQ>
n$_3$=<Id_Card(), *deliv*, RQ>
n$_1$
<Driving_Lic+*l* >
l
n$_2$
n$_3$
n$_4$
n$_6$
l
n$_5$
n$_7$
CONTROLLER POLICY:
p4= ({} Certified_service<-*DELIV*)
n$_1$=<Rental_Car(), *deliv*, CN>
n$_6$=<CreditCard(....) , *deliv*, RQ>
n$_7$=<Certified_service(....) , *deliv*, CN>
n$_2$=<Driving licence(....), *deliv*, RQ>
n$_3$=<Id_Card(), *deliv*, RQ>
n$_1$
<Driving_Lic+*l* >
l
n$_2$
n$_3$
n$_4$
n$_6$
l
n$_5$
n$_7$

(a) (b)

Fig. 10. Example DELIV state propagation.

- $\forall\, n \in \mathcal{N}', \exists$ *a multipath connecting* n *to the leaf nodes in* $\mathcal{N}'$ *and a multipath connecting the root to* n;
- $\forall$ *nonleaf node* $n \in \mathcal{N}'$, $\exists$ *a set of edges* $E = \{e_1, \ldots, e_k\} \in \mathcal{E}$ *rooted at* n *such that:*
  - $\exists!\, e \in E$ *such that* $\phi(e) = ''$;
  - $\forall e \in E$ *such that* $\phi(e) \neq ,$

    $$''\exists\, \overline{e} = \{(n, n_1), \ldots, (n, n_1)\} \in E$$

    and each $\mathcal{T}(n_i) \in \{\mathcal{T}(n_1), \ldots, \mathcal{T}(n_k)\}$ *appear in* $\phi(e)$;
  - $\forall e, e' \in E, \phi(e) \neq \phi(e')$;
  - $\exists!\, e \in E$ *such that* $\phi(e) =$ *'*l '*.

Each valid view denotes a partially ordered list of certificates which ordered disclosure leads to a successful negotiation. The order in which the certificates must be disclosed is implied by the label edges and also keeps into account the constraints imposed by policy preconditions. Fig. 11 presents the tree traversal function that receives as input a valid view and returns the corresponding trust sequence. Function $SequenceGenerator$, besides extracting in the correct order the terms in the view, binds the unknown terms (terms of the form $X(C)$ formalized in Definition 4.3) to a certificate whenever possible. It is a task of the receiver party to complete the binding for its unknown terms in order to determine a complete sequence. The actual satisfaction of the conditions specified in the terms is verified in the subsequent step of the negotiation process, as the certificates are actually disclosed.

The sequence is built from the terms in the nodes composing the view, starting from the tree root. The ordering for sibling edges is given by the edge labels. The shadowed portion of the tree in Fig. 10b is an example of valid view. The correctness of the function is stated by the following theorem.

Theorem 6.1. *Let NT be a negotiation tree for a resource $\mathcal{R}$, and let $\mathcal{WT}$ be a valid view on NT. Let $TS' = [R, C_1, \ldots, C_n]$ be the output of Function SequenceGenerator when its input is $\mathcal{WT}$. TS' is a trust sequence for $\mathcal{R}$.*

The formal proof is reported in Appendix B (which can be found on the Computer Society Digital Library at http://computer.org/tkde/archives.htm).

6.4 Detection of Repeated Nodes

Since parties are not aware of counterpart policies, during policy exchange, the evaluation of some policies can be recursive and create cycles. Repeated terms can be easily detected in the negotiation tree as a term appears twice in the same path. In this case, the $TreeManager$ prunes the portion of the tree creating the redundancy. The pruning is executed from the last repeated node (a leaf node) to the first instance of the term found going up towards the root. Obviously, each term is pruned only if it does not have any other edge in addition to the edge that creates the redundancy.

The following example makes this concept more precise. Consider two nodes n_1 and n_2 connected to the root by different paths. Suppose $\mathcal{T}(n_1) = \mathcal{T}(n_2)$. If the state of n_1 is $DELIV$, it means that n_1 is the root of a valid view for n_1. A pointer can then be used to link n_2 with n_1, appending thus automatically the subtree of n_1 (assuming n_1 is not a leaf node) after n_2. As a result, n_2 can be immediately managed as a $DELIV$ node, without the need of negotiating again the terms in the subtree rooted at n_1. By contrast, if the state of n_1 was $OPEN$, the link is anyway added, in order to avoid redundant policies exchanges, but the state of both nodes is not modified. Finally, the efficiency of the process is further improved if the same conditions hold for two nodes n_1 and n_2 where n_2 is an unknown term. The party can link the nodes, thus immediately binding the unknown term with the certificate type specified in $\mathcal{T}(n_1)$.

In the Rental Car scenario, suppose that $Cars$ requires `Corrier` employees driving licenses in order to disclose its

```
SequenceGenerator($\mathcal{WT}$)

$Input$ :
    $\mathcal{WT} = \langle \mathcal{N}, \mathcal{R}, \mathcal{E}, \phi \rangle$ is a valid view for $\mathcal{R}$ on negotiation tree $\mathcal{NT}$:
$Output$ :
    A trust sequence of certificates $TS$ initialized at the empty set

begin
Let $n_o$ be the root node
If $party(n_o) = local$ then
    Extract the certificate $C$ corresponding to $n_0$;
    $TS = C$;
if $n_o$ is a leaf node then Return(TS);
E=Edge($n_o$)        %Function that returns the edges rooted at $n_o$
if $|E| > 1$ then: Sort_E=Order(E)    % Function that returns the edges
        sorted by the labels starting from the one labeled with$' * l'$
  For each $e \in Sort\_E$
    If $e = \{(n, n_1)...(n, n_k)\}$ then
      For each node $n'$ entering in $e$
        Let $\mathcal{WT}'$ be the subtree rooted at $n'$
        $TS = TS$ $\cup$SequenceGenerator($\mathcal{WT}'$)
      endfor
    else if $e = \{(n, n_1)\}$
      Let $\mathcal{WT}'$ be the subtree rooted at $n_1$
      $TS = TS\cup$ SequenceGenerator($\mathcal{WT}'$)
  endfor
end.
```

Fig. 11. Function SequenceGenerator().

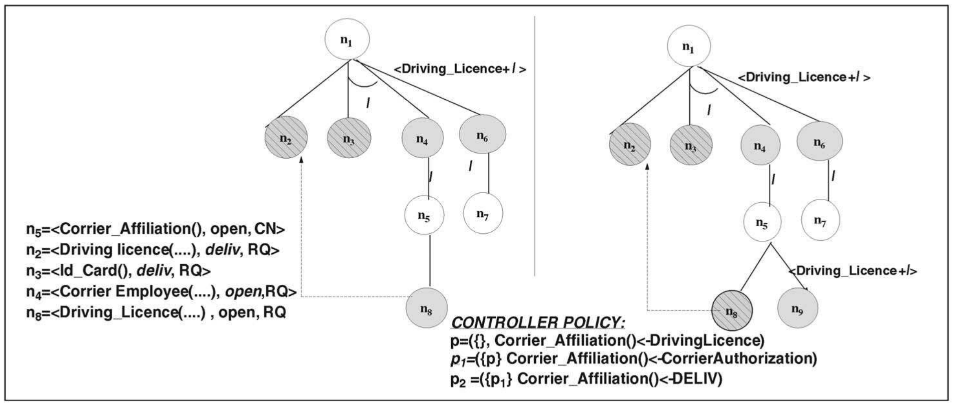

n1
<Driving_Licence+/ >
n2
n3
n4
n6
n5
n7
n8
n9
n5=<Corrier_Affiliation(), open, CN>
n2=<Driving licence(....), deliv, RQ>
n3=<Id_Card(), deliv, RQ>
n4=<Corrier Employee(....), open,RQ>
n8=<Driving_Licence(....) , open, RQ
CONTROLLER POLICY:
p=({}, Corrier_Affiliation()<-DrivingLicence)
p1=({p} Corrier_Affiliation()<-CorrierAuthorization)
p2 =({p1} Corrier_Affiliation()<-DELIV)
(a)
(b)

Fig. 12. Example of link usage.

TABLE 1
Comparison of Trust-$\mathcal{X}$ with Trust Negotiation Systems (Key: Y-Yes, N-No, P-Partial Support)

Requirements	PSPL	TPL	Trust- X	KeyNote	Trust Builder
Well-defined semantics	Y	Y	**Y**	Y	Y
Monotonicity	Y	Y (DTPL)	**Y**	Y	Y
Credential combinations	Y	Y	**Y**	Y	Y
Constraints on attribute values	Y	Y	**Y**	N	Y
Inter-credential constraints	Y	Y	**Y**	N	Y
Credential chains	Y	Y	**P**	N	N
Authentication	Y	N	N	**N**	N
Who submits?	N	N	**N**	N	N
Sensitive Policies	Y	N	**Y**	N	Y
Compliance Checker Modes	Y	P	**Y**	N	P
Credential Validity	Y	Y	**Y**	Y	Y
Credential ownership	N	N	**P**	N	Y
Unified formalism	N	Y	**Y**	Y	N
Interoperable language	N	Y	**Y**	Y	N
Fast policy evaluation	N	N	**Y**	N	N

affiliation with `Corrier` society and then a declaration released by `Corrier` authorizing the employee to rent the car, named `Corrier_Authorization`. A new node, i.e., n_8, is then added to the tree and linked to node n_2, since it refers to the same term (Fig. 12a). As a result, the state of n_8 is immediately set to $DELIV$, thus satisfying the precondition to disclose the following policy, modeled by a new edge connecting n_5 with n_9 (Fig. 12b).

7 Related Work

In this section, we compare Trust-$\mathcal{X}$ with other framework for trust negotiation. The comparison is based on a set of dimensions that deal with expressiveness and semantics. Some of the considered dimensions are taken from [5]. However, we have identified several additional requirements to evaluate trust negotiation systems. The introduced requirements deal with the use of a unified formalism for modeling protected resources and security related information, the adoption of a metalanguage to define the syntax of the negotiation language, and the use of trust tickets to speed up trust establishment processes. Table 1 summarizes the result of our analysis. The analyzed languages are PSPL [3], TPL [4], KeyNote [2], and Trust Builder [9]. PSPL [3] is part of a uniform framework to formulate and reason about information release on the Web. It is a protection language for expressing access control policies for services and release policies for client and service portfolios. The language also includes a policy filtering mechanism, to provide compact policy disclosures and to protect privacy during policy disclosures. The main difference of PSPL with respect to our language is that it only provides a logical definition of the language constructs. Therefore, no actual language is provided. The Trust Policy Language (TPL) [4] is an XML-based framework for specifying and managing role-based access control in a distributed context where the involved parties are characterized by credentials and digital certificates are used for authentication. One of the most important features of TPL is its support for transitive closure and credential chain discovery. Like Trust-$\mathcal{X}$, TPL exploits the flexibility of XML to encode security information and includes a tool called TrustEstablishment (TE for short), for enabling trust relationships between strangers based on public key certificates. However, it does not provide support for sensitive credentials. One of the TE's basic assumptions is that credentials can be disclosed whenever they are requested. Further, TE does not have the notion of sensitive policies, neither in TPL nor in the system architecture. KeyNote [2] is the most well-known trust management language. It was designed to work for a variety of large and small scale Internet-based applications. It provides a unified language for both local policies and credentials. KeyNote policies and credentials, called "assertions," contain predicates describing delegations in terms of actions that are relevant to a given application. As a result KeyNote policies, because of the language intended use for delegation authority, do not handle credentials as a means to establish trust. Therefore, it has several shortcomings with respect to trust negotiations.

TrustBuilder [9] currently represents one of the most significant proposals in the negotiation research area. It is a system supporting trust negotiations between security agents that mediate access to protected resources. TrustBuilder provides a set of negotiation protocols that define the ordering of messages and the type of information messages will contain, and a variety of strategies, to allow strangers to establish trust through the exchange of digital credentials and the use of access control policies. TrustBuilder is the approach that more greatly influenced our work. For instance, we borrow from [9] the use of a tree structure to maintain the progress of a negotiation and keep track of possible alternative strategies. In comparison with Trust-$\mathcal{X}$, Trust Builder does not have any facility to speed up negotiation whenever possible, neither it has the notion of sequence caching. However, Seamons et al., in [6], have explored the issue of supporting sensitive policies, obtained by the introduction of hierarchies in policy definitions.

8 Concluding Remarks

In this paper, we have presented Trust-$\mathcal{X}$, a comprehensive XML-based framework for trust negotiations specifically conceived for a peer-to-peer environment. The framework we have proposed is particularly well-suited for open systems, like Internet, where the involved entities belong to different security domains and need to establish trust before interactions can take place. Trust-$\mathcal{X}$ presents a number of innovative features such as, for instance, the use of trust tickets and the support for different negotiation strategies. Future work includes the extension of $\mathcal{X}$-TNL along several directions such as the possibility of disclosing only portions of a credential during the negotiation process. This will allows us to support a fine-grained protection of the elements of a credential. Another research direction we are currently working on is the compliance with P3P policies [10]. Additionally, we are developing techniques for credential chains discovery, for recovery upon negotiation failures, and for implementing more articulated similarity measures between trust sequences. Finally, an implementation of Trust- $\mathcal{X}$ is in progress on a platform based on Java and the Oracle DBMS. Such protoype systems will allow us to develop a systematic benchmark to assess the system performance under a variety of conditions.

Acknowledgments

The authors wish to thank the anonymous reviewers for their useful comments.

References

[1] E. Bertino, E. Ferrari, and A. Squicciarini, "$\mathcal{X}$-TNL—An XML Based Language for Trust Negotiations," *Proc. Fourth IEEE Int'l Workshop Policies for Distributed Systems and Networks,* June 2003.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System," RFC 2704, Sept. 1999.

[3] P. Bonatti and P. Samarati, "Regulating Access Services and Information Release on the Web," *Proc. Seventh ACM Conf. Computer and Comm. Security,* Nov. 2000.

[4] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid, "Access Control Meets Public Infrastructure, Or: Assigning Roles to Strangers." *Proc. IEEE Symp. Security and Privacy,* May 2000.

[5] K.E. Seamons, M. Winslett, and T. Yu, "Requirements for Policy Languages for Trust Negotiation." *Proc. Third IEEE Int'l Workshop Policies for Distributed Systems and Networks,* June 2002.

[6] K.E. Seamons, M. Winslett, and T. Yu, "Limiting the Disclosure of Access Control Policies during Automated Trust Negotiation," *Proc. Network and Distributed System Security Symp.,* Feb. 2001.

[7] W. Stallings, *Cryptography and Network Security: Principles and Practice,* second ed. Prentice Hall, 1999.

[8] Y. Stanley et al., "An Internet-Based Negotiation Server for E-Commerce," *Very LArge Data Bases J.,* vol. 10, no. 1, pp. 72-90, 2001.

[9] T. Yu, M. Winslett, and K.E. Seamons, "Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation," *ACM Trans. Information and System Security,* vol. 6, no. 1, Feb. 2003.

[10] World Wide Web Consortium, The Platform for Privacy Preferences (P3P) 1.0, available at http://www.w3c.org/TR/P3P, 2002.

[11] W.H. Winsborough and N. Li, "Protecting Sensitive Attributes in Automated Trust Negotiation," *Proc. ACM Workshop Privacy in the Electronic Soc.,* Nov. 2002.

[12] World Wide Web Consortium, available at http://www.w3c.org/, 1998.

Elisa Bertino is a professor of database systems in the Department of Computer Science and Communication at the University of Milan where she is currently the chair of the Department and the director of the DB&SEC laboratory. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Purdue University, and at Telcordia Technologies. Her main research interests include security, privacy, database systems, object-oriented technology, and multimedia systems. In those areas, Dr. Bertino has published more than 200 papers in all major refereed journals, and in proceedings of international conferences and symposia. She is a fellow of the IEEE and a member of the ACM and has been named a Golden Core Member for her service to the IEEE Computer Society.

Elena Ferrari received the PhD degree in computer science from the University of Milano, in 1997. She is a professor of database systems at the University of Insubria at Como, Italy. She has also been on the faculty in the Department of Computer Science at the University of Milano, Italy, from 1998 to March 2001. Dr. Ferrari has been a visiting researcher at George Mason University, Fairfax, Virginia and at Rutgers University, Newark, New Jersey. Her main research interests include database and Web security, temporal, and multimedia databases. In those areas, Dr. Ferrari has published several papers in all major refereed journals, and in proceedings of international conferences and symposia. She is a member of the ACM and the IEEE.

Anna Cinzia Squicciarini received the degree in computer science from the University of Milan in July 2002 with full marks. She is a PhD student at the University of Milan, Italy. During the Autumn of 2003, she was a visiting researcher at the Swedish Institute of Computer Science, Stockholm. Her main research interests include trust negotiations, privacy and, recently, models and mechanisms for privilege and contract management in virtual organizations.