Automated Trust Negotiation Using Cryptographic Credentials

JIANGTAO LI
Intel Corporation
NINGHUI LI
Purdue University
WILLIAM H. WINSBOROUGH
University of Texas at San Antonio

In automated trust negotiation (ATN), two parties exchange digitally signed credentials that contain attribute information to establish trust and make access control decisions. Because the information in question is often sensitive, credentials are protected according to access control policies. In traditional ATN, credentials are transmitted either in their entirety or not at all. This approach can at times fail unnecessarily, either because a cyclic dependency makes neither negotiator willing to reveal her credential before her opponent, because the opponent must be authorized for all attributes packaged together in a credential to receive any of them, or because it is necessary to disclose the precise attribute values, rather than merely proving they satisfy some predicate (such as being over 21 years of age). Recently, several cryptographic credential schemes and associated protocols have been developed to address these and other problems. However, they can be used only as fragments of an ATN process. This paper introduces a framework for ATN in which the diverse credential schemes and protocols can be combined, integrated, and used as needed. A policy language is introduced that enables negotiators to specify authorization requirements that must be met by an opponent to receive various amounts of information about certified attributes and the credentials that contain it. The language also supports the use of uncertified attributes, allowing them to be required as part of policy satisfaction, and to place their (automatic) disclosure under policy control.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Design

Additional Key Words and Phrases: Access control, automated trust negotiation, digital credentials, privacy

Invited submission to the ACM Transactions on Information and System Security, special issue of selected papers of 12th ACM Conference on Computer and Communications Security (CCS'2005). Preliminary version appeared in Proceedings of CCS'2005 under the same title.

Authors' addresses: Jiangtao Li, Intel Corporation, 2111 NE 25th Ave, Hillsboro, OR 97124; email: jiangtao.li@inte.com. Ninghui Li, Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907; email: ninghui@cs.purdue.edu. William H. Winsborough, Department of Computer Science, University of Texas at San Antonio, 6900 N. Loop 1604 West, San Antonio, TX 78249; email: wwinsborough@acm.org.

1. INTRODUCTION

In automated trust negotiation (ATN) [Hess et al. 2002; Seamons et al. 2001; Seamons et al. 2002; Winsborough and Li 2002a; 2002b; 2004; Winsborough et al. 2000; Winslett et al. 2002; Yu and Winslett 2003b; Yu et al. 2003], two parties exchange digitally signed credentials that contain attribute information to establish trust and make access control decisions. In traditional ATN approaches the only way to use a credential is to send it as a whole, thus disclosing all the information in the credential. In other words, a digital credential is viewed as a black-box, and the information in a credential is disclosed in an all-or-nothing fashion. In these approaches sensitive attribute values stored in a credential are protected using access control techniques. There is an access control policy associated with each credential and a credential can be disclosed if its access control policy has been satisfied. Disclosing credentials in an all-or-nothing fashion severely limits the power of ATN. The following are some of the limitations.

—If there is a cyclic dependency among credentials and their policies, negotiations can fail unnecessarily. For example, in a negotiation between A and B, suppose A has a credential c_1 that can be disclosed only if B has c_2, and B has c_2, but can disclose it only if A has c_1. Using traditional ATN techniques, the negotiation would fail because neither c_1 nor c_2 can be disclosed before the other, even though allowing A and B to exchange *both* c_1 and c_2 would not violate either negotiator's policy.

—Because attribute information is disclosed in an all-or-nothing fashion, each attribute can be disclosed only when the policy governing the credential and its entire contents is satisfied, leading to unnecessary failure. For example, suppose B would allow A to access a resource provided A is over 21, and A has a digital driver license that includes A's date of birth (DoB) and address. If A does not want to reveal her address (or her exact DoB) to B, the negotiation would fail, even if A were willing to prove she is over 21.

—When one negotiator does not want to disclose detailed information about his policy and the other negotiator does not want to disclose too much information about her attributes, a negotiation can fail even though the amount of information that needs to be disclosed by each party is acceptable to both. For example, suppose B is a bank that offers a special-rate loan and A would like to know whether she is eligible for such a loan before she applies. B is willing to reveal that his loan-approval policy uses one's DoB, current salary, and the length of the current employment; however, B considers further details of this policy to be a trade secret that he is unwilling to reveal. A would like to know whether she is eligible for the loan while disclosing as little information about her attributes as possible. In particular, A does not want to disclose the exact values of her DoB or salary level. Using traditional ATN techniques, this negotiation would fail.

A number of cryptographic credential schemes and associated protocols have been developed to address these and other problems. Oblivious signature based envelope [Li et al. 2003], hidden credentials [Bradshaw et al. 2004; Holt et al. 2003], and secret handshakes [Balfanz et al. 2003] can be used to address the policy cycle problem. Oblivious Attribute Certificates (OACerts) [Li and Li 2005a], private credentials [Brands 2000], and anonymous credentials [Camenisch and Herreweghen 2002; Camenisch and Lysyanskaya

2001; Chaum 1985; Lysyanskaya et al. 1999] together with zero-knowledge proof protocols can be used to prove that an attribute satisfies a policy without disclosing any other information about the attribute. Certified input private policy evaluation (CIPPE) [Li and Li 2005b] enables A and B to determine whether A's attribute values satisfy B's policies without revealing additional information about A's attributes or B's policies.

While these credential schemes and associated protocols all address some limitations in ATN, they can be used only as fragments of an ATN process. For example, a protocol that can be used to handle cyclic policy dependencies should be invoked only when such a cycle occurs during the negation process. A zero-knowledge proof protocol can be used only when one knows the policy that needs to be satisfied and is willing to disclose the necessary information to satisfy the policy. An ATN framework that harness these powerful cryptographic credentials and protocols has yet to be developed. In this paper, we develop an ATN framework that does exactly that. Our framework has the following salient features.

—The ATN framework supports diverse credentials, including standard digital credentials (such as X.509 certificates [Boeyen et al. 1999; Housley et al. 1999]) as well as OACerts, hidden credentials, and anonymous credentials.
—In addition to attribute information stored in credentials, the ATN framework also supports attribute information that is not certified. For example, oftentimes one is asked to provide a phone number in an online transaction, though the phone number need not be certified in any certificate. In our framework, uncertified attribute information and certified attribute information are protected in a uniform fashion.
—The ATN framework has a logic-based policy langauge that we call Attribute-based Trust Negotiation Language (ATNL), which allows one to specify policies that govern the disclosure of partial information about a sensitive attribute. ATNL is based on the RT family of Role-based Trust-management languages [Li and Mitchell 2003; Li et al. 2002; Li et al. 2003].
—The ATN framework has a negotiation protocol that enables the various cryptographic protocols to be used to improve the effectiveness of ATN. This protocol is an extension of the Trust-Target Graph (TTG) ATN protocol [Winsborough and Li 2002b; 2004].

The rest of this paper is organized as follows. We discuss related work in Section 2, and then review several credential schemes and associated protocols that can be used in ATN in Section 3. In Section 4, we present the language ATNL. In Section 5 we present our negotiation protocol. We give a detailed discussion on how to break policy cycles in Section 6 and on ATNL in Section 7. Finally we conclude our paper in Section 8.

2. RELATED WORK

The approach of using digitally signed credentials to document attributes of entities and delegation relationships among entities has been used in the extensive literature on trust management (TM), e.g., [Blaze et al. 1999; Blaze et al. 1996; Clarke et al. 2001; DeTreville 2002; Ellison et al. 1999; Gunter and Jim 2000; Jim 2001; Li et al. 2003; Li et al. 2002; Li et al. 2003; Rivest and Lampson 1996]. In TM systems, an entity's privilege is based on its attributes instead of its domain-specific identities. An entity's attributes are demonstrated through digitally signed credentials. Delegation is an important mechanism for scalable and flexible trust management. Instead of relying on one or a few commonly trusted parties (e.g., certificate authorities), delegation allows each domain to autonomously determine who can access its resources and how such trust decisions can be

propagated to entities from other domains; this nicely models complicated trust relationships between collaborating parties.

Automated trust negotiation (ATN), introduced by Winsborough et al. [Winsborough et al. 2000], adopts the basic TM approach but considers the fact that credentials may contain sensitive information and need protection just as resources do. ATN techniques enable strangers to establish trust in each other through cautious, iterative, bilateral disclosure of credentials and policies. Winsborough et al. [Winsborough et al. 2000] presented two negotiation strategies: an eager strategy in which negotiators disclose each credential as soon as its access control policy is satisfied, and a "parsimonious" strategy in which negotiators disclose credentials only after exchanging sufficient policy content to ensure that a successful outcome is ensured. Yu et al. [Yu et al. 2003] developed a family of strategies called the disclosure tree family such that strategies within the family can interoperate with each other in the sense that negotiators can use different strategies within the same family. Seamons et al. [Seamons et al. 2001] and Yu and Winslett [Yu and Winslett 2003b] studied the problem of protecting contents of policies as well as credentials. On the aspect of system architecture for trust negotiation, Hess et al. [Hess et al. 2002] proposed the Trust Negotiation in TLS (TNT) protocol, which is an extension to the SSL/TLS handshake protocol by adding trust negotiation features. Winslett et al. [Winslett et al. 2002] introduced the TrustBuilder architecture for trust negotiation systems. The problem of leaking attribute information was recognized by Winsborough and Li [Winsborough and Li 2002b], Seamons et al. [Seamons et al. 2002], and Yu and Winslett [Yu and Winslett 2003a]. Winsborough and Li [Winsborough and Li 2002a; 2002b; 2004] introduced the notion of acknowledgement policies to protect this information and provided a formal notion of safety against illegal attribute information leakage. Further, Irwin and Yu [Irwin and Yu 2005] proposed a general framework for the safety of trust negotiation systems, in which they developed policy databases as a mechanism to help prevent unauthorized information inferences during trust negotiation. Bonatti and Samarati [Bonatti and Samarati 2000] proposed a framework for regulating service access and information release on the web. Their framework supports both certified attributes and uncertified attributes.

Recently, several cryptographic protocols have been proposed to address the limitations in ATN. For example, oblivious signature based envelopes [Li et al. 2003], hidden credentials [Bradshaw et al. 2004; Holt et al. 2003], oblivious commitment based envelopes [Li and Li 2005a], and secret handshakes [Balfanz et al. 2003; Castelluccia et al. 2004] can be used to handle policy cycle problems. Access control using pairing-based cryptography [Smart 2003], anonymous identification [Dodis et al. 2004], certified input private policy evaluation [Li and Li 2005b], hidden policies with hidden credentials [Frikken et al. 2004], and policy-based cryptography [Bagga and Molva 2005] are proposed to address the privacy issues in access control, in particular, these protocols can be used to protect the server's policy and the client's identities or attributes. Recently, Frikken et al. [Frikken et al. 2006] proposed a privacy-preserving trust negotiation protocol. However, their scheme only works for hidden credentials. While all the preceding protocols are useful tools and building blocks for ATN, they are not general enough to solve arbitrary trust negotiation problems in a systematic way. Credential schemes that can be used in ATN include OACerts [Li and Li 2005a], private credentials [Brands 2000], and anonymous credentials [Camenisch and Herreweghen 2002; Camenisch and Lysyanskaya 2001; Chaum 1985; Lysyanskaya et al. 1999]. We will summarize the features of these protocols and credential schemes in the next section.

3. OVERVIEW OF CRYPTOGRAPHIC CREDENTIALS AND TOOLS FOR ATN

We now give an overview of six properties that are provided by cryptographic credential schemes and their associated cryptographic tools. These properties can improve the privacy protection and effectiveness of ATN.

(1) *Separation of credential disclosure from attribute disclosure:* In several credential systems, including private credentials [Brands 2000], anonymous credentials [Camenisch and Herreweghen 2002; Camenisch and Lysyanskaya 2001; Chaum 1985; Lysyanskaya et al. 1999] and OACerts [Li and Li 2005a], a credential holder can disclose her credentials without revealing the attribute values in them. In the OACerts scheme, a user's attribute values are not stored in the clear; instead, they are stored in a committed form in her credentials. When the commitment of an attribute value is stored in a credential, looking at the commitment does not enable one to learn anything about the attribute value. Private credentials and anonymous credentials share somewhat similar ideas: a credential holder can prove in zero-knowledge that she has a credential without revealing it; thus, the attribute values in the credential are not disclosed. For example, consider a digital driver license certificate from Bureau of Motor Vehicles (BMV) consisting of name, gender, DoB, and address. In trust negotiation, a user can show that her digital driver license is valid, *i.e.*, that she is currently a valid driver, without disclosing any of her name, gender, DoB, and address.

(2) *Selective show of attributes:* A credential holder can select which attributes she wants to disclose (and which attribute she does not want to disclose) to the verifier. As each attribute in a credential is in committed form, the credential holder can simply open the commitments of the attributes she wants to reveal. For instance, using the digital driver license, the credential holder can show her name and address to a verifier without disclosing her gender and DoB. Cryptographic properties of the commitment schemes ensure that the credential holder cannot open a commitment with a value other than the one that has been committed.

(3) *Zero-knowledge proof that attributes satisfy a policy:* A credential holder can use zero-knowledge proof protocols [Boudot 2000; Cramer and Damgård 1998; Cramer et al. 1996; Durfee and Franklin 2000] to prove that her attributes satisfy a predicate without revealing the actual attribute values. For example, a credential holder can prove that she is older than 21 by using her digital driver license without revealing any other information about her actual DoB.

(4) *Oblivious usage of a credential:* A credential holder can use her credentials in an oblivious way to access resources using Oblivious Signature Based Envelope (OSBE) [Li et al. 2003], hidden credentials [Holt et al. 2003], or secret handshakes [Balfanz et al. 2003; Castelluccia et al. 2004]. In OSBE, a user sends the contents of her credential (without the signature) to a server. The server verifies that the contents satisfy his requirement, then conducts a joint computation with the user such that in the end the user sees the server's resource if and only if she has the signature on the contents she sent earlier. The hidden credentials and the secret handshakes share the similar concept; however, they assume that the server can guess the contents of the user's credentials; thus the user does not need to send the contents to the server. The oblivious usage of a credential enables a user to obtain a resource from a server without revealing the fact that she has the credential.

(5) *Oblivious usage of an attribute:* A credential holder can use her attributes in an oblivious way to access resources using Oblivious Commitment Based Envelope (OCBE) [Li and Li 2005a]. In OCBE, a credential holder and a server run a protocol such that in the end the credential holder receives the server's resource if and only if the attributes in her credential satisfy the server's policy. The server does not learn anything about the credential holder's attribute values, not even the result of the policy predicate when applied to the attribute values.

(6) *Certified input private policy evaluation (CIPPE):* In CIPPE [Li and Li 2005b], a credential holder and a server run a protocol in which the credential holder inputs the commitments of her attribute values from her credentials, and the server inputs his private policy function. In the end, both parties learn whether the credential holder satisfies the server's policy, without the attribute values being revealed to the server, or the private function, to the credential holder. For example, suppose that the server's policy is that age must be greater than 25 and the credential holder's age is 30. The credential holder can learn that she satisfies the server's policy without revealing her exact DoB or knowing the threshold in the server's policy.

There are other useful properties achieved in the private credentials [Brands 2000] and the anonymous credentials [Camenisch and Herreweghen 2002; Camenisch and Lysyanskaya 2001; Chaum 1985; Lysyanskaya et al. 1999], such as multi-show unlinkable property, anonymous property, etc. Some of these properties require anonymous communication channels to be useful. In this paper, we focus on the six properties described above, because they either have been applied to ATN in the literature before or were developed explicitly for ATN. Our goal is to integrate them into a coherent trust negotiation framework.

Note that we do not assume each negotiating participant supports all six properties. For instance, if one participant uses an anonymous credential system and supports properties 1–3, and the other participant supports properties 1–6, then they can use properties 1–3 when they negotiate trust. We present an ATN framework that can take advantage of these properties when they are available, but that does not require them.

4. THE LANGUAGE OF CREDENTIALS AND POLICIES

In this section, we present the Attribute-based Trust Negotiation Language (ATNL), a formal language for specifying credentials and policies. ATNL is based on RT, a family of Role-base Trust-management languages introduced in [Li and Mitchell 2003; Li et al. 2002; Li et al. 2003]. We first give an example trust negotiation scenario in ATNL, then describe the syntax of ATNL in detail in Section 4.2.

4.1 An Example

In this example, the two negotiators are BookSt (a bookstore) and Alice. We give the credentials and policies belonging to BookSt first, then give those for Alice, and then describe a negotiation process between BookSt and Alice.

BookSt's credentials and policies are given in Figure 1. BookSt has a credential ($\ell 1$) issued by the Small Business Administration (SBA) asserting that BookSt has a valid business license. BookSt is certified in ($\ell 2$) by the Better Business Bureau (BBB) to have a good security process.

BookSt offers a special discount to anyone who satisfies the policy ($m1$), which means that the requester should be certified by StateU to be a student majoring in computer sci-

BookSt's credentials:

$\ell 1$: SBA.businessLicense $\longleftarrow$ BookSt

$\ell 2$: BBB.goodSecProcess $\longleftarrow$ BookSt

BookSt's policies:

$m1$: BookSt.discount(phoneNum $= x_3$) $\longleftarrow$ StateU.student(program $= x_1$) $\cap$ BookSt.DoB(val $= x_2$) $\cap$ Any.phoneNum(val $\Rightarrow x_3$) ; $((x_1 =$ 'cs') $\wedge$ $(x_2 >$ '01/01/1984'))

$m2$: BookSt.DoB(val $= x$) $\longleftarrow$ BMV.driverLicense(DoB $= x$)

$m3$: BookSt.DoB(val $= x$) $\longleftarrow$ Gov.passport(DoB $= x$)

$m4$: disclose(ac, SBA.businessLicense) $\longleftarrow$ true

$m5$: disclose(ac, BBB.goodSecProcess) $\longleftarrow$ true

Fig. 1. The credentials and policies of BookSt

ence, under 21 (as of January 1, 2005), and willing to provide a phone number. Since the discount is a resource, the head of this policy, BookSt.discount(phoneNum $= x_3$), defines a part of the application interface provided by the ATN system using this policy; the parameter phoneNum is made available to the application through this interface. That is, the application will issue a query to determine whether the requester satisfies BookSt.discount(phoneNum $= x_3$), and if it succeeds, the variable x_3 will be instantiated to the phone number of the requester. The body of policy ($m1$) (*i.e.*, the part to the right of $\longleftarrow$) consists of the following two parts.

Part 1: StateU.student(program $= x_1$) $\cap$ BookSt.DoB(val $= x_2$) $\cap$ Any.phoneNum(val $\Rightarrow x_3$)

Part 2: $((x_1 =$ 'cs') $\wedge$ $(x_2 >$ '01/01/1984'))

Part 1 describes the role requirement of the policy and consists of the intersection of 3 roles. To satisfy the role StateU.student(program $= x_1$), one must provide a credential (or a credential chain) showing that one is certified by StateU to be a student; program $= x_1$ means that the value of the program field is required to satisfy additional constraints. In Any.phoneNum(val $\Rightarrow x_3$), the keyword Any means that the phone number does not need to be certified by any party and the symbol $\Rightarrow$ means that the phone number must be provided (enabling it to be returned to the application). Part 2 describes the constraints on specific field values.

BookSt's policies $(m2)$ and $(m3)$ mean that BookSt considers both a driver license from BMV and a passport issued by the government (Gov) to be valid documents for DoB. BookSt's policies $(m4)$ and $(m5)$ mean that BookSt treats his SBA certificate and BBB certificate as non-sensitive resources and can reveal these certificates to anyone. The term ac in $(m4)$ and $(m5)$ denotes access control policy.

Alice's credentials and policies are given in Figure 2. Alice holds three credentials. Credential ($n1$) is issued by StateU and delegates to College of Science (CoS) the authority to certify students. Credential ($n2$) is Alice's student certificate issued by CoS. Credentials ($n1, n2$) prove that Alice is a valid student from StateU. Credential ($n3$) is her digital driver license issued by BMV. For simplicity, we assume that the digital driver license contains only name and DoB. Among her credentials, Alice considers her student certificate to be sensitive, and provides it only to those who have a valid business license from SBA ($p1$). Alice does not protect the content of her driver license, except for its DoB field. She

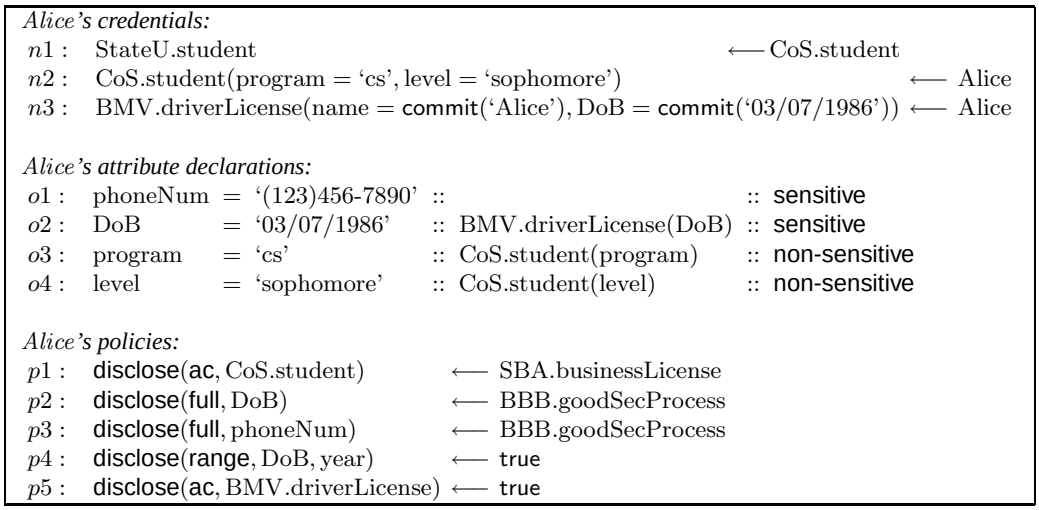

Fig. 2. The credentials and policies possessed by Alice

considers her date of birth and phone number to be sensitive information, thus she reveals them only to organizations whose security practices are adequate to provide reasonable privacy ($p2, p3$). For this, we assume that BBB provides a security process auditing service. Further, Alice is willing to reveal to everyone her year of birth ($p4$) and her digital driver license ($p5$).

A negotiation between BookSt and Alice When Alice requests a discount sale from BookSt, BookSt responds with his discount policy ($m1$). Alice first discloses her driver license ($n3$), which is assumed to be an OACert, to BookSt without revealing her DoB. To protect her phone number and her student certificate, Alice wants BookSt to show a business license issued by SBA and a good security process certificate issued by BBB. After BookSt shows the corresponding certificates ($\ell 1$, $\ell 2$), Alice reveals her student certificate chain ($n1$, $n2$) and phone number ($o1$). As Alice is allowed by her policy $p4$ to reveal her year of birth to everyone, she uses a zero-knowledge proof protocol to prove to BookSt that her DoB in her driver license is between '$1/1/1986$' and '$12/31/1986$'. BookSt now knows that Alice is younger than 21, thus satisfies his discount policy. During the above interactions, Alice proves that she is entitled to obtain the discount.

The above negotiation process uses the first three properties described in Section 3.

4.2 The Syntax

Figure 3 gives the syntax of ATNL in Backus Naur Form (BNF). In the following, we explain the syntax. The numbers in the text below correspond to the numbers of definitions in Figure 3.

Each negotiation party has a *policy base* (3) that contains all information that may be used in trust negotiation. A party's policy base consists of three parts: *credentials*, *attribute declarations*, and *policy statements*. In the following, we discuss each of the three parts in detail.

⟨list of X⟩ ::= ⟨X⟩ | ⟨X⟩ “,” ⟨list of X⟩ (1)
⟨set of X⟩ ::= ϵ | ⟨X⟩ ⟨set of X⟩ (2)

⟨policy-base⟩ ::= ⟨set of credential⟩ ⟨set of attr-decl⟩ ⟨set of policy-stmt⟩ (3)

⟨credential⟩ ::= ⟨member-cred⟩ | ⟨delegation-cred⟩ (4)
⟨member-cred⟩ ::= ⟨role⟩ “←−” ⟨prin⟩ (5)
⟨delegation-cred⟩ ::= ⟨role⟩ “←−” ⟨role⟩ (6)

⟨role⟩ ::= ⟨prin⟩ “.” ⟨role-term⟩ (7)
⟨role-term⟩ ::= ⟨role-name⟩ | ⟨role-name⟩ “(” ⟨list of field⟩ “)” (8)
⟨field⟩ ::= ⟨field-name⟩ “=” (⟨var⟩ | ⟨constant⟩ | ⟨commitment⟩) (9)

⟨attr-decl⟩ ::= ⟨attr-name⟩ “=” ⟨constant⟩ “::” [⟨list of attr-ref⟩]
“::” (“sensitive” | “non-sensitive”) (10)
⟨attr-ref⟩ ::= ⟨prin⟩ “.” ⟨role-name⟩ “(” ⟨field-name⟩ “)” (11)

⟨policy-stmt⟩ ::= ⟨policy-head⟩ “←−” ⟨policy-body⟩ (12)
⟨policy-body⟩ ::= ⟨p-role-req⟩ [“;” ⟨p-constraint⟩] | true (13)
⟨p-role-req⟩ ::= [⟨role⟩ “!”] ⟨conj-of-p-roles⟩ (14)
⟨p-constraint⟩ ::= [⟨pre-cond⟩ “!”] ⟨constraint⟩ (15)
⟨pre-cond⟩ ::= ⟨role⟩ | “false” (16)
⟨conj-of-p-roles⟩ ::= ⟨p-role⟩ | ⟨p-role⟩ “∩” ⟨conj-of-p-roles⟩ (17)
⟨p-role⟩ ::= ⟨prin⟩ “.” ⟨p-role-term⟩ | Any.⟨p-role-term⟩ (18)
⟨p-role-term⟩ ::= ⟨role-name⟩ | ⟨role-name⟩ “(” ⟨list of p-field⟩ “)” (19)
⟨p-field⟩ ::= ⟨field-name⟩ (“=” | “⇒”) (⟨var⟩ | ⟨constant⟩) (20)

⟨policy-head⟩ ::= ⟨p-role⟩ | ⟨dis-ack⟩ | ⟨dis-ac⟩ | ⟨dis-full⟩ | ⟨dis-bit⟩ | ⟨dis-range⟩ (21)
⟨dis-ack⟩ ::= “disclose” “(” “ack” “,” ⟨role⟩ “)” (22)
⟨dis-ac⟩ ::= “disclose” “(” “ac” “,” ⟨role⟩ “)” (23)
⟨dis-full⟩ ::= “disclose” “(” “full” “,” ⟨attr-name⟩ “)” (24)
⟨dis-bit⟩ ::= “disclose” “(” “bit” “,” ⟨attr-name⟩ “)” (25)
⟨dis-range⟩ ::= “disclose” “(” “range” “,” ⟨attr-name⟩, ⟨precision⟩ “)” (26)

Fig. 3. Syntax of ATNL in BNF. Macros and symbols of this figure are defined in Figure 4.

ϵ	empty string
⟨list of X⟩, ⟨set of X⟩	macro parameterized by X
⟨var⟩,⟨constant⟩,⟨prin⟩	a variable, a constant, and a principal
⟨role-name⟩, ⟨field-name⟩, ⟨attr-name⟩	identifiers drawn from disjoint sets
⟨commitment⟩, ⟨precision⟩, ⟨constraint⟩	undefined, will be explained in the text

Fig. 4. Definitions of macros and symbols in Figure 3.

4.2.1 *Credentials and Roles.* Two central concepts that ATNL takes from *RT* [Li et al. 2002; Li et al. 2003] are principals and roles. A principal is identified with an individual or agent, and may be represented by a public key. In this sense, principals can issue credentials and make requests. Semantically, a role designates a set of principals; we say that these principals are members of the role. Each principal has its own localized name space

for roles in which it has sole authority to define (which principals are members of) these roles. Syntactically, a *role* (7) takes the form of a principal followed by a role term, separated by a dot. The simplest kind of a role term consists of just a role name. As roles are parameterized, a role term may also contain fields, which will be explained later. We use A, B, D, S, and V, sometimes with subscripts, to denote principals. We use R, often with subscripts, to denote role terms. A role $A.R$ can be read as A's R role. Only A has the authority to define the members of the role $A.R$, and A does so by issuing role-definition statements.

In ATNL, a credential can be either a membership credential or a delegation credential. A *membership credential* (5) takes the form $A.R \longleftarrow D$, where A and D are (possibly the same) principals. This means that A defines D to be a member of A's role R. A *delegation credential* (6) takes the form $A.R \longleftarrow B.R_1$, where A and B are (possibly the same) principals, and R and R_1 are role terms. In this statement, A defines its R role to include all members of B's R_1 role. Note that, for simplicity, we do not limit number of delegation depths. In other words, we allow unbounded chains of delegation.

For example, BookSt's credential ($\ell 1$) in Figure 1 is a membership credential. It means SBA issued a business license certificate for BookSt. Alice's credential ($n1$) in Figure 2 is a delegation credential. It says that StateU delegates its authority over identifying students to College of Science (CoS). Alice's credential ($n2$) in Figure 2 means that CoS asserts that Alice is a sophomore student in StateU majoring in computer science.

A *role term* (8) is a role name possibly followed by a list of fields. Each *field* (9) has a field name and a field value. A field value can be a variable, a constant, or a commitment. For example, $\mathrm{SBA.businessLicense}$ is a role without any fields, $\mathrm{CoS.student}(\mathrm{program} = \text{`cs'}, \mathrm{level} = \text{`sophomore'})$ and $\mathrm{BMV.driverLicense}(\mathrm{name} = \mathsf{commit}(\text{`Alice'}), \mathrm{DoB} = \mathsf{commit}(\text{`}03/07/1986\text{'}))$ are roles with fields. In the preceding roles, CoS is a principal name, $\mathrm{student}$ is a role name, $\mathrm{program}$ is a field name, 'cs' is a constant of string type, and $\mathsf{commit}(\text{`Alice'})$ is a commitment. In ATNL, a *commitment* takes of the form $\mathsf{commit}(c)$, where c is a constant, and commit denotes the output of a commitment algorithm of a commitment scheme [Damgård and Fujisaki 2002; Pedersen 1991][1].

If a credential is a regular certificate, such as an X.509 certificate [Housley et al. 1999], then each field in the credential takes the form $x = c$, where x is the field name and c is a constant. For example, Alice's student certificate ($n2$) may be an X.509 certificate. When a credential is implemented as a cryptographic certificate, such as an OACert or an anonymous credential, the attribute values are committed in the credential. Therefore, each field takes the form $x = \mathsf{commit}(c)$, where $\mathsf{commit}(c)$ is the commitment of a constant c. For example, Alice's digital driver license ($n3$) is modeled as a cryptographic certificate.

We note that credentials in ATNL can capture proxy credentials used in grid computing. Proxy credentials permit one entity to allow another entity to act on his behalf for a limited period of time. A principal A allows B to act on behalf of A can be encoded as "A.proxy(source=A) $\longleftarrow B$", where a short validity period. For a resource owner C To grant access to some resource to A and to its proxies, one can write "C.access $\longleftarrow A$" and "C.access $\longleftarrow A$.proxy(source=A)". If A allows B to let other parties to act on behalf of A, then A can issue "A.proxy(source=A)$\longleftarrow B$.proxy(source=A)."

[1] In order to have the hiding property, a commitment scheme usually cannot be deterministic, thus the commitment of a value also depends on a secret random value. For simplicity of presentation, we do not explicitly model the random secret in the representation of a commitment.

4.2.2 *Attribute declarations.* Each *attribute declaration* (10) gives the name of the attribute, the value of the attribute, a list of attribute references that correspond to this attribute, and whether this attribute is considered sensitive or not. For example, Alice's attribute declaration ($o1$) in Figure 2 means that Alice has a phone number (123)456-7890 and she considers her phone number to be sensitive information. Alice's attribute declaration ($o3$) indicates that Alice's major is 'cs' and that her program appears in her student certificate, issued by CoS. We use attr to denote attribute names.

Each *attribute reference* (11) corresponds to a field name in a role. The attribute reference is used to link the declared attribute to a specific role field. For example, Alice's DoB attribute declaration has an attribute reference $\mathrm{BMV.driverLicense(DoB)}$, it means that Alice's DoB is documented in the DoB field of the role $\mathrm{BMV.driverLicense}$. It is possible to have several attribute references for an attribute. This means that the attribute is documented by several roles[2]. For example, suppose Alice also has a passport, and her DoB is certified in her passport. Then the attribute declaration for her DoB looks like

$$\begin{aligned}\mathrm{DoB} = {} & \text{`}03/07/1986\text{'} :: \mathrm{BMV.driverLicense(DoB)},\\ & \mathrm{Gov.passport(BirthDate)} :: \textsf{sensitive}\end{aligned}$$

Because the disclosure of attribute values in a credential can be separated from the disclosure of the credential, one purpose of the attribute declarations is to uniformly manage the disclosure of an attribute value that appears in different credentials. That is, the policy author gives disclosure policies for attribute DoB, instead of assigning separate disclosure policies for $\mathrm{BMV.driverLicense(DoB)}$ and $\mathrm{Gov.passport(BirthDate)}$.

When the list of the attribute references is empty, the corresponding attribute does not appear in any role that is certified by a credential. In other words, the attribute is *uncertified* by any authorities. Unlike most prior trust negotiation systems, our framework supports uncertified attributes. In many online e-business scenarios, like the example in Section 4.1, the access control policies require some personal information about the requester, such as phone number and email address, which may not be documented by any digitally signed credentials. Like certified attributes, uncertified attributes may be sensitive, and should be protected in the same way. We treat all attributes uniformly, whether certified or not, by protecting them with disclosure policies.

If an attribute is not sensitive, then the keyword non-sensitive appears at the end of its corresponding attribute declaration. This means that the attribute can be revealed to anyone. There is no access control policy for this attribute. On the other hand, if an attribute is treated as a sensitive resource, the attribute owner will mark its attribute declaration with the keyword sensitive. In this case, if there are disclosure policy statements for this attribute, one has to satisfy the body of one of these statements to learn information about the attribute. If there is no disclosure policy statement for a sensitive attribute, it means the attribute must never be disclosed.

[2]We assume that the attribute values from different roles are the same, however we do not require each principal to use the same field name. For example, BMV may use DoB as the field name for date of birth, whereas Gov uses $\mathrm{BirthDate}$ as the field name. Name agreement for different field names can be achieved using application domain specification documents [Li et al. 2002; Li et al. 2003]. Note that if the attribute values from different roles are different, we treat them as different attributes.

4.2.3 *Policy statements.* In ATNL, a *policy statement* (12) takes the form $\langle$policy-head$\rangle \longleftarrow \langle$policy-body$\rangle$ in which $\langle$policy-body$\rangle$ either is true or takes the form:

$$\begin{array}{l}\text{pre-cond-1} \,!\, B_1.R_1 \cap \cdots \cap B_k.R_k \;\; ; \\ \text{pre-cond-2} \,!\, \psi(x_1, \ldots, x_n)\end{array}$$

where $B_1, \ldots, B_k$ are principals, $R_1, \ldots, R_k$ are role terms, k is an integer greater than or equal to 1, $\mathrm{pre\text{-}cond\text{-}1}$ and $\mathrm{pre\text{-}cond\text{-}2}$ are two pre-conditions (which we discuss shortly), ψ is a constraint from a constraint domain Φ, and $x_1, x_2, \ldots, x_n$ are the variables appearing in the fields of $R_1, \ldots, R_k$. The constraint $\psi(x_1, \ldots, x_n)$ is optional. We call $B_1.R_1 \cap \cdots \cap B_k.R_k$ in the policy statement an *intersection*. In the syntax of ATNL, we do not support multiple occurrences of the same variable in the intersection and pre-conditions of a policy statement.

A *pre-condition* is defined to be a role or the keyword false. $\mathrm{Pre\text{-}cond\text{-}2}$ (16) can be either of these; when it exists, $\mathrm{pre\text{-}cond\text{-}1}$ is a role (14). The motivation for pre-conditions is that, oftentimes, policies may contain sensitive information. The policy enforcer does not want to reveal the policy statement to everyone. If a pre-condition is false, the pre-condition is never satisfied. If the pre-condition is a role, say $B.R$, then the negotiation opponent has to be a member of $B.R$ for the pre-condition to be satisfied. Returning to the policy body, if $\mathrm{pre\text{-}cond\text{-}1}$ is satisfied (or if $\mathrm{pre\text{-}cond\text{-}1}$ is omitted), then the negotiation opponent is allowed to see $B_1.R_1 \cap \cdots \cap B_k.R_k$, otherwise, she is not permitted to know the content of this policy body. Once $\mathrm{pre\text{-}cond\text{-}1}$ is satisfied, if $\mathrm{pre\text{-}cond\text{-}2}$ is also satisfied, then the negotiation opponent is allowed to see the constraint $\psi(x_1, \ldots, x_n)$.

Verifying that a principal satisfies a policy body takes two steps. In the first step, the policy enforcer verifies that the principal has all roles and has provided all uncertified attributes given by $B_1.R_1, \ldots, B_k.R_k$. In the second step, the policy enforcer verifies that the variables in the parameters of $R_1, \ldots, R_k$ satisfy the constraint $\psi(x_1, \ldots, x_n)$. Such two-step policy verification process is made feasible by using cryptographic credentials and the associated cryptographic tools (see Section 3). The first step corresponds to verifying that the principal has the desired credentials. The second step corresponds to verifying that the principal's attribute values in the credentials satisfy the constraint $\psi(x_1, \ldots, x_n)$. If $\psi(x_1, \ldots, x_n)$ is disclosed, which happens only when the second pre-condition has been satisfied, then the principal can use zero-knowledge proof protocols to prove that her attribute values satisfy the constraint or simply reveal all her credentials along with all her attributes; otherwise, the principal can elect to run a private policy evaluation protocol with the policy enforcer, enabling each to determine whether she satisfies the constraint.

Using the example in Section 4.1, BookSt's policy ($m2$) in Figure 1 is a policy statement with no constraint. It states that BookSt considers a driver license from BMV to provide adequate documentation of date of birth. The variable x is used in the statement to indicate that the field value of $\mathrm{BookSt.DoB}$ is the same as the DoB field value in $\mathrm{BMV.driverLicense}$.

The BookSt policy statement ($m1$) means that, in order to be a member of the role $\mathrm{BookSt.discount}$, a principal has to have the roles $\mathrm{BookSt.student}(\mathrm{program} = x_1)$, $\mathrm{BookSt.DoB}(\mathrm{val} = x_2)$, and $\mathsf{Any}.\mathrm{phoneNum}(\mathrm{val} \Rightarrow x_3)$. It further requires that the program field value x_1 in $\mathrm{BookSt.student}$ and the DoB field value x_2 in $\mathrm{BookSt.DoB}$ satisfy the constraint $(x_1 = \text{'cs'}) \wedge (x_2 > \text{'01/01/1984'})$. The symbol $\Rightarrow$ in the role $\mathsf{Any}.\mathrm{phoneNum}(\mathrm{val} \Rightarrow x_3)$ indicates that BookSt must receive a phone number from the negotiation opponent. Where the equality symbol $=$ is used, the policy requires only proof

that the associated field value satisfies any constraints given in the policy statement.

4.2.4 *Policy heads.* The policy head in a policy statement determines which resource is to be disclosed and how it is to be disclosed. A *policy head* (21) can be a role or a disclosure. When the policy head is a role, the statement means that if the negotiation opponent satisfies the policy body, then she is a member of the role. Roles defined in policy statements are controlled by the policy owner and are called *dummy roles* because they serve only to define local policies. If the policy head is a disclosure, then the opponent is granted a permission specified in the disclosure, once the policy body is satisfied. This section explains each type of disclosure and its associated permission. In the rest of this paper, we use ack for acronyms of acknowledgement and ac for acronyms of access control.

We call (the body of) a policy statement with head $\mathsf{disclose}(\mathsf{ack}, A.R)$ (22) an *Ack policy* for the role $A.R$. The opponent has to satisfy one of $A.R$'s Ack policies to gain permission to learn whether the policy enforcer is a member of $A.R$. Until such satisfaction is shown, the policy enforcer's behavior should not depend in any way on whether she belongs to $A.R$.

We call a policy statement with head $\mathsf{disclose}(\mathsf{ac}, A.R)$ (23) an *AC policy* for the credential $A.R \longleftarrow D$. We assume, in this case, that the policy enforcer is D and that D has the membership credential $A.R \longleftarrow D$. When the negotiation opponent has satisfied an AC policy for the credential $A.R \longleftarrow D$, he is authorized to receive a copy of the credential.

We call a policy statement with head $\mathsf{disclose}(\mathsf{full}, \mathrm{attr})$ (24) a *full policy* for the attribute attr. If a full policy for attr is satisfied, the negotiation opponent is allowed to see the full value of attr. When attr is an uncertified attribute, this means the policy enforcer can simply disclose its value. When the field value linked to the attribute reference of attr is a commitment, it means the policy enforcer can open the commitment to the opponent.

We call a policy statement with head $\mathsf{disclose}(\mathsf{bit}, \mathrm{attr})$ (25) a *bit policy* for the attribute attr. Bit policies are defined only for certified attributes. If a bit policy for attr is satisfied, the negotiation opponent has the permission to receive one bit of information about the value of attr, in the sense of receiving the answer to the question whether the value satisfies some predicate. We stress that the one bit information of attr in our context is not necessarily the value of a certain bit in the binary representation of attr, but can be the output of any predicate on attr. More specifically, the policy enforcer can run a private policy evaluation with the opponent in which the opponent learns whether attr, together with other attributes of the enforcer, satisfies the opponent's private policy. Alternatively, the policy enforcer can prove that attr satisfies (or does not satisfy) the opponent's public policy using zero-knowledge proof techniques. While specifying the bit disclosure policy, one should be aware that the bit disclosure of attr is vulnerable to a probing attack. If an adversarial opponent runs the private policy evaluation multiple times using different policies that constrain attr, she may learn more information about the value of attr.

We call a policy statement with head $\mathsf{disclose}(\mathsf{range}, \mathrm{attr}, \mathrm{precision})$ (26) a *range policy* for the attribute attr. Range policies are defined only for certified attributes of certain data types, such as finite integer type, finite float type, and ordered enumeration type. If the range policy for attr is satisfied, then the negotiation opponent has permission to learn that attr belongs to a range with the given precision. For example, if the negotiation opponent has satisfied the policy for $\mathsf{disclose}(\mathsf{range}, \mathrm{DoB}, \mathrm{year})$, then she is allowed to know the year of DoB, but not the exact date. How to specify precision depends on the data type of the attribute. For example, assume credit score takes integer

values from 1 to 1000, and Alice has a credit score of 722 documented in her credit report certificate using cryptographic credential schemes. If BookSt satisfies Alice's policy of disclose(range, $\mathrm{score}, 50$), then Alice can prove to BookSt that her credit score is between 701 and 750 using zero-knowledge proof protocols. Similarly, the policy with head disclose(range, $\mathrm{score}, 10$) means that if the policy is satisfied, the opponent can learn that Alice's credit score is between 721 to 730.

When no Ack policy is specified for an attribute, this indicates that the Ack policy is trivially satisfied. Although a more natural logical interpretation would be that in this case it is trivially unsatisfiable, such an Ack policy would render its attribute unusable, which is not useful. The other types of policies (*i.e.*, AC policy, full policy, bit policy, and range policy) are taken to be unsatisfiable if they are not defined.

So if there is no Ack policy associated with a role $A.R$ in the policy base, then the policy enforcer can reveal to everyone that she is (or is not) a member of $A.R$. On the other hand, if there is no AC policy associated with a role $A.R$ in the policy base, then the policy enforcer should never reveal her credential $A.R \longleftarrow D$ to anyone. If there are both an Ack policy and an AC policy with a role $A.R$, the access control policy is actually the intersection of these two policies, *i.e.*, only if the negotiation opponent satisfies both policies can she see the credential corresponding to $A.R$. That is enforced implicitly through our trust negotiation protocol.

5. THE EXTENDED TRUST TARGET GRAPH (ETTG) PROTOCOL

In this section, we introduce a trust negotiation protocol that can take advantage of ATNL and the cryptographic protocols. This protocol extends the trust-target graph protocol introduced in [Winsborough and Li 2002b; 2004], to deal with the additional features of ATNL and cryptographic certificates.

In this protocol, a trust negotiation process involves the two negotiators working together to construct a *trust-target graph* (TTG). A TTG is a directed graph, each node of which is a trust target. Introduced below, trust targets represent questions that negotiators have about each other. When a requester requests access to a resource, the access mediator and the requester enter into a negotiation process. The access mediator creates a TTG containing one target, which we call the *primary target*. The access mediator then tries to process the primary target by decomposing the question that it asks and expanding the TTG accordingly in a manner described below. It then sends the partially processed TTG to the requester. In each following round, one negotiator receives new information about changes to the TTG, verifies that the changes are legal and justified, and updates its local copy of the TTG accordingly. The negotiator then tries to process some nodes, making its own changes to the graph, which it then sends to the other party, completing the round. The negotiation succeeds when the primary target is satisfied; it fails when the primary target is failed, or when a round occurs in which neither negotiator changes the graph.

5.1 Nodes in a Trust-Target Graph

A node in a TTG is one of the five kinds of targets, defined as follows. We use the notation $e \twoheadleftarrow S$ for several different categories of e, meaning that S belongs to, satisfies, or has the property e. We introduce the various usages of the notation informally as they are used in the following list.

—A *role target* takes the form $\langle V : A.R \stackrel{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, $A.R$ is

a role[3], and S is a principal. S is often $opp(V)$, the negotiator opposing V, but it can be any principal. This target means that V wants to see the proof of $A.R \twoheadleftarrow S$.

—A *policy target* takes the form $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, S is a principal, and policy-id uniquely identifies a policy statement in V's policy base. We assume each negotiator assigns each of her policy statements a unique identifier for this purpose. This target means that V wants to see the proof that S satisfies the body of the statement corresponding to policy-id.

—An *intersection target* takes the form $\langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, S is a principal, $B_1.R_1, \ldots, B_k.R_k$ are roles, and k is an integer greater than 1. This means that V wants to see the proof of $B_1.R_1 \cap \cdots \cap B_k.R_k \twoheadleftarrow S$.

—A *trivial target* takes the form $\langle V : S \overset{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, and S is a principal. Representing questions whose answers are always affirmative, trivial targets provide placeholders for edges that represent credentials in the TTG.

—An *attribute goal* takes the form $\langle V : \text{attr} \overset{?}{\twoheadleftarrow} S \rangle$, in which attr is the name of an attribute in S's attribute declaration. This goal means that V wants to learn some information about the value of attr, *e.g.*, V may want to learn the full value of the attribute, or to learn partial information about the attribute, e.g., whether it satisfies a policy.

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

5.2 Edges in a Trust-Target Graph

Seven kinds of edges are allowed in a trust-target graph, listed below. We use $\leftharpoondown$ to represent edges in TTG's.

—A *credential edge* takes the form $\langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle \leftharpoondown \langle V : e \overset{?}{\twoheadleftarrow} S \rangle$, in which $A.R$ is a role, and e is either a principle or a role. We call $\langle V : e \overset{?}{\twoheadleftarrow} S \rangle$ a credential child of $\langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle$. (We use similar "child" terminology for other kinds of edges.) An edge always points from the child to the parent. Unlike the other kinds of edges, a credential edge needs to be *justified* to be added into the TTG; a credential edge is justified if the edge is accompanied by a credential that proves $A.R \twoheadleftarrow e$.

—A *policy edge* takes the form $\langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle \leftharpoondown \langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S \rangle$, in which policy-id is a policy identifier and $A.R$ is the role in the head of the policy statement (that corresponds to policy-id).

—A *policy control edge* takes the form $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S \rangle \leftharpoondown \langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle$, in which policy-id is a policy identifier and $A.R$ is one of the pre-conditions in the policy statement.

—A *policy expansion edge* takes the form $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S \rangle \leftharpoondown \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S \rangle$, in which policy-id is a policy identifier and $B_1.R_1 \cap \cdots \cap B_k.R_k$ is the intersection in the policy statement. If $k > 1$, the policy expansion child is an intersection target; otherwise, it is a role target. Each policy expansion edge has associated with it up to one tag consisting of a constraint.

—An *intersection edge* takes the form $\langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S \rangle \leftharpoondown \langle V : B_i.R_i \overset{?}{\twoheadleftarrow} S \rangle$, where i is in $1..k$, and k is greater than 1.

[3]Technically, the roles in the TTG correspond syntactically to the non-terminal ⟨p-role⟩, rather than to ⟨role⟩. This is because they are derived from policies, and so can contain symbols such as Any and ⇒.

—An *attribute edge* takes the form $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : \mathrm{attr} \overset{?}{\twoheadleftarrow} S\rangle$, in which S is the negotiation opponent of V, attr is an attribute name, and $A.R$ is a role. This is used when the attribute attr is linked to a specific field in $A.R$ in S's attribute declarations.

—An *attribute control edge* takes the form $\langle V : e \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle opp(V) : \text{policy-id} \overset{?}{\twoheadleftarrow} V\rangle$, in which $opp(V)$ denotes the opponent of V, policy-id is a policy identifier, and e is the role or attribute name in the head of the policy statement. Attribute control edges are used for handling disclosure policies. Each attribute control edge has a tag consisting of one of ac, ack, full, bit, or range; in the range case, it also includes a precision parameter.

The optional tag on a policy expansion edge is used to express the constraint portion of the policy statement identified by policy-id. The tag on an attribute control edge characterizes the information that V can gain permission to learn by satisfying the body of the statement identified by policy-id.

5.3 State Propagation in TTG

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

Each target has a *satisfaction state*, which has one of three values: satisfied, failed, and unknown. For each role target or intersection target, there is a *field-state table*. The field-state table is used to maintain information about the field values in the corresponding role or intersection target. Each field-state table contains zero or more tuples. Each tuple has multiple *field states*, one for each field in the target, *i.e.*, for each field in the role or intersection target, there is a field state corresponding to it in the tuples of the field state table. Each field state has three entries, one for full disclosure, one for bit disclosure, and one for range disclosure[4]. Each entry can have value false, indicating that the corresponding disclosure policy has been found to be unsatisfiable by the negotiator desiring to know the field value. Entry values can also be of several other types, as will be discussed shortly. Each attribute goal has an *attribute state*. An attribute state has three entries, one for full disclosure, one for bit disclosure, and one for range disclosure. Each entry can be one of the three values: true, false, or unknown. A true value means the corresponding policy in that entry has been satisfied. A unknown value means the corresponding policy has not been satisfied yet. A false value means the corresponding policy is failed by the opponent.

We now describe how to determine the satisfaction state of targets, the field state of fields, the attribute state of attribute goals, and corresponding local states.

5.3.1 *Satisfaction state.*

The trust target satisfaction state is determined as follows:

(1) *Role target.* The initial satisfaction state of a role target is unknown. It becomes satisfied when one of its credential children or one of its policy children is satisfied, and for each field in its role with the $\Rightarrow$ symbol (the verifier wants to see the full value of this field), the full policy entry in its field state table is not unknown (the full value of the field has been disclosed). It becomes failed when it is fully processed and it has no child, or all of its children are failed, or there exists some field in the role with

[4]In this specification, we support only a single range policy for each field, though it can be easily extended to allow multiple range policies.

the $\Rightarrow$ symbol whose full entry value in the field state is false. It becomes satisfied when one of its children is satisfied and each field in the role with the $\Rightarrow$ symbol has a non-false value in the full entry.

(2) *Policy target.* Let $\mathrm{policy\text{-}id}$ be the policy identifier in this policy target. If the policy body corresponding to $\mathrm{policy\text{-}id}$ is the constant true, then the inital satisfaction state of this target is satisfied. Otherwise, the initial satisfaction state of a policy target is unknown.
   (a) If there is no constraint in the policy corresponding to $\mathrm{policy\text{-}id}$, the satisfaction state of the policy target becomes satisfied when it is fully processed and its policy expansion child is satisfied. It becomes failed when it is fully processed and either it has no policy expansion child (the pre-condition for the policy has not been satisfied) or its policy expansion child is failed.
   (b) If there is a constraint in the policy corresponding to $\mathrm{policy\text{-}id}$, the satisfaction state of the policy target becomes satisfied when it is fully processed, its policy expansion child is satisfied, and the constraint is evaluated and also satisfied. If the constraint has been revealed (*i.e.*, any policy control child for the constraint has been satisfied), it can be evaluated when the value or the range of each variable in the constraint has been disclosed. If the constraint is private, it can be evaluated by using the private policy evaluation, or by conventional means once the full value of each variable in the policy has been disclosed. It becomes failed when it is fully processed and it has no policy expansion child, or its policy expansion child is failed, or the constraint uses a variable whose corresponding field-policy entries are all false, or the constraint is not satisfied.

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

(4) *Trivial target.* A trivial target is always satisfied.

5.3.2 *Attribute state.* There are three entries in the attribute state of an attribute goal, one for full policy, one for bit policy, and one for range policy. The initial value of each entry is unknown. If the satisfaction state of the attribute control child of the attribute goal becomes satisfied, we mark the value of the corresponding entry in the attribute state to be true. On the other hand, if the satisfaction state of the attribute control child becomes failed, we mark the value of the corresponding entry in the attribute state to be false.

5.3.3 *Field-state table.* The field-state table for each role or intersection target is initially set to be empty. The values in the field-state table are computed based on the field-state tables of its children or its grandchildren, as they become available. If the given target is an intersection target, then the field-state table is the cross-product of all the field-state tables in its intersection children. If the given target is a role target and has a delegation-credential child, then the field-state table is copied from its child. If the given target is a role target and has a policy child, then the field-state table is the subset of all tuples in the field-state table of its grandchild in which the field values satisfy the constraint of its policy child's policy. If the given target has a non-delegation credential child and the corresponding credential is a standard credential (*i.e.*, one not containing commitments, such as X.509 certificate), then the precise value of the field is copied to the full entry of the field-state table. Otherwise, if the current target has an attribute child, depending on the attribute state of the attribute goal, the opponent reveals the attribute value as follows. If

the full entry in the attribute state of the attribute child is true, then the opponent reveals the exact value of the field and the value is added to the full entry of the field state in the field-state table. If the bit entry in the attribute state of the attribute child is true, the bit entry in the field state is set to contain a reference to the current role target, as well as a reference to the corresponding attribute in that role target. As field information flows up the TTG from its sources to constraints that the fields' values must satisfy, these references enable the negotiator to determine which fields of which credentials must satisfy those constraints. If a range disclosure entry in the attribute state of the attribute child is true, the opponent proves that the field value belongs to some range according the precision parameter. The disclosed range is then written into the range entry of the field state. If an entry in the attribute state of the attribute child is false, then we write the value false into the corresponding entry in the field state.

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

5.4 Messages in the Protocol

As described before, negotiators cooperate by using the protocol to construct a shared TTG, a copy of which is maintained by each negotiator. Negotiators take turns transmitting messages each of which contains a sequence of TTG update operations and a set of credentials to be used in justifying credential edges. Negotiators may also run a set of cryptographic protocols, described in Section 3, during the ETTG protocol. On receiving an update operation, a negotiator verifies it is legal before updating its local copy of the shared TTG. The following are *legal* TTG update operations:

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

The legal initial processing state of a trivial target is fully-processed. Both a policy target and an intersection target are initially opponent-processed. An attribute goal is initially verifier-processed. A role target is initially either opponent-processed or verifier processed. These operations construct a connected graph. Satisfaction states of trust targets, field-state tables of trust targets, and attribute states of attribute goals are not transmitted in messages; instead, each negotiation party infers them independently.

5.5 Node Processing

Previously we described the ETTG negotiation protocol, in which two negotiators exchange update messages. The protocol defines what updates are legal, and the receiver of a message can verify that the updates in the message are legal. We now describe procedures for *correct processing*, which update the TTG in a manner designed to satisfy the primary target whenever this is possible, while enforcing each negotiator's policies. Correct processing continues until either the primary target is satisfied (negotiation success),

it is failed (negotiation failure), or neither negotiator can perform a correct update (also negotiation failure).

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

5.5.1 *Node Processing State Initialization.* When a new node is added to a TTG, its processing state should be initialized as follows:

—A trivial target is fully processed, its satisfaction state is satisfied, and it has no field state.
—For a role target, $\langle K_V : K.r \stackrel{?}{\leftarrow} K_S \rangle$, if $K.r$ is a dummy role (defined in a policy statement), the target is opponent-processed, which means that the opponent cannot process it; otherwise, it is verifier-processed. The initial satisfaction state for this target is unknown. If there are fields in the role $K.r$, we add an empty field-state table for this target.
—A policy target is initially opponent-processed. If the policy body corresponding to the policy identifier in this target is `true`, then the initial satisfaction state is satisfied, otherwise, the satisfaction state is unknown. There is no field state for this target.
—An intersection target is initially opponent-processed. The initial satisfaction state for this target is unknown. If there exist fields in any roles in the intersection target, we create an empty field-state table for this target.
—An attribute goal is initially verifier-processed. The attribute state for the attribute goal is set to be empty. That is, there is no entry in the attribute state corresponding to this attribute goal.

5.5.2 *Verifier-Side Processing.* We now describe how a negotiator V processes a node when it is the verifier of the node. These rules apply to nodes that are not yet marked verifier-processed. We assume that $\mathrm{mgu}(R, R')$ returns a most general substitution that makes the corresponding fields of role names R and R' syntactically identical.

1. Processing $T = \langle V : A.R \stackrel{?}{\leftarrow} S \rangle$
(a) For each of V's local policy statements in which $A.R'$ is a dummy role in the policy head, $A.R$ and $A.R'$ are unifiable, and $\mathrm{policy\text{-}id}$ is the corresponding policy identifier, V can add a policy edge $T \leftarrowtail \langle V : \mathrm{policy\text{-}id} \stackrel{?}{\leftarrow} S \rangle$.
(b) V can mark T as verifier-processed only after (a) is *done*, meaning that all edges that can be added according to (a) have been added.
(c) If one of the policy children has been satisfied, V copies the values in the field state of each field from its grandchild, the policy expansion child of the newly satisfied policy child, to the field states in its current target.

2. Processing $T = \langle V : \mathrm{policy\text{-}id} \stackrel{?}{\leftarrow} S \rangle$
(a) Let $[\mathrm{pre\text{-}cond\text{-}1}\ !]\ B_1.R_1 \cap \cdots \cap B_k.R_k\ ;\ [[\mathrm{pre\text{-}cond\text{-}2}\ !]\ \psi(x_1, \ldots, x_n)]$ be the policy body corresponding to $\mathrm{policy\text{-}id}$, and let $\sigma = \mathrm{mgu}(R, R')$ exist in which R and R' are the

role names in the parent node and the policy head, respectively. If $\mathrm{pre\text{-}cond\text{-}1}$ is a role, say $A_1.R_1$, V can add a policy control edge $T \leftarrowtail \langle V : A_1.\sigma(R_1) \overset{?}{\twoheadleftarrow} S \rangle$.
(b) After (a) is done and $\langle V : A_1.\sigma(R_1) \overset{?}{\twoheadleftarrow} S \rangle$ is satisfied, V can add a policy expansion edge $T \leftarrowtail \langle V : B_1.\sigma(R_1) \cap \cdots \cap B_k.\sigma(R_k) \overset{?}{\twoheadleftarrow} S \rangle$. V can also do so in the case that there is no pre-condition for the intersection.
(c) If there is no constraint for this policy, (c) is trivially done. Otherwise, if $\mathrm{pre\text{-}cond\text{-}2}$ is a role, say $A_2.\sigma(R_2)$, V can add a policy control edge $T \leftarrowtail \langle V : A_2.\sigma(R_2) \overset{?}{\twoheadleftarrow} S \rangle$.
(d) After (c) is done and either $\langle V : A_2.\sigma(R_2) \overset{?}{\twoheadleftarrow} S \rangle$ is satisfied or there is no pre-condition for the constraint, V can add a tag to the policy expansion edge with the constraint in it.
(e) V can mark T as verifier-processed only after (d) is *done*, or if there is no constraint for the policy after (b) is *done*, or if (a) is *done* and the policy control child added in (a) has been marked fail.
(f) T is satisfied only if its policy expansion child has been satisfied and the constraint (if it exists) in the tag has been satisfied. The constraint can be evaluated only if there is enough information in the field states corresponding to the required fields. There are the following three cases.

—When each of the variables in the constraint has in its full entry in the field state a non-empty value that is not equal to false (*i.e.*, all the required attribute values have been fully disclosed), V determines whether those values satisfy the constraints in the policy statement identified by policy-id. If the constraint is satisfied, V marks T to be fully-satisfied; otherwise, V marks T to be failed. If the constraint is public, then both V and S can verify the constraint; otherwise, only V verifies the constraint.

—When each of the variables in the constraint has in its full and bit entries in the field states non-empty values not equal to false (*i.e.*, V is allowed to see either one bit or full information for each of the required attributes in the constraint), the bit entry in each field state contains a reference to the role target corresponding to the credential providing the field's value. If the constraint is private, V runs a private policy evaluation protocol with S to evaluate the constraint. If the constraint is public, S can prove to V using zero-knowledge proof techniques that her attributes satisfy (or do not satisfy) the constraint by using the information stored in the bit entries of the field states to identify the credentials and fields within them from which each variable in the constraint obtains its value.

—When some variables in the constraint have in their range entries in the field states a non-empty value that is not equal to false (*i.e.*, all the required attribute values have been disclosed with certain precisions), V checks whether the range information in these range entries of the field states, when added to the available information about the other variable values, is enough to determine whether the constraint can be satisfied. If the range information is enough to evaluate the constraint, V verifies the constraint accordingly. If the constraint is satisfied, V marks T to be fully-satisfied, otherwise, V marks T to be failed. If the constraint cannot be evaluated, the satisfaction state of T remains unknown. If the constraint is public, then both V and S can verify the constraint, otherwise, only V verifies the constraint.

3. Processing $T = \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S \rangle$
(a) V can add the k intersection edges, $T \leftarrowtail \langle V : B_i.R_i \overset{?}{\twoheadleftarrow} K_S \rangle$, $1 \leq i \leq k$

(b) V can mark T verifier-processed only after (a) is done.
(c) For each of its intersection children, if it has been satisfied, V copies the values in the field state of each field from the child target to the field states of its current target. The intersection target is satisfied if all of its intersection children are satisfied.

5.5.3 *Opponent-Side Processing.* We now describe how a negotiator S process a node when it is the opponent of the verifier of the node. These rules apply to nodes that are not yet marked opponent-processed.

1. Processing $T = \langle V : A.R \overset{?}{\leftarrow} S \rangle$
(a) If there exists a policy statement with head disclose(ack, $A.R$), S can add an attribute control edge $T \leftarrowtail \langle S : \text{ack-id} \overset{?}{\leftarrow} V \rangle$, where ack-id is the policy identifier for the ack policy.
(b) After (a) is done and $\langle S : \text{ack-id} \overset{?}{\leftarrow} V \rangle$ is satisfied (if it exists), if S has the credential $A.R \longleftarrow S$, and if there exist a policy statement ac-id with head disclose(ac, $A.R$), S can add an attribute control edge $T \leftarrowtail \langle S : \text{ac-id} \overset{?}{\leftarrow} V \rangle$.
(c) After (b) is done and $\langle S : \text{ac-id} \overset{?}{\leftarrow} V \rangle$ (if it exists) is satisfied, S can add the credential edge $T \leftarrowtail \langle V : S \overset{?}{\leftarrow} S \rangle$. Once S reveals her credential $A.R \longleftarrow S$, S mark T to be fully-satisfied. If the credential disclosed is a traditional certificate (and all the attributes in the credential has been disclosed as well), S copies the attribute values to the full entries of the field states in node T.
(d) After (a) is done and $\langle S : \text{ack-id} \overset{?}{\leftarrow} V \rangle$ is satisfied, if S has a delegation credential $A.R' \longleftarrow A_1.R_1$, $A.R$ and $A.R'$ are unifiable and $\sigma = \text{mgu}(R, R')$, S can add the credential edge $T \leftarrowtail \langle V : A_1.\sigma(R_1) \overset{?}{\leftarrow} S \rangle$.
(e) S can mark T as opponent-processed if T is satisfied, or all of the above steps are done.

2. Processing $T = \langle V : \text{attr} \overset{?}{\leftarrow} S \rangle$
(a) If there exists a policy statement full-id with head disclose(full, attr), S can add an attribute control edge $T \leftarrowtail \langle S : \text{full-id} \overset{?}{\leftarrow} V \rangle$. S adds a full entry to the attribute state and sets its value to be unknown. If the attribute control child has been satisfied, S sets the full entry of the attribute state to be true. Once the full entry of the attribute state becomes true, S reveals the attribute value corresponding to $attr$, and copies the value to the full entry of the field state in the parent node of T.
(b) If there exists a policy statement bit-id with head disclose(bit, attr), S can add an attribute control edge $T \leftarrowtail \langle S : \text{bit-id} \overset{?}{\leftarrow} V \rangle$. S adds a bit entry to the attribute state and sets its value to be unknown. If the attribute control child has been satisfied, S sets the bit entry of the attribute state to be true. Let us denote by P the parent node of T. Once the bit entry of the attribute state becomes true, S writes the identity of P to the bit entry of the field state in P.
(c) If there exists a policy statement range-id with head disclose(range, attr, precision), S can add an attribute control edge $T \leftarrowtail \langle S : \text{range-id} \overset{?}{\leftarrow} V \rangle$. S adds a range entry with the precision parameter to the attribute state and sets its value to be unknown. If the attribute control child has been satisfied, S sets the range entry of the attribute state to be true. Then S runs a zero-knowledge proof protocol with V to prove that attr belongs to a range with certain precision, and writes the range value into the range entry of the field state in the parent node of T.
(d) S can mark T as opponent-processed if T is satisfied, or all of the above steps are done.

5.6 Examples of The Extended Trust-Target Graph (ETTG) Protocol

We now give two examples that illustrate the ATNL language and the ETTG protocol. Example 1 shows the usage of various types of credentials in ATN; example 2 deals with the scenario in which the constraint is private.

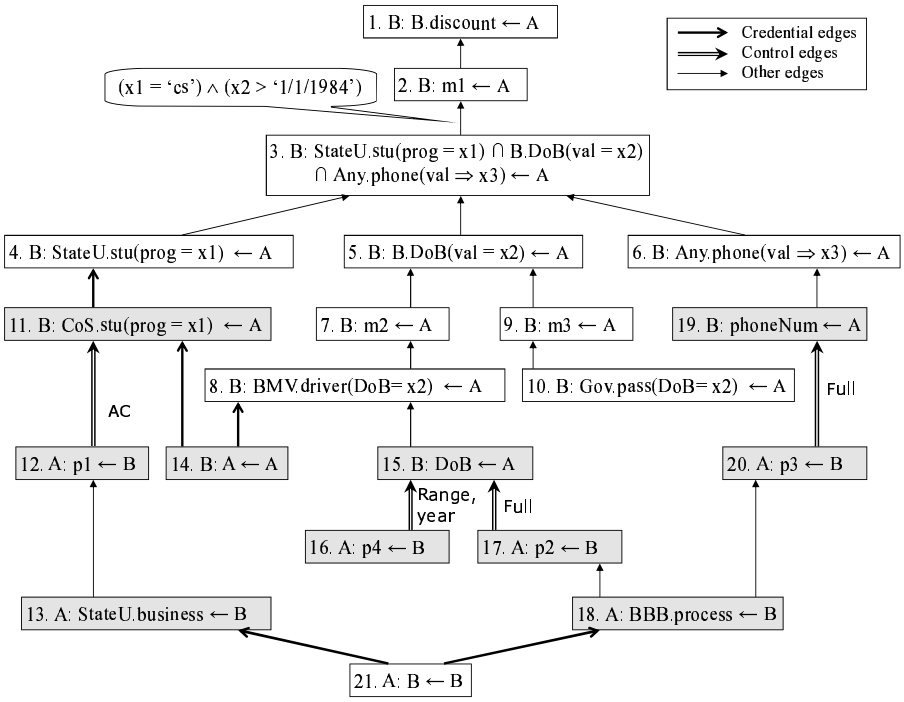

Fig. 5. Final TTG for the bookstore example. In this figure, ← denotes the symbol ↞, A denotes Alice, and B denotes BookSt. The white nodes are created by BookSt and the grey nodes are created by Alice.

EXAMPLE 1. This example is a simple instance of the ETTG protocol and illustrates the usage of the first three properties described in Section 3. Referring to the bookstore example in Section 4.1, we depict the final TTG in Figure 5. Alice and BookSt run the ETTG protocol as follows: As BookSt wants to see the proof of $\mathrm{BookSt.discount} \twoheadleftarrow \mathrm{Alice}$ in order to grant Alice access, BookSt creates the primary target (node 1) for the negotiation and sets its satisfaction state to be unknown. If node 1 becomes satisfied, then the negotiation succeeds. In BookSt's policy base, there is a policy statement ($m1$) for $\mathrm{BookSt.discount}$, hence BookSt creates a policy target (node 2) and adds a policy edge between node 1 and node 2. As the policy statement ($m1$) has no pre-conditions, BookSt reveals the policy by adding a policy expansion child (node 3) and a constraint tag between the parent (node 2) and the child (node 3). Based on the policy ($m1$), BookSt wants to see Alice's phone number and wants to know whether Alice's program and DoB satisfy his constraint. BookSt then creates node 4, 5, 6 and adds them as intersection children to node 3. Since the role $\mathrm{BookSt.DoB}$ is a dummy role and there are policies ($m2, m3$) associated

with it, BookSt adds a policy target (node 7) as the policy child to node 6. BookSt then adds a policy expansion child (node 8) to node 7. Similarly, BookSt adds node 9 and 10. Essentially, BookSt wants to see Alice's DoB from either a driver license or a passport. Now BookSt cannot process the TTG any more.

After receiving the TTG from BookSt, Alice begins to process the graph. Alice first discloses her credential $n1$ (as it is not sensitive) and adds a credential child (node 11). She cannot disclosure her student credential ($n2$) immediately, as there exists an AC policy ($p1$) for $n2$. Therefore Alice adds a policy target (node 12) and expands it with a role target (node 13). Note that the edge between node 11 and 12 is an attribute control edge, which means that if node 12 is satisfied, then Alice can disclose her student credential ($n2$). Alice also reveals her digital driver license (without revealing her DoB) to BookSt, creates a trivial target (node 14), and adds a credential edge between node 8 and node 14. At this point, Alice notices that she needs to prove she is younger than '1/1/1984' and to reveal her phone number, she adds an attribute goal (node 15) for her DoB attribute and another attribute goal (node 19) for her $\mathrm{phoneNum}$, she also expands the TTG by adding nodes 16, 17, 18, 20. As the node 16 is trivially satisfied (because the policy for $p4$ is true), Alice proves to BookSt that she was born in 1986. Alice's year of birth flows up from node 8 to node 3.

BookSt adds a trivial target (node 21) and shows to Alice his $\mathrm{StateU.businessLicense}$ certificate and $\mathrm{BBB.goodSecProcess}$ certificate, which triggers the satisfaction of the nodes 12 and 20. Alice then reveals her student credential ($n2$) and her uncertified $\mathrm{phoneNum}$. The values of Alice's attribute $\mathrm{program}$ and $\mathrm{phoneNum}$ flow up to node 3, where BookSt verifies that Alice's attributes satisfy the constraint. Finally, the primary target is satisfied and the negotiation succeeds.

EXAMPLE 2. This example illustrates the usage of properties 1, 2, and 6 (private policy evaluation) described in Section 3. Suppose BankWon, an online bank certified by National Credit Union Administration (NCUA), offers a special-rate loan. Before applying the loan, an applicant is required to show a valid driver license. The loan policy is that the applicant must have either (1) a credit score more than 680 and an income more than 55k, or (2) a credit score more than 700 and an income more than 45k. BankWon considers his loan policy as private information, and discloses (the thresholds of) the policy only to BankWon's preferred members. Carol, who is not one of BankWon's preferred members, wants to know whether she is eligible for that loan. She has a credit report from Experian and a tax certificate from Internal Revenue Service (IRS). Carol considers her credit score and her income to be sensitive attributes. BankWon and Carol's credentials and policies are given in Figure 6.

Using the ETTG protocol, BankWon and Carol can negotiate trust successfully. The final TTG of the negotiation is given in Figure 7. In the ETTG protocol, BankWon first creates a primary target (node 1), a policy target (node 2), and a role target (node 3). The edge between node 2 and 3 is a policy control edge. After Carol reveals her driver license and adds node 4, BankWon is able to expand the loan policy and adds nodes 5 – 14. Carol then reveals her tax certificate and credit report without revealing her sensitive attributes to BankWon, and adds two attribute goals (node 15 and 19) to TTG. As node 6 is not satisfied, the constraint of the loan policy is not revealed to Carol. However, as the bit policies for Alice's income and score are satisfied, Carol and BankWon are able to run a private policy evaluation on income and score with BankWon's private constraint.

Bank's credentials and policies:

$q1$: NCUA.member $\longleftarrow$ Bank

$r1$: Bank.loan $\longleftarrow$ BMV.driverLicense !
IRS.tax(income $= x_1$) $\cap$ Bank.credScore(val $= x_2$) ;
Bank.preferred !
$((x_1 > 680) \wedge (x_2 >$ '55k')) $\vee$ $((x_1 > 700) \wedge (x_2 >$ '45k')

$r2$: Bank.credScore(val $= x$) $\longleftarrow$ Equifax.credReport(score $= x$)

$r3$: Bank.credScore(val $= x$) $\longleftarrow$ Experian.credReport(score $= x$)

$r4$: Bank.credScore(val $= x$) $\longleftarrow$ TransUnion.credReport(score $= x$)

$r5$: disclose(ac, NCUA.member) $\longleftarrow$ true

Carol's credentials:

$s1$: Experian.credReport(score = commit(720)) $\longleftarrow$ Carol

$s2$: IRS.tax(income = commit('65k'), employer = commit('Company A')) $\longleftarrow$ Carol

$s3$: BMV.driverLicense(name = 'Carol', DoB = commit('06/18/1972')) $\longleftarrow$ Carol

Carol's attribute declarations:

$t1$: DoB = '06/18/1972' :: BMV.driverLicense(DoB) :: sensitive

$t2$: score = 720 :: Experian.credReport(score) :: sensitive

$t3$: income = '48k' :: IRS.tax(income) :: sensitive

$t4$: employer = 'Company A' :: IRS.tax(employer) :: non-sensitive

Carol's policies:

$u1$: disclose(full, DoB) $\longleftarrow$ BBB.goodSecProcess

$u2$: disclose(bit, score) $\longleftarrow$ NCUA.member

$u3$: disclose(range, score, 50) $\longleftarrow$ true

$u4$: disclose(bit, income) $\longleftarrow$ true

$u5$: disclose(range, income, 10k) $\longleftarrow$ BBB.goodSecProcess

$u6$: disclose(ac, Experian.credReport) $\longleftarrow$ true

$u7$: disclose(ac, IRS.tax) $\longleftarrow$ true

$u8$: disclose(ac, BMV.driverLicense) $\longleftarrow$ true

Fig. 6. The credentials and policies for Example 2

After the private policy evaluation outputs true (*i.e.*, Carol's certified attributes satisfy the constraint), node 2 becomes satisfied. In the end, node 1 is also satisfied and the ETTG protocol succeeds.

6. BREAKING POLICY CYCLES

In this section, we discuss how to break policy cycles using the ETTG protocol.

6.1 Examples of Policy Cycles

We begin with a few examples of policy cycles and illustrate how such cycles can be broken using cryptographic protocols.

EXAMPLE 3. Both Alice and Bob have a CIA credential. Alice will reveal that she has a CIA credential only to those who are also agents of the CIA. Similarly, Bob only reveals his CIA credential to his peers. The negotiation starts by Alice requesting a document from Bob. The credentials and policies are given in Figure 8(a) in ATNL. Figure 8(b) depicts the TTG of this example where a policy cycle can be found in nodes 3, 4, 5, and 6. In

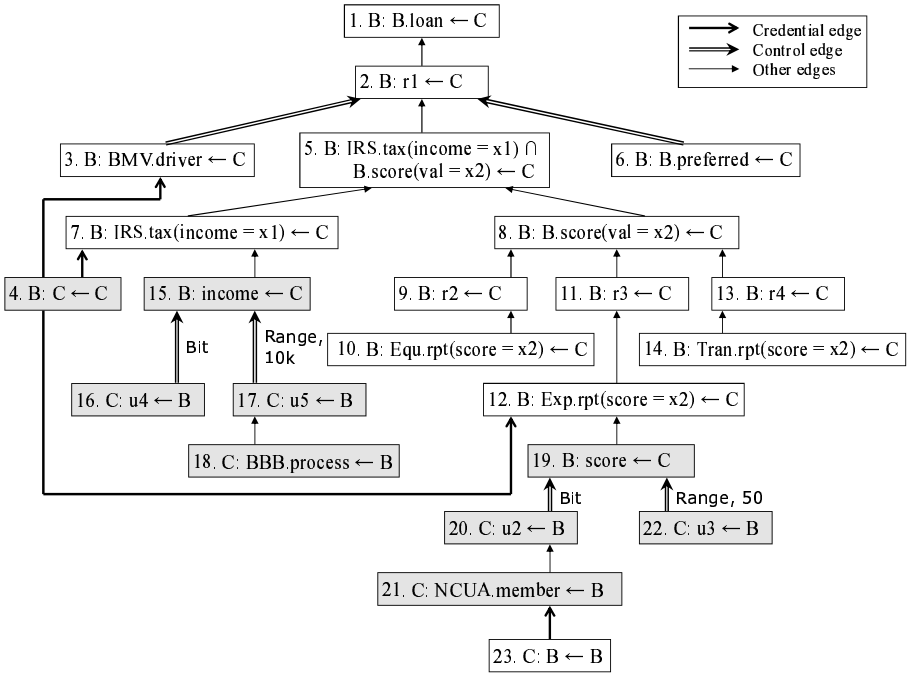

Fig. 7. Final TTG for Example 2. In this figure, ← denotes the symbol ⇜, B denotes Bank, and C denotes Carol. The white nodes are created by Bank and the grey nodes are created by Carol.

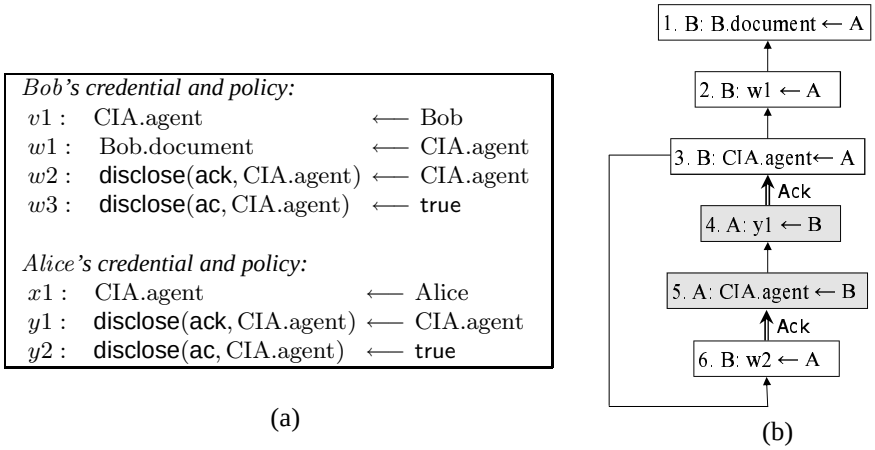

Fig. 8. (a) The credentials and policies for Example 3. (b) TTG for Example 3. In this figure, ← denotes the symbol ⇜, A denotes Alice, and B denotes Bob. The white nodes are created by Bob and the grey nodes are created by Alice.

the ETTG protocol, Bob first creates notes 1, 2, and 3. Bob wants to see whether Alice is a CIA agent. Alice has an Ack policy for her CIA credential and she only reveals her CIA credential to her peers, thus Alice creates nodes 4 and 5. Bob creates node 6 as his

CIA credential has an Ack policy $(w2)$, then creates an edge between node 3 and node 6 by expanding the Ack policy. At this point, without using the cryptographic tools listed in Section 3, the negotiation would fail because none of the nodes in the cycle could be satisfied. To break this policy cycle, Alice and Bob can run an OSBE protocol [Li et al. 2003] in which Bob delivers an envelope to Alice with the property that Alice can open the envelope if she has a CIA agent credential. This envelope contains Bob's CIA agent credential. Since Alice is a CIA agent, she can successfully open the envelope and see Bob's CIA credential. Therefore, Alice can mark node 5 satisfied, and can reveal her CIA credential to Bob. In the end, node 1 becomes satisfied and the negotiation succeeds. If Bob does not have a CIA agent credential, he would still engage in an OSBE protocol, but sending a random string in the envelope. In this way, Bob can ensure that Alice is unable to observe any behavioral characteristic indicating whether Bob has a CIA credential unless Alice satisfies Bob's Ack policy and is therefore authorized for the information.

Bob's credentials, attributes, and policies:

$v1:$ $\mathrm{CIA.agent}(\mathrm{level} = \mathsf{commit}(3)) \longleftarrow \mathrm{Bob}$

$v2:$ $\mathrm{FBI.agent} \longleftarrow \mathrm{Bob}$

$v3:$ $\mathrm{level} = 3 :: \mathrm{CIA.agent}(\mathrm{level}) :: \mathsf{sensitive}$

$w1:$ $\mathsf{disclose}(\mathsf{ack}, \mathrm{CIA.agent}) \longleftarrow \mathrm{CIA.agent}(\mathrm{level} = x)\ ;\ x \geq 2$

$w2:$ $\mathsf{disclose}(\mathsf{ac}, \mathrm{CIA.agent}) \longleftarrow \mathsf{true}$

$w3:$ $\mathsf{disclose}(\mathsf{full}, \mathrm{level}) \longleftarrow \mathrm{FBI.agent}$

$w4:$ $\mathsf{disclose}(\mathsf{ac}, \mathrm{FBI.agent}) \longleftarrow \mathsf{true}$

Alice's credentials, attributes, and policies:

$x1:$ $\mathrm{CIA.agent}(\mathrm{level} = \mathsf{commit}(4)) \longleftarrow \mathrm{Alice}$

$x2:$ $\mathrm{FBI.agent} \longleftarrow \mathrm{Alice}$

$x3:$ $\mathrm{level} = 4 :: \mathrm{CIA.agent}(\mathrm{level}) :: \mathsf{sensitive}$

$y1:$ $\mathsf{disclose}(\mathsf{ack}, \mathrm{CIA.agent}) \longleftarrow \mathrm{CIA.agent}\ ;\ x \geq 2$

$y2:$ $\mathsf{disclose}(\mathsf{ac}, \mathrm{CIA.agent}) \longleftarrow \mathsf{true}$

$y3:$ $\mathsf{disclose}(\mathsf{full}, \mathrm{level}) \longleftarrow \mathrm{FBI.agent}$

$y4:$ $\mathsf{disclose}(\mathsf{ac}, \mathrm{FBI.agent}) \longleftarrow \mathsf{true}$

Fig. 9. The credentials and policies for Example 4, illustrating that not every cycle can be broken.

EXAMPLE 4. We now give an example of a policy cycle that cannot be broken by using TTG. The corresponding credentials and policies in this example are given in Figure 9. Observe that there is a policy cycle between Alice and Bob's Ack policies. Recall that in the ETTG protocol, the attribute disclosure policies can be added to the TTG only if the corresponding credential has been disclosed. This is because, unlike Ack policies, attribute policies may differ from one negotiator to another, and are presumed to be specified by people who actually have the credential. Thus disclosing an attribute policy can strongly suggest that the negotiator has the credential, which should not be done unless the opponent satisfies the associated Ack poliy. However, in the example, when Bob sends to Alice an oblivious envelope containing proof that his CIA credential satisfies Alice's policy, Alice cannot update the TTG to reflect this even though she can open the envelope. This is because in doing so Alice would reveal fact that her level is greater than or equal to 2,

which Bob is not authorized to know because of Alice's attribute policy. Therefore, the policy cycle cannot be broken principally because Alice's full policy for the level attribute cannot be revealed to Bob unless Alice knows he satisfies her Ack policy. One would wish Bob could be told he should include his FBI credential in the oblivious envelope, which the attribute policy would tell him. But if Alice disclosed the attribute policy, this would enable Bob to infer that Alice has the CIA credential even if Bob were unauthorized for this information. Thus it is unsafe for Alice to provide the attribute policy before the oblivious envelope is sent and therefore sending the oblivious envelope is not sufficient to break the impass.

6.2 Breaking Policy Cycles Using ETTG

Detecting a policy cycle is easy—we can use any of the existing graph cycle detection algorithms. However, deciding when and how to break a policy cycle is not a trivial task. When a policy cycle is detected, we might not be able to break it immediately. For example, if A depends on B and C, and B depends on A. Even if we detect a cycle between A and B, we cannot break it until C has been satisfied. Another example is that two policy cycles may be strongly connected, in which case we have to break two cycles simultaneously. Our strategy to break policy cycles is that we (1) detect policy cycles, then (2) analyze the cycles to make sure they can be broken, finally (3) use OSBE and OCBE protocols to break the policy cycles. We next present a more complex example to illustrate how policy cycles can be broken.

EXAMPLE 5. This example illustrates the usage of properties 1, 2, 4, and 5 (oblivious usage of credentials and attributes) described in Section 3. Suppose Bob, a CIA agent, has a secret document to which access is allowed by CIA agents only. Bob has a security clearance certificate from Gov with his security clearance level committed in it. Bob can show his CIA agent credential only to his peers, and can reveal his security clearance level only to those whose level is greater than or equal to 3. Similarly, Alice has a CIA agent credential and a security clearance certificate with certain disclosure policies. Alice shows her CIA agent credential only to CIA agents with security level greater than or equal to 2. She discloses her security level only to CIA agents. See Figure 10 for the description of these credentials and policies in ATNL. When Alice wants to access Bob's document, they engage in the ETTG protocol and build a TTG as depicted in Figure 11(a).

There are two policy cycles in the TTG, one cycle has nodes 3, 4, 5, 6, and 8, the other cycle has nodes 3, 4, 5, 7, 10, 11, 12, 14, 15, 6, and 8. Without breaking the policy cycles, the negotiation between Alice and Bob would fail, because neither Alice nor Bob can update the TTG any more. As the two policy cycles share common nodes, we cannot break them separately. See Figure 11(b) for the dependency relation between Alice and Bob's attributes. To break the policy cycles, Alice and Bob run an OSBE protocol in which Bob delivers an envelope to Alice with the property that Alice can open the envelope if she has a CIA agent credential. This envelope contains Bob's CIA agent credential. In the mean time, they run an OCBE protocol in which Bob delivers another envelope to Alice such that Alice can open the envelope if and only if her security level is greater than 2. In the second envelope, Bob opens the commitment of his security level. Bob learns nothing from the previous interactions. After Alice opened the two envelopes, she verifies whether the received CIA credential and security level satisfy her policies. If so, she reveals her CIA agent credential and her security level to Bob. Now the policy cycles are broken.

Bob's credentials, attributes, and policies:

$v1:$	CIA.agent	$\longleftarrow$	Bob
$v2:$	Gov.secClearance(level = commit(3))	$\longleftarrow$	Bob
$v3:$	level = 3 :: Gov.secClearance(level)	::	sensitive
$w1:$	Bob.document	$\longleftarrow$	CIA.agent
$w2:$	disclose(ack, CIA.agent)	$\longleftarrow$	CIA.agent
$w3:$	disclose(full, level)	$\longleftarrow$	Gov.secClearance(level = x) ; $x \geq 3$
$w4:$	disclose(ac, CIA.agent)	$\longleftarrow$	true
$w5:$	disclose(ac, Gov.secClearance)	$\longleftarrow$	true

Alice's credentials, attributes, and policies:

$x1:$	CIA.agent	$\longleftarrow$	Alice
$x2:$	Gov.secClearance(level = commit(4))	$\longleftarrow$	Alice
$x3:$	level = 4 :: Gov.secClearance(level)	::	sensitive
$y1:$	disclose(ack, CIA.agent)	$\longleftarrow$	CIA.agent
			$\cap$ Gov.secClearance(level = x) ; $x \geq 2$
$y2:$	disclose(full, level)	$\longleftarrow$	CIA.agent
$y3:$	disclose(ac, CIA.agent)	$\longleftarrow$	true
$y4:$	disclose(ac, Gov.secClearance)	$\longleftarrow$	true

Fig. 10. The credentials and policies for Example 5

7. DISCUSSIONS ON ATNL

In most previous ATN protocols, the only way to satisfy a policy is by sending the credentials that document the attributes needed to satisfy the policy. In this paper, we view a credential as a structured object and allow the use of cryptographic protocols such as zero-knowledge proof protocols. This leads to two different degrees in which one can satisfy a policy. For example, when Alice uses a zero-knowledge proof protocol to prove to Bob that she has a credential, Bob is convinced himself, but Bob is unable to convince any third party using the record of the communication. We call this a *non-repeatable proof*. On the other hand, if Alice hands over her credentials to Bob, and Bob stores the credentials, then Bob is able to produce the proof that Alice indeed has the required attributes to convince other parties, for example, during an audit. We call such *a repeatable proof*. Note that a repeatable proof does not require sending the exact bit-stream of a credential. A noninteractive zero-knowledge proof of possession of a credential also serves the purpose. However, sending a credential is the most efficient way of proving possession of it, and there appears to be no reason to use more expensive ways of providing a repeatable proof of possessing a credential.

In general, a non-repeatable proof is more expensive than a repeatable proof; however, it has privacy advantages, since the proof cannot be used to convince any other party. In this paper we take the approach that all policies can be satisfied by non-repeatable proofs. Our approach can be extended to deal with the situation that certain policies may require repeatable proofs. For example, ATNL can be extended so that when a role appears in the body of policy rule, the policy author can specify whether a repeatable proof is needed.

On the credential owner's side, there are three levels of revealing possession of an attribute: (1) one can behave in a way that suggests one has or doesn't have the attribute, but the other party cannot be certain (2) one can provide a non-repeatable proof that one has credentials documenting the attribute, (3) one can provide a repeatable proof for possession

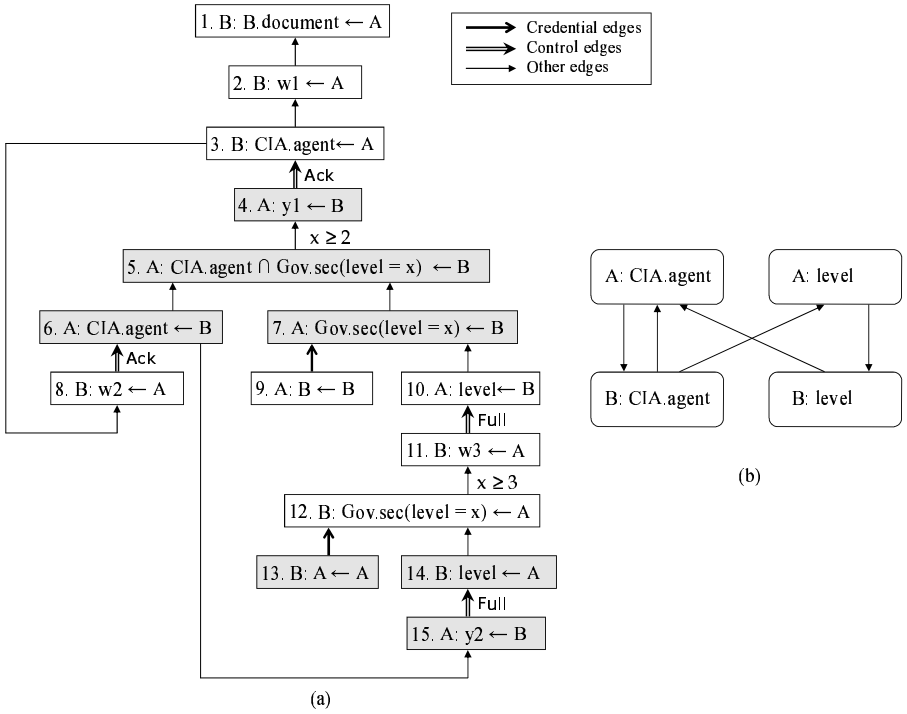

Fig. 11. (a) Final TTG for the Example 5. In this figure, ← denotes the symbol ↞, A denotes Alice, and B denotes Bob. The white nodes are created by Bob and the grey nodes are created by Alice. (b) Disclosure dependency graph for Alice's and Bob's sensitive attributes.

of the attribute. One can use different policies to protect different level of revelation. The approach taken in this paper uses two kinds of policies: Ack policies govern (1) and (2), and AC policies govern (3). That is, before the Ack policy governing an attribute has been satisfied, one has to behave the same way whether one has the attribute or not; after the Ack policy has been satisfied, one can provide a non-repeatable proof of possession. An AC policy governs the release of the credentials documenting the attribute, which constitutes a repeatable proof.

Other designs are also possible. The simplest one is to have only one kind of policy that govern all three levels of revealment. One problem with this design is that the policies of different parties about the same attribute may be quite different, and the differences may hint whether one possesses an attribute or not. Another design is to have Ack policies govern (1), and AC policies govern (2) and (3). In this design, a policy cycle involving two Ack policies cannot be broken, for the following reasons. Before the Ack policy governing an attribute is satisfied, one cannot serve the AC policy, as the fact of having an AC policy implies possession of the attribute. Even if one runs an OSBE protocol, one cannot send in the envelope a proof that one has the attribute as doing so would violate the AC policy in general. Opening the envelope does not enable the receiver to confirm anything about the other party's attributes. Therefore, one cannot make progress in the negotiation.

8. CONCLUSION

We have introduced a framework for ATN that supports the combined use of several cryptographic credential schemes and protocols that have been previously introduced piecemeal to provide capabilities that are useful in various negotiation scenarios. Our framework enables these various schemes to be combined flexibly and synergistically, on the fly as the need arises. The framework has two key components: ATNL, a policy language that enables negotiators to specify authorization requirements that must be met by an opponent to receive various amounts of information about certified attributes and the credentials that contain it; ETTG, an ATN protocol that organizes negotiation objectives and the use of cryptographic techniques to meet those objectives. We have shown several examples that illustrate how our framework enables negotiations to succeed that would not were they conducted using traditional ATN techniques.

ACKNOWLEDGMENTS

This work is supported by NSF IIS-0430274, NSF CCR-0325951, and sponsors of CERIAS. We thank the anonymous reviewers for their helpful comments.

REFERENCES

BAGGA, W. AND MOLVA, R. 2005. Policy-based cryptography and applications. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*.

BALFANZ, D., DURFEE, G., SHANKAR, N., SMETTERS, D., STADDON, J., AND WONG, H.-C. 2003. Secret handshakes from pairing-based key agreements. In *Proceedings of the IEEE Symposium on Security and Privacy*. 180–196.

BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. D. 1999. The KeyNote trust-management system, version 2. IETF RFC 2704.

BLAZE, M., FEIGENBAUM, J., AND LACY, J. 1996. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 164–173.

BOEYEN, S., HOWES, T., AND RICHARD, P. 1999. Internet X.509 public key infrastructure LDAPc2 schema. IETF RFC 2587.

BONATTI, P. AND SAMARATI, P. 2000. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*. ACM Press, 134–143.

BOUDOT, F. 2000. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology — EUROCRYPT '00*. LNCS, vol. 1807. 431–444.

BRADSHAW, R., HOLT, J., AND SEAMONS, K. 2004. Concealing complex policies with hidden credentials. In *Proceedings of 11th ACM Conference on Computer and Communications Security*. 146–157.

BRANDS, S. A. 2000. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press.

CAMENISCH, J. AND HERREWEGHEN, E. V. 2002. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (18–22). ACM, 21–30.

CAMENISCH, J. AND LYSYANSKAYA, A. 2001. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology — EUROCRYPT '01*. LNCS, vol. 2045. Springer, 93–118.

CASTELLUCCIA, C., JARECKI, S., AND TSUDIK, G. 2004. Secret handshakes from CA-oblivious encryption. In *Advances in Cryptology — ASIACRYPT '04*. 293–307.

CHAUM, D. 1985. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM 28*, 10, 1030–1044.

CLARKE, D., ELIEN, J.-E., ELLISON, C., FREDETTE, M., MORCOS, A., AND RIVEST, R. L. 2001. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security 9*, 4, 285–322.

CRAMER, R. AND DAMGÅRD, I. 1998. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology — CRYPTO '98*. LNCS, vol. 1462. Springer, 424–441.

CRAMER, R., FRANKLIN, M. K., SCHOENMAKERS, B., AND YUNG, M. 1996. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology — EUROCRYPT '96*. LNCS, vol. 1070. Springer, 72–83.

DAMGÅRD, I. AND FUJISAKI, E. 2002. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology — ASIACRYPT '02* (1–5). LNCS, vol. 2501. Springer, 125–142.

DETREVILLE, J. 2002. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 105–113.

DODIS, Y., KIAYIAS, A., NICOLOSI, A., AND SHOUP, V. 2004. Anonymous identification in ad hoc groups. In *Advances in Cryptology — EUROCRYPT '04*. 609–626.

DURFEE, G. AND FRANKLIN, M. 2000. Distribution chain security. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*. ACM Press, 63–70.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999. SPKI certificate theory. IETF RFC 2693.

FRIKKEN, K. B., ATALLAH, M. J., AND LI, J. 2004. Hidden access control policies with hidden credentials. In *Proceedings of the 3rd ACM Workshop on Privacy in the Electronic Society*.

FRIKKEN, K. B., LI, J., AND ATALLAH, M. J. 2006. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *Proceedings of 13th Network and Distributed System Security Symposium*. 157–172.

GUNTER, C. A. AND JIM, T. 2000. Policy-directed certificate retrieval. *Software: Practice & Experience 30*, 15 (Sept.), 1609–1640.

HESS, A., JACOBSON, J., MILLS, H., WAMSLEY, R., SEAMONS, K. E., AND SMITH, B. 2002. Advanced client/server authentication in TLS. In *Network and Distributed System Security Symposium*. 203–214.

HOLT, J. E., BRADSHAW, R. W., SEAMONS, K. E., AND ORMAN, H. 2003. Hidden credentials. In *Proceedings of the 2nd ACM Workshop on Privacy in the Electronic Society*. 1–8.

HOUSLEY, R., FORD, W., POLK, T., AND SOLO, D. 1999. Internet X.509 public key infrastructure certificate and CRL profile. IETF RFC 2459.

IRWIN, K. AND YU, T. 2005. Preventing attribute information leakage in automated trust negotiation. In *Proceedings of the 12th ACM conference on Computer and Communications Security*. 36–45.

JIM, T. 2001. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 106–115.

LI, J. AND LI, N. 2005a. OACerts: Oblivious attribute certificates. In *Proceedings of the 3rd Conference on Applied Cryptography and Network Security*. LNCS, vol. 3531. Springer, 301–317.

LI, J. AND LI, N. 2005b. Policy-hiding access control in open environment. In *Proceedings of the 24nd ACM Symposium on Principles of Distributed Computing*. ACM Press, 29–38.

LI, N., DU, W., AND BONEH, D. 2003. Oblivious signature-based envelope. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*. ACM Press, 182–189.

LI, N., GROSOF, B. N., AND FEIGENBAUM, J. 2003. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security 6*, 1 (Feb.), 128–171.

LI, N. AND MITCHELL, J. C. 2003. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*. Number 2562 in LNCS. Springer, 58–73.

LI, N., MITCHELL, J. C., AND WINSBOROUGH, W. H. 2002. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 114–130.

LI, N., WINSBOROUGH, W. H., AND MITCHELL, J. C. 2003. Distributed credential chain discovery in trust management. *Journal of Computer Security 11*, 1 (Feb.), 35–86.

LYSYANSKAYA, A., RIVEST, R. L., SAHAI, A., AND WOLF, S. 1999. Pseudonym systems. In *Proceedings of the 6th Workshop on Selected Areas in Cryptography*. LNCS, vol. 1758. Springer, 184–199.

PEDERSEN, T. P. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*. LNCS, vol. 576. Springer, 129–140.

RIVEST, R. L. AND LAMPSON, B. 1996. SDSI — A simple distributed security infrastructure.

SEAMONS, K. E., WINSLETT, M., AND YU, T. 2001. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Symposium on Network and Distributed System Security*.

SEAMONS, K. E., WINSLETT, M., YU, T., YU, L., AND JARVIS, R. 2002. Protecting privacy during on-line trust negotiation. In *2nd Workshop on Privacy Enhancing Technologies*. Springer-Verlag.

SMART, N. 2003. Access control using pairing based cryptography. In *Proceedings of the Cryptographers' Track at the RSA Conference 2003*. Springer-Verlag LNCS 2612, 111–121.

WINSBOROUGH, W. H. AND LI, N. 2002a. Protecting sensitive attributes in automated trust negotiation. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*. ACM Press, 41–51.

WINSBOROUGH, W. H. AND LI, N. 2002b. Towards practical automated trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society Press, 92–103.

WINSBOROUGH, W. H. AND LI, N. 2004. Safety in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy*. 147–160.

WINSBOROUGH, W. H., SEAMONS, K. E., AND JONES, V. E. 2000. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*. Vol. I. IEEE Press, 88–102.

WINSLETT, M., YU, T., SEAMONS, K. E., HESS, A., JACOBSON, J., JARVIS, R., SMITH, B., AND YU, L. 2002. Negotiating trust on the web. *IEEE Internet Computing 6*, 6 (November/December), 30–37.

YU, T. AND WINSLETT, M. 2003a. Policy migration for sensitive credentials in trust negotiation. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*. ACM Press, 9–20.

YU, T. AND WINSLETT, M. 2003b. Unified scheme for resource protection in automated trust negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 110–122.

YU, T., WINSLETT, M., AND SEAMONS, K. E. 2003. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security 6*, 1 (Feb.), 1–42.

Received February 2006