CERIAS Tech Report 2005-59

AUTOMATED TRUST NEGOTIATION USING CRYPTOGRAPHIC CREDENTIALS

by Jiangtao Li and Ninghui Li and William H. Winsborough

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

Automated Trust Negotiation Using Cryptographic Credentials

Jiangtao Li [1] Ninghui Li [1] William H. Winsborough [2]
[1]Dept. of Computer Science, Purdue University, {jtli,ninghui}@cs.purdue.edu
[2]Dept. of Computer Science, Univ. of Texas at San Antonio, wwinsborough@acm.org

Abstract

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 fully disclose the attributes, 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.

1 Introduction

In automated trust negotiation (ATN) [16, 27, 28, 29, 30, 31, 32, 33, 36, 37], 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. Viewing a credential as a black-box 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 and c_2 can be disclosed before the other is, 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 that 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 would be 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 [21], hidden credentials [6, 17], and secret handshakes [2] can be used to address the policy cycle problem. Oblivious Attribute Certificates (OACerts) [19], private credentials [7], and anonymous credentials [8, 9, 10, 25] 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) [20] 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 [3, 18]) as well as OACerts, hidden credentials, and anonymous credentials.
- In addition to attribute information stored in credentials, the ATN framework supports also attribute information that is not certified. For example, oftentimes one is asked to provide a phone number in an online transaction, and the phone number may 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 [22, 23, 24].
- 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 [30, 31].

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 conclude our paper in Section 6.

2 Related Work

Automated trust negotiation was introduced by Winsborough et al. [32], who 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. [37] 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. [27] and Yu and Winslett [36] studied the problem of protecting contents of policies as well as credentials. On the aspect of system architecture for trust negotiation, Hess et al. [16] 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. [34] introduced the TrustBuilder architecture for trust negotiation systems. The problem of leaking attribute information was recognized by Winsborough and Li [30], Seamons et al. [28], and Yu and Winslett [35]. Winsborough and Li [29, 30, 31] introduced the notion of acknowledgement policies to protect this information and provided a formal notion of safety against illegal attribute information leakage. Bonatti and Samarati [4] proposed a framework for regulating service access and information release on the web. Their framework supports both certified attributes and uncertified attributes.

Recent work on using cryptographic protocols for ATN includes hidden credentials [6, 15, 17], secret handshakes [2], oblivious signature based envelope [21], oblivious commitment based envelope [19], certified input private policy evaluation [20], and policy-based cryptography [1]. While these 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 [19], private credentials [7], and anonymous credentials [8, 9, 10, 25]. 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 [7], anonymous credentials [8, 9, 10, 25], and OACerts [19], 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. Therefore, a credential holder can disclose her credentials without revealing the attribute values in them. 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 of attributes satisfying a policy:* A credential holder can use zero-knowledge proof protocols [5, 11, 12, 14] to prove that her attributes satisfy some property 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) [21], hidden credentials [17], or secret handshakes [2]. 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 Envelop (OCBE) [19]. 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 whether the values satisfy the policy or not.

6. *Certified input private policy evaluation (CIPPE):* In CIPPE [20], 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 revealing the attribute values 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 [7] and the anonymous credentials [8, 9, 10, 25], 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 we believe they are most related to trust negotiation. 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 [22, 23, 24]. 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:

$\ell 1:$ $\mathrm{SBA.businessLicense} \longleftarrow \mathrm{BookSt}$

$\ell 2:$ $\mathrm{BBB.goodSecProcess} \longleftarrow \mathrm{BookSt}$

BookSt's policies:

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

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

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

Figure 1: The credentials and policies of BookSt

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 science, under 21, and willing to provide a phone number. Since the discount is a resource, the head of this policy, $\mathrm{BookSt.discount}(\mathrm{phoneNum} = x_3)$, defines a part of the application interface provided by the ATN system using this policy; the parameter $\mathrm{phoneNum}$ is made available to the application through this interface. That is, the application will issue a query to determine whether the requester satisfies $\mathrm{BookSt.discount}(\mathrm{phoneNum} = x_3)$, 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: $\text{StateU.student}(\text{program} = x_1) \cap \text{BookSt.DoB}(\text{val} = x_2) \cap \textsf{Any}.\text{phoneNum}(\text{val} \Rightarrow x_3)$

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

Part 1 describes the role requirement of the policy and consists of the intersection of 3 roles. To satisfy the role $\text{StateU.student}(\text{program} = x_1)$, one must provide a credential (or a credential chain) showing that one is certified by StateU to be a student; $\text{program} = x_1$ means that the value of the program field is required to satisfy additional constraints. In $\textsf{Any}.\text{phoneNum}(\text{val} \Rightarrow x_3)$, 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.

Alice's credentials:

$n1$:	StateU.student	$\longleftarrow$	CoS.student
$n2$:	CoS.student(program = 'cs', level = 'sophomore')	$\longleftarrow$	Alice
$n3$:	BMV.driverLicense(name = commit('Alice'), DoB = commit('03/07/1986'))	$\longleftarrow$	Alice

Alice's attribute declarations:

$o1$:	phoneNum	=	'(123)456-7890'	::		::	sensitive
$o2$:	DoB	=	'03/07/1986'	::	BMV.driverLicense(DoB)	::	sensitive
$o3$:	program	=	'cs'	::	CoS.student(program)	::	non-sensitive
$o4$:	level	=	'sophomore'	::	CoS.student(level)	::	non-sensitive

Alice's policies:

$p1$:	disclose(ac, CoS.student)	$\longleftarrow$	SBA.businessLicense
$p2$:	disclose(full, DoB)	$\longleftarrow$	BBB.goodSecProcess
$p3$:	disclose(full, phoneNum)	$\longleftarrow$	BBB.goodSecProcess
$p4$:	disclose(range, DoB, year)	$\longleftarrow$	true

Figure 2: The credentials and policies possessed by Alice

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 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 ($p4$).

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.

⟨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⟩ "$\longleftarrow$" ⟨prin⟩ (5)
⟨delegation-cred⟩ ::= ⟨role⟩ "$\longleftarrow$" ⟨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⟩ "$\longleftarrow$" ⟨policy-body⟩ (12)
⟨policy-body⟩ ::= ⟨p-role-req⟩ [";" ⟨p-constraint⟩] | true (13)
⟨p-role-req⟩ ::= [⟨pre-cond⟩ "!"] ⟨conj-of-p-roles⟩ (14)
⟨p-constraint⟩ ::= [⟨pre-cond⟩ "!"] ⟨constraint⟩ (15)
⟨pre-cond⟩ ::= ⟨role⟩ | "false" (16)
⟨conj-of-p-roles⟩ ::= ⟨p-role⟩ | ⟨p-role⟩ "$\cap$" ⟨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⟩ ("=" | "$\Rightarrow$") (⟨var⟩ | ⟨constant⟩) (20)

⟨policy-head⟩ ::= ⟨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)

Figure 3: Syntax of ATNL in BNF. The first two definitions ⟨list of X⟩ and ⟨set of X⟩ are macros parameterized by X. The symbol ϵ in (2) denotes the empty string. The symbols ⟨var⟩, ⟨constant⟩, and ⟨prin⟩ each represents a variable, a constant, and a principal respectively. The symbols ⟨role-name⟩, ⟨field-name⟩, and ⟨attr-name⟩ represent identifiers drawn from disjoint sets. The syntax for non-terminals ⟨commitment⟩, ⟨precision⟩, ⟨constraint⟩ are not defined here; they are explained in the text.

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.

4.2.1 Credentials and Roles

Two central concepts that ATNL takes from RT [23, 24] 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. A *role* designates a set of principals who are members of this role. Each principal has its own localized name space for roles in which it has sole authority to define roles. 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 of $A.R \longleftarrow D$, 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 of $A.R \longleftarrow B.R_1$, 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.

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 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, $\mathrm{CoS.student(program = \text{`cs'}, level = \text{`sophomore'})}$ and $\mathrm{BMV.driverLicense(name = \mathsf{commit}(\text{`Alice'}), DoB = \mathsf{commit}(\text{`03/07/1986'}))}$ 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 of $\mathsf{commit}(c)$, where c is a constant, and commit denotes the output of a commitment algorithm of a commitment scheme [13, 26][1].

If a credential is a regular certificate, such as an X.509 certificate [18], then each field in the credential takes the form of $x = c$, 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 of $x = \mathsf{commit}(c)$, 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.

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.

[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.

[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 [23, 24].

Then the attribute declaration for her DoB looks like

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

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.

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:

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

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, pre-cond-1 and pre-cond-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 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*.

A *pre-condition* (16) is defined to be a role or the keyword false. The motivation for the pre-condition 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 pre-cond-1 is satisfied (or if pre-cond-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 pre-cond-1 is satisfied, if pre-cond-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; 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 are not defined in signed credentials, but serve only to aid in defining 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.

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, the policy enforcer can simply disclose its value. When the field value linked to the attribute reference of attr is a commitment, 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. 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 define 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 a 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 $\mathsf{disclose}(\mathsf{range}, \mathrm{score}, 50)$, 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 $\mathsf{disclose}(\mathsf{range}, \mathrm{score}, 10)$ means that if the policy is satisfied, the opponent can learn that Alice's credit score is between 721 to 730.

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 [30, 31], 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, 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 $\langle V : \text{policy-id} \stackrel{?}{\twoheadleftarrow} S \rangle$, 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 $\langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \stackrel{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, S is a principal, $B_1.R_1, \ldots, B_k.R_k$ are roles, 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 $\langle V : S \stackrel{?}{\twoheadleftarrow} S \rangle$, in which V is one of the negotiators, and S is a principal. Trivial targets provide placeholders for edges in the TTG.
- An *attribute goal* takes the form $\langle V : \text{attr} \stackrel{?}{\twoheadleftarrow} S \rangle$, 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. Each target has a *satisfaction state*, which has one of three values: *satisfied*, *failed*, or *unknown*. The value is determined inductively depending on the containing TTG structure and the credentials present, as discussed below.

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 $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : e \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : B_i.R_i \overset{?}{\twoheadleftarrow} S\rangle$, where i is in $1..k$, and k is greater than 1.
- An *attribute edge* takes the form $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle V : \text{attr} \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : e \overset{?}{\twoheadleftarrow} S\rangle \leftarrowtail \langle \mathit{opp}(V) : \text{policy-id} \overset{?}{\twoheadleftarrow} V\rangle$, in which $\mathit{opp}(V)$ 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 Overview of The Extended Trust-Target Graph (ETTG) Protocol

We now sketch the ETTG protocol. Details of the ETTG protocol are given in Appendix A. We begin with an example of the ETTG protocol, then briefly discuss how to process each node in TTG, and how to handle constraints in the policies.

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 4. Alice and BookSt run the ETTG protocol as follows: As BookSt wants to see the proof of $\text{BookSt.discount} \twoheadleftarrow \text{Alice}$ 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 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 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.

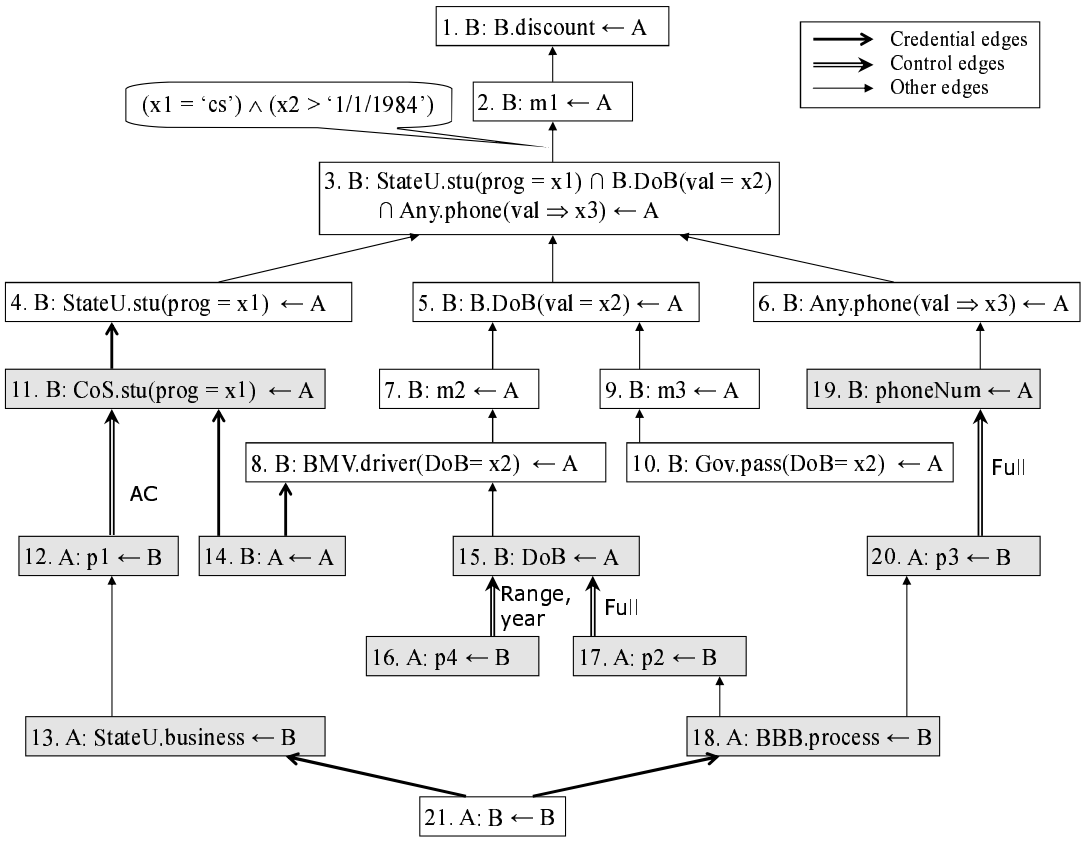

Figure 4: Final TTG for the bookstore example. In this figure, ← denotes the symbol $\twoheadleftarrow$, A denotes Alice, and B denotes BookSt. The white nodes are created by BookSt and the grey nodes are created by Alice.

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 is born in 1986. Alice's year of birth flows up from node 8 to node 3.

BookSt 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.

Node Processing in ETTG We briefly explain how each node is processed in the ETTG protocol.

1. *Role target.* Suppose the role in a role target $T = \langle V : V.R \stackrel{?}{\twoheadleftarrow} S \rangle$ is a dummy role. For each of the verifier's policies that have $V.R$ as the policy head, the verifier adds a new policy child for T. The role target is satisfied if one of its children is satisfied.

   Now suppose the role in a role target $T = \langle V : A.R \stackrel{?}{\twoheadleftarrow} S \rangle$ is not a dummy role. If the opponent of V has an Ack policy for $A.R$, he adds an attribute control child for T. Once the Ack policy (if any) has been satisfied, if

the opponent has an AC policy for $A.R$, he adds another attribute control child. After all of T's attribute control children have been satisfied, the opponent can disclose his credential for $A.R$ (if any), adds a credential edge, and marks T satisfied. If the credential disclosed is a membership credential, then for each field in $A.R$, if there is a sensitive attribute linked to the field in the opponent's attribute declarations, the opponent adds an attribute child for that attribute.

2. *Policy target.* Consider the policy target $T = \langle V : \text{policy-id} \stackrel{?}{\leftarrow} S \rangle$. Suppose the policy body associated with policy-id takes the form pre-cond-1! $B_1.R_1 \cap \cdots \cap B_k.R_k$; pre-cond-2! $\psi(x_1, \ldots, x_n)$. The verifier first adds a policy control child for pre-cond-1. Once the policy control child is satisfied, the verifier adds a policy expansion child for $B_1.R_1 \cap \cdots \cap B_k.R_k$ and a policy control child for pre-cond-2. If the policy control child for pre-cond-2 is satisfied, the verifier adds the constraint $\psi(x_1, \ldots, x_n)$ as a tag on the policy expansion edge. A policy target is satisfied if its policy expansion child is satisfied and the constraint is evaluated and satisfied. We explain how and when to evaluate a constraint below.

3. *Intersection target.* For the intersection target $T = \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \stackrel{?}{\leftarrow} S \rangle$, the verifier adds an intersection child for each role $B_i.R_i$. The intersection target is satisfied if all of its intersection children are satisfied.

4. *Attribute goal.* For the attribute target $T = \langle V : \text{attr} \stackrel{?}{\leftarrow} S \rangle$, the opponent adds an attribute policy child for each disclosure policy that contains attr in the policy head.

Constraint Handling We now explain how the constraint of a policy is evaluated. If there is a constraint tag in the policy expansion edge of the policy (*i.e.*, the constraint is revealed to the opponent), it can be evaluated as follows: Whenever a full disclosure policy or a range disclosure policy for an attribute is satisfied by the verifier, the opponent reveals the attribute information accordingly. The attribute information flows from the attribute goal to the policy expansion edge where the constraint is attached. At the policy expansion edge, when the verifier receives enough information from the opponent to determine whether or not the constraint is satisfied, he evaluates the constraint based on the attribute information received so far and outputs the result.

If there is no constraint tag in the policy expansion edge (*i.e.*, the pre-condition for the constraint has not been satisfied), the verifier can satisfy all the full disclosure policies of the attributes required in the constraint, enabling it to obtain all the attribute values from the opponent. The verifier then evaluates the constraint secretly and informs the opponent the result of the evaluation. Alternatively, the verifier can try to satisfy all the bit disclosure policies of the attributes, and then run a private policy evaluation protocol with the opponent.

5.4 Additional Examples

In this section, we give two additional examples that illustrate the ATNL language and the ETTG protocol. Example 2 deals with the scenario in which the constraint is private; example 3 illustrates how the ETTG protocol breaks the policy cycles.

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 5.

Using the ETTG protocol, BankWon and Carol can negotiate trust successfully. The final TTG of the negotiation is given in Figure 6. 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

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 $= x$)

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

Figure 5: The credentials and policies for Example 2

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. 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.

Example 3 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 the access is allowed by CIA agents only. Bob has a security clearance certificate from Gov with the security 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 security 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. And she discloses her security level only to CIA agents. See Figure 7 for the description of the credentials and policies in ATNL. When Alice wants to access Bob's document, they engage the ETTG protocol and build a TTG as depicted in Figure 8(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 8(b) for the dependency relation between Alice and Bob's attributes. To break the policy cycles, Alice and Bob run an OSBE protocol [21] 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 [19] in which Bob delivers another envelope to Alice such

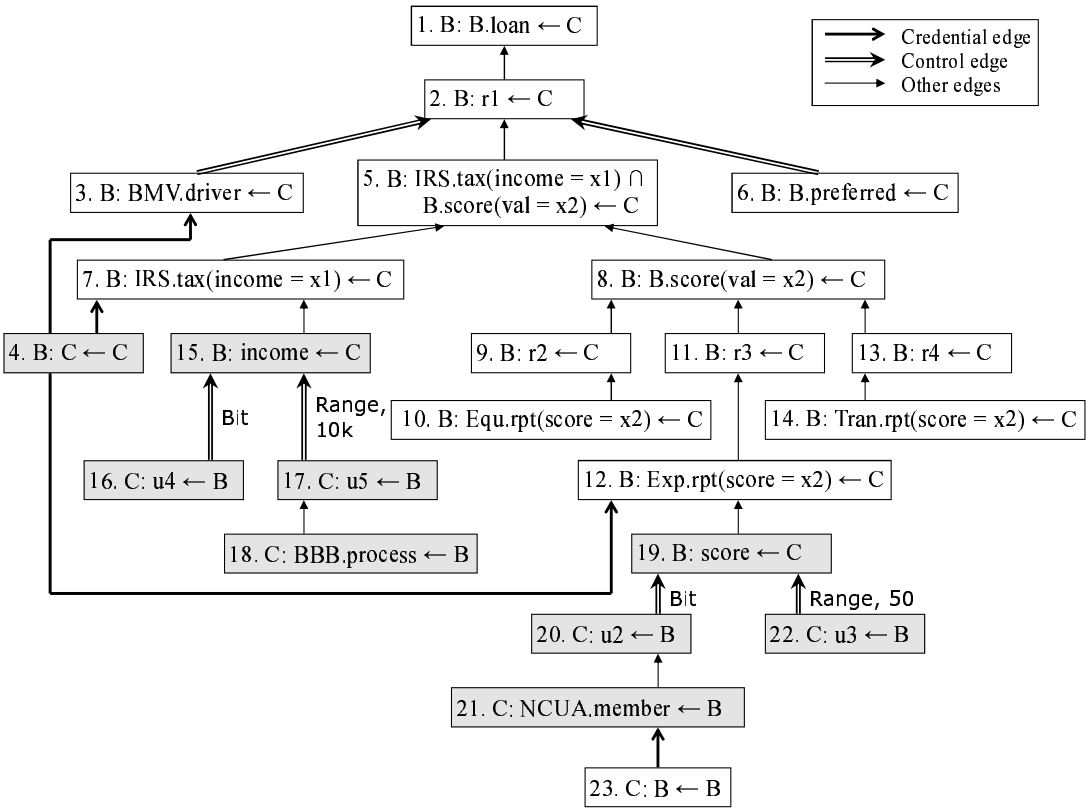

Figure 6: Final TTG for Example 2. In this figure, ← denotes the symbol ⇜, B denotes Bank, and C denotes Carol. The white nodes are created by Bank and the grey nodes are created by Carol.

Bob's credentials, attributes, and policies:

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

Alice's credentials, attributes, and policies:

$x1:$	CIA.agent	⟵	Alice
$x2:$	Gov.secClearance(level = commit(4))	⟵	Alice
$x3:$	level = 4 :: Gov.secClearance(level)	::	sensitive
$y1:$	disclose(ack, CIA.agent)	⟵	CIA.agent ∩ Gov.secClearance(level = x) ; $x \geq 2$
$y2:$	disclose(full, level)	⟵	CIA.agent

Figure 7: The credentials and policies for Example 3

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.

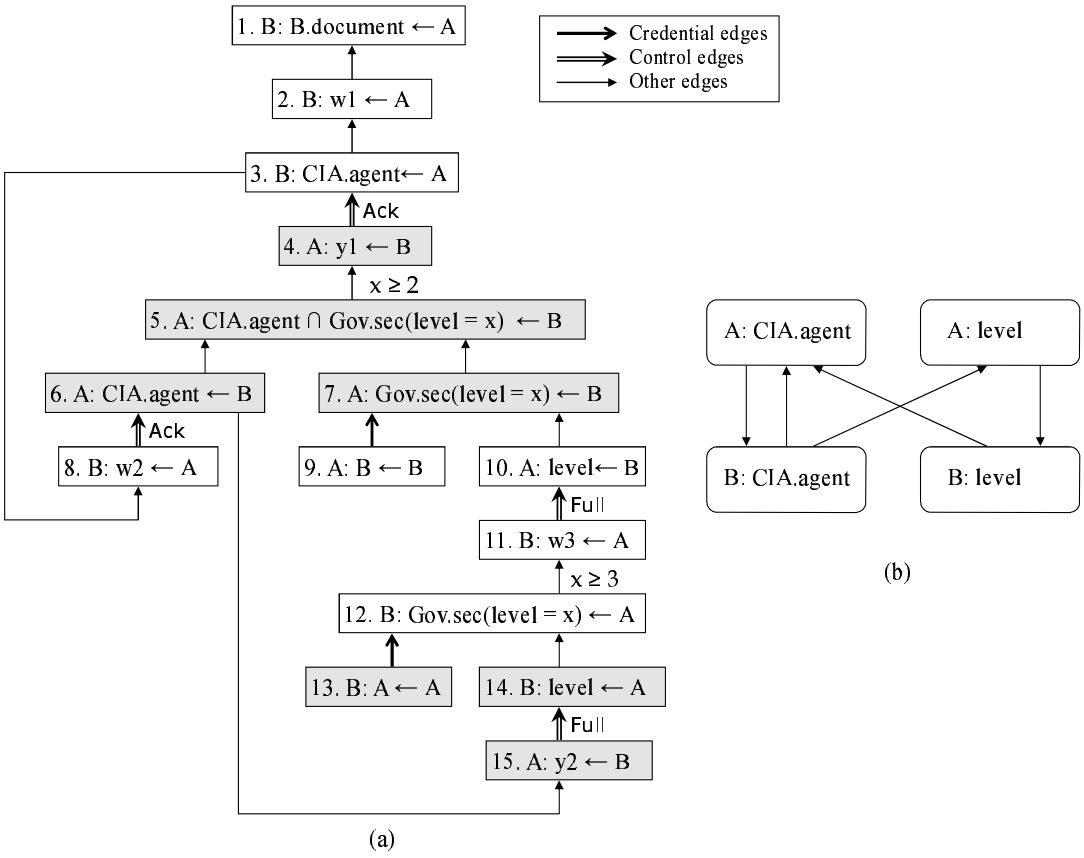

Figure 8: (a) Final TTG for the Example 3. In this figure, ← denotes the symbol ⇐, A denotes Alice, and B denotes Bob. The white nodes are created by Bob and the grey nodes are created by Alice. (b) Disclosure dependency graph for Alice's and Bob's sensitive attributes.

6 Conclusion and Future Work

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. The appendix presents the details of the process of constructing the trust target graph and other aspects of the ETTG negotiation state. In on-going research we are developing comprehensive analysis algorithms that negotiators will use to recognize all cyclic dependencies that can be resolved.

Acknowledgement

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

References

[1] W. Bagga and R. Molva. Policy-based cryptography and applications. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, Feb. 2005.

[2] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H.-C. Wong. Secret handshakes from pairing-based key agreements. In *Proceedings of the IEEE Symposium and Security and Privacy*, pages 180–196, May 2003.

[3] S. Boeyen, T. Howes, and P. Richard. Internet X.509 Public Key Infrastructure LDAPc2 Schema. IETF RFC 2587, June 1999.

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

[5] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology: EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, May 2000.

[6] R. Bradshaw, J. Holt, and K. Seamons. Concealing complex policies with hidden credentials. In *Proceedings of 11th ACM Conference on Computer and Communications Security*, Oct. 2004.

[7] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Aug. 2000.

[8] J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02*, pages 21–30. ACM, nov 2002.

[9] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology: EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.

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

[11] R. Cramer and I. Damgård. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 1998.

[12] R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology: EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.

[13] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology: ASIACRYPT '02*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, Dec. 2002.

[14] G. Durfee and M. Franklin. Distribution chain security. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 63–70. ACM Press, 2000.

[15] K. B. Frikken, M. J. Atallah, and J. Li. Hidden access control policies with hidden credentials. In *Proceedings of the 3rd ACM Workshop on Privacy in the Electronic Society*, Oct. 2004.

[16] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith. Advanced client/server authentication in TLS. In *Network and Distributed System Security Symposium*, pages 203–214, Feb. 2002.

[17] J. E. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *Proceedings of the 2nd ACM Workshop on Privacy in the Electronic Society*, Oct. 2003.

[18] R. Housley, W. Ford, T. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF RFC 2459, Jan. 1999.

[19] J. Li and N. Li. OACerts: Oblivious attribute certificates. In *Proceedings of the 3rd Conference on Applied Cryptography and Network Security (ACNS)*, volume 3531 of *Lecture Notes in Computer Science*. Springer, June 2005.

[20] J. Li and N. Li. Policy-hiding access control in open environment. In *Proceedings of the 24nd ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, July 2005.

[21] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, July 2003.

[22] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, pages 58–73. Springer, Jan. 2003.

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

[24] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, Feb. 2003.

[25] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography, 6th Annual International Workshop, SAC '99*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.

[26] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[27] K. E. Seamons, M. Winslett, and T. Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS'01)*, February 2001.

[28] K. E. Seamons, M. Winslett, T. Yu, L. Yu, and R. Jarvis. Protecting privacy during on-line trust negotiation. In *2nd Workshop on Privacy Enhancing Technologies*. Springer-Verlag, Apr. 2002.

[29] W. H. Winsborough and N. Li. Protecting sensitive attributes in automated trust negotiation. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, pages 41–51. ACM Press, Nov. 2002.

[30] W. H. Winsborough and N. Li. Towards practical automated trust negotiation. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 92–103. IEEE Computer Society Press, June 2002.

[31] W. H. Winsborough and N. Li. Safety in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 147–160, May 2004.

[32] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88–102. IEEE Press, Jan. 2000.

[33] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[34] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[35] T. Yu and M. Winslett. Policy migration for sensitive credentials in trust negotiation. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, pages 9–20. ACM Press, Oct. 2003.

[36] T. Yu and M. Winslett. Unified scheme for resource protection in automated trust negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 110–122. IEEE Computer Society Press, May 2003.

[37] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, Feb. 2003.

A Details of the ETTG protocol

In this Appendix, we present the details of the ETTG protocol. We first describe how states are propagated in TTG, then describe the messages in the protocol, finally present how nodes are processed.

A.1 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 field in the roles of a role node or an intersection node, there is a *field state*. Each field state has multiple entries depending on how many disclosure policies this attribute has. For example, if an attribute has a full disclosure policy and a bit disclosure policy, then the field state has two entries, one for the full policy, the other for the bit policy. The entry in the field state for the bit policy takes either true or false value. The entry for other policies can take arbitrary values. Each attribute has an *attribute state*. An attribute state has multiple entries depending on how many disclosure policies this attribute has. Each entry can be one of the two values: true or false. A true value means the corresponding policy in that entry has been satisfied. A false value means the corresponding policy has not been satisfied.

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

Satisfaction state The trust target satisfaction state is determined as follows:

1. *Role target.* The initial satisfaction state a role target is unknown. It becomes fully satisfied when one of its credential children or one of its policy children is fully 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 empty (the full value of the field has been disclosed). It becomes failed when it is full processed and it has no child, or all of its children are failed, or there exists some field in the role with the $\Rightarrow$ symbol whose value has not been disclosed.
2. *Policy target.* Let policy-id be the policy identifier in this policy target. If the policy body corresponding to policy-id is true, then the satisfaction state of this target is fully satisfied. Otherwise, the initial satisfaction state of a policy target is unknown.
   (a) If there is no constraint in the policy corresponding to policy-id, the satisfaction state of the policy target becomes satisfied when it is full processed and its policy expansion child is satisfied. It becomes failed when 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 policy-id, the satisfaction state of the policy target becomes satisfied when it is full processed, its policy expansion child is satisfied, and the constraint is evaluated and also satisfied. If the constraint has been revealed (*i.e.*, there exists a policy control child for the constraint), 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 using the private policy evaluation, or the full value of each variable in the policy has been disclosed. It becomes failed when it has no policy expansion child, or its policy expansion child is failed, or the constraint cannot be evaluated, 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 fully satisfied.

Field state Informally speaking, the values of field state flow up from child to parent. There is no entry when a field state is initialized. The values of the field state are copied from one of its children or its grandchildren, if the values are available. If the current node has a non-delegation credential child and the corresponding credential is a regular credential (such as X.509 certificate), then write the full value of the field to the full entry. Otherwise, if the current node has an attribute child, depends on the attribute state of the attribute goal, the opponent reveals the attribute value according. For example, if the full entry in the attribute child is true, then the opponent reveals the full value of the field and write the value in the full entry of the field state. If the bit entry in the attribute state of the attribute child is true, the bit entry in the field state is set to be true also. 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.

Attribute state Let attr be the attribute name in the attribute goal. If there is a disclosure policy for attr, we add an entry in the attribute state. The initial value for that entry is false. If the satisfaction state of the attribute control child corresponding to the disclosure policy becomes satisfied, the value of the entry becomes true.

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.

A.2 Messages in the Protocol

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

- Initialize the TTG to contain a given primary 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 verified processed. These operations construct a connected graph. Satisfaction state of trust targets, field state of fields in trust targets, and attribute state of attribute goals are not transmitted in messages; instead, each negotiation party infers them independently.

A.3 Node Processing

Previously we described the TTG negotiation protocol, in which two negotiators exchange update messages. The protocol defines what updates are legal, and the receiver of a message can verify that the updates in the message is legal. 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.

A.3.1 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.
- For a role target, $\langle K_V : K.r \stackrel{?}{\leftarrow} K_S \rangle$, if $K.r$ is a dummy role, the target is opponent-processed, which means that the opponent cannot process it; otherwise, it is verifier-processed.
- A policy target is initially opponent-processed.
- An intersection target is initially opponent-processed.
- An attribute goal is initially verifier-processed.

A.3.2 Verifier-Side Processing

We now describe how a negotiator V process a node when it is the verifier of the node. These rules apply to nodes that are not yet marked verifier-processed.

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 and $\mathrm{policy\text{-}id}$ is the corresponding policy identifier, V can add a policy edge $T \hookleftarrow \langle V : \mathrm{policy\text{-}id} \stackrel{?}{\leftarrow} S \rangle$.
(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.

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 $\mathrm{policy\text{-}id}$. If $\mathrm{pre\text{-}cond\text{-}1}$ is a role, say $A_1.R_1$, V can add a policy control edge $T \hookleftarrow \langle V : A_1.R_1 \stackrel{?}{\leftarrow} S \rangle$.
(b) After (a) is done and $\langle V : A_1.R_1 \stackrel{?}{\leftarrow} S \rangle$ is satisfied, or there is no pre-condition for the intersection, V can add a policy expansion edge $T \hookleftarrow \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \stackrel{?}{\leftarrow} S \rangle$.
(c) Suppose there is a constraint for this policy. If $\mathrm{pre\text{-}cond\text{-}2}$ is a role, say $A_2.R_2$, V can add a policy control edge $T \hookleftarrow \langle V : A_2.R_2 \stackrel{?}{\leftarrow} S \rangle$.
(d) After (c) is done and $\langle V : A_2.R_2 \stackrel{?}{\leftarrow} S \rangle$ 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 (b) is *done* if there is no constraint for the policy.

3. Processing $T = \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \stackrel{?}{\leftarrow} S \rangle$
(a) V can add the k intersection edges, $T \leftarrowtail \langle V : B_i.R_i \stackrel{?}{\leftarrow} K_S \rangle$, $1 \leq i \leq k$
(b) V can mark T verifier-processed only after (a) is done.

A.3.3 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 $T = \langle V : A.R \stackrel{?}{\leftarrow} S \rangle$
(a) If there exists a policy statement with head $\mathsf{disclose}(\mathsf{ack}, A.R)$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{ack-id} \stackrel{?}{\leftarrow} V \rangle$, where ack-id is the policy identifier for the ack policy.
(b) After (a) is done and $\langle S : \text{ack-id} \stackrel{?}{\leftarrow} V \rangle$ 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 $\mathsf{disclose}(\mathsf{ac}, A.R)$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{ac-id} \stackrel{?}{\leftarrow} V \rangle$.
(c) After (b) is done and $\langle S : \text{ac-id} \stackrel{?}{\leftarrow} V \rangle$ (if it exists) is satisfied, S can add the credential edge $T \leftarrowtail \langle V : S \stackrel{?}{\leftarrow} S \rangle$.
(d) After (a) is done and $\langle S : \text{ack-id} \stackrel{?}{\leftarrow} V \rangle$ is satisfied, if S has a delegation credential $A.R \longleftarrow A_1.R_1$, S can add the credential edge $T \leftarrowtail \langle V : A_1.R_1 \stackrel{?}{\leftarrow} S \rangle$.
(e) S can mark T as opponent-processed if T is satisfied, or all of the above steps are done.

2. Processing $T = \langle V : \text{attr} \stackrel{?}{\leftarrow} S \rangle$
(a) If there exists a policy statement full-id with head $\mathsf{disclose}(\mathsf{full}, \text{attr})$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{full-id} \stackrel{?}{\leftarrow} V \rangle$.
(b) If there exists a policy statement bit-id with head $\mathsf{disclose}(\mathsf{bit}, \text{attr})$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{bit-id} \stackrel{?}{\leftarrow} V \rangle$.
(c) If there exists a policy statement range-id with head $\mathsf{disclose}(\mathsf{range}, \text{attr}, \text{precision})$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{range-id} \stackrel{?}{\leftarrow} V \rangle$.
(d) S can mark T as opponent-processed if T is satisfied, or all of the above steps are done.