PRIVACY ENHANCED AUTOMATED TRUST NEGOTIATION

by Jiangtao Li

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

PRIVACY ENHANCED AUTOMATED TRUST NEGOTIATION

by Jiangtao Li

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

PRIVACY ENHANCED AUTOMATED TRUST NEGOTIATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Jiangtao Li

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2006

Purdue University

West Lafayette, Indiana

To my father, my mother, and my wife Jie.

ACKNOWLEDGMENTS

Throughout my graduate study at Purdue University, I have been very fortunate to meet with great people, who have had a dramatic impact on my research and my accomplishments.

I am very grateful to my advisor Professor Mikhail Atallah, who always advised me, supported me, and helped me during my six year graduate research. He has provided me a lot of support and freedom that I needed to be successful in my research. His insights and suggestions have been invaluable to my work. I am also very grateful to my other advisor Profoessor Ninghui Li. It has been a pleasure to work with him, to talk with him, and to learn from him. His willingness to discuss with me at anytime has been very helpful. Without my advisors' help and support, this thesis report would never have been realized.

My graduate studies were conducted in the Center for Education and Research in Information Assurance and Security (CERIAS). I benefited a lot from the research and education environment that CERIAS provided. I thank all the faculties, staffs, and colleagues in CERIAS for their support.

Most of all, I would like to thank my beloved wife, my parents, and my brother for their unconditional love, encouragement, and support.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ABAC	Attribute-Based Access Control
ATN	Automated Trust Negotiation
CA	Certificate Authority
CIPPE	Certified Input Private Policy Evaluation
CIOT	Committed-Integer based Oblivious Transfer
CDH	Computational Diffie-Hellman
DDH	Decision Diffie-Hellman
DL	Discrete Logarithm
OACerts	Oblivious Attribute Certificate
OCBE	Oblivious Commitment-Based Envelope
OSBE	Oblivious Signature-Based Envelope
PKG	Private Key Generator
PKI	Public Key Infrastructure
PPT	Probabilistic Polynomial Time
RE	Reverse Eager
SCE	Scrambled Circuit Evaluation

ABSTRACT

Li, Jiangtao. Ph.D., Purdue University, May, 2006. Privacy Enhanced Automated Trust Negotiation. Major Professors: Mikhail J. Atallah and Ninghui Li.

In automated trust negotiation, 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 trust negotiation, 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).

In this thesis, we introduce a number of techniques that address the previous problems. In particular,

- We propose Oblivious Attribute Certificates (OACerts), an attribute certificate scheme in which a certificate holder can select which attributes to use and how to use them. In particular, a user can use attribute values stored in an OACert to obtain a resource from a service provider without revealing any information about these values. Using OACerts, we develop a policy-hiding access control scheme that protects both sensitive attribute values and sensitive policies.

- We present a privacy-preserving trust negotiation protocol that enforces each credential's policy (thereby protecting sensitive credentials). Our result is not achieved through the routine use of standard techniques to implement, in this framework, one of the known strategies for trust negotiations (such as the "eager strategy").

Rather, we use novel techniques to implement a non-standard trust negotiation strategy specifically suited to this framework.

- We introduce a framework for automated trust negotiation in which 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.

1 INTRODUCTION

Computer systems traditionally are closed, centrally managed systems in which each subject has one or more identities. The system grants or denies a subject's requests to access certain resources based on its access control policies and the authenticated identities of the requester. It is assumed that subjects in the system already know each other. Thus, trust can be easily established based on each other's identity. Furthermore, without obtaining a local identity, a subject is not able to interact with the system and gain access to the resources of the system.

The move towards a globally interconnected infrastructure and open environment, such as the Internet, provides opportunities for two or more parties who are strangers to each other to share resources or conduct business transactions. Such interactions often involve release of sensitive information and remote access to a party's local resources. Mutual trust between two parties is crucial in such environments. As parties belong to different security domains controlled by different authorities, establishing trust based on identity is not a feasible solution. Therefore, identity information such as username and password is usually inadequate for establishing trust between strangers.

Blaze, Feigenbaum, and Lacy [1] introduced the term *trust management* to group together some principles dealing with decentralized authorization. In trust management [1–6], access control decisions are based on authenticated attributes of the subjects, which are established by digitally signed credentials. Each credential associates a public key with the key holder's identity and/or attributes such as employer, group membership, credit card information, birth-date, citizenship, and so on. Because these credentials are digitally signed, they can serve to introduce strangers to one another without online contact with the attribute authorities.

Winsborough, Seamons, and Jones [7] introduced the notion of *automated trust negotiation* (ATN). The goal of ATN [7–12] is to enable resource requesters and access mediators to establish trust in one another through cautious, iterative, bilateral disclosure of credentials. In the existing ATN literature, access control policies are established to regulate the disclosure of credentials, in addition to the granting of system resources. The negotiation consists of a sequence of exchanges that begin with disclosing credentials that are not sensitive. As credentials flow, higher levels of mutual trust are established, and access control policies for more sensitive credentials are satisfied, enabling these credentials also to flow. In successful negotiations, credentials eventually flow that satisfy the policy of the desired resource. Trust negotiation differs from trust management in that:

1. In trust negotiation, credentials are modeled as sensitive information, and protected by access control policies just same as other resources in the system.

2. Trust negotiation is performed in a peer-to-peer architecture, where a client and a server are treated equally. Instead of a one-shot authorization, trust is established incrementally through a sequence of credential disclosure.

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.

1. 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 Bob would allow Alice to access a resource provided Alice is over 21, and Alice has a digital driver license that includes Alice's birth-date and address. If Alice does not want to reveal her address (or her exact birth-date) to Bob, the negotiation would fail, even if Alice were willing to prove she is over 21.

2. 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 Bob is a bank that offers a special-rate loan and Alice would like to know whether she is eligible for such a loan before she applies. Bob is willing to reveal that his loan-approval policy uses one's birth-date, current salary, and the length of the current employment; however, Bob considers further details of this policy to be a trade secret that he is unwilling to reveal. Alice would like to know whether she is eligible for the loan while disclosing as little information about her attributes as possible. In particular, Alice does not want to disclose the exact values of her birth-date or salary level. Using traditional ATN techniques, this negotiation would fail.

3. If there is a cyclic dependency among credentials and their policies, negotiations can fail unnecessarily. For example, in a negotiation between Alice and Bob, suppose Alice has a credential c_1 that can be disclosed only if Bob has c_2, and Bob has c_2, but can disclose it only if Alice 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 Alice and Bob to exchange *both* c_1 and c_2 would not violate either negotiator's policy.

Thesis Statement

The goal of my thesis is to design an ATN scheme that has better privacy protection and is able to avoid the above mentioned limitations. More specifically, our goal is to develop cryptographic credentials and protocols for Alice and Bob to negotiate trust while minimizing the disclosure of each party's sensitive credentials and policies. Furthermore, we want to develop a new ATN framework that supports these cryptographic credentials and protocols.

Our Contribution

To address the previously mentioned limitations of ATN, we propose Oblivious Attribute Certificates (OACerts), an attribute certificate scheme in which a certificate holder can select which attributes to use and how to use them. Using OACerts, a certificate holder is no longer limited to the all-or-nothing property of traditional trust negotiation. For example, suppose Alice's digital driver license is documented using OACerts. Alice can prove that she is older than 21 without revealing her exact birth-date. Thus, the first above-mentioned limitation is naturally solved. Furthermore, a user can use attribute values stored in an OACert obliviously, i.e., the user obtains a service if and only if the attribute values satisfy the policy of the service provider, yet the service provider learns nothing about these attribute values. This way, the service provider's access control policy is enforced in an oblivious fashion.

Based on the OACerts scheme, we further develop a policy-hiding access control scheme that protects both sensitive attributes and sensitive policies. That is, Bob can decide whether Alice's certified attribute values satisfy Bob's policy, without Bob learning any other information about Alice's attribute values or Alice learning Bob's policy. Using this policy-hiding access control, we can address the second limitation.

We develop a privacy-preserving trust negotiation protocol and several novel cryptographic protocols for carrying it out. We propose a reverse eager trust negotiation strategy that handles arbitrary policy cycles, whereas the existing traditional trust negotiation strategies are inherently unable to handle such cycles. Using our protocol, Alice and Bob can determine whether the trust can be established without disclosing any of their private credentials and policies.

Finally, we introduce a framework for trust negotiation 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.

Organization of the Thesis

This thesis is organized as follows. We first review several cryptographic tools that will be used in our thesis in Chapter 2. We then present our construction of OACerts in Chapter 3. In Chapter 4, we present a policy-hiding access control scheme based on OACerts. In Chapter 5, we describe a privacy-preserving trust negotiation protocol that can seamlessly handle policy cycles. In Chapter 6, we present a trust negotiation framework that supports diverse cryptographic credentials and protocols. Finally, we discuss the related work in Chapter 7 and summarize this thesis in Chapter 8.

2 CRYPTOGRAPHIC TOOLS

In this chapter we review some cryptographic tools and building blocks that will be used in this thesis, right after a brief description of several standard assumptions in cryptography. We say that a function f is *negligible* in the security parameter t if, for every polynomial p, $f(t)$ is smaller than $1/|p(t)|$ for large enough t; otherwise, it is *non-negligible*.

- *Discrete Logarithm (DL) Assumption.* The DL problem is the following: Given a finite cyclic group G, a generator $g \in G$, and a group element y, compute $\log_g y$. The DL assumption is that there exists no polynomial-time algorithm that can solve the DL problem with non-negligible probability.

- *Computational Diffie-Hellman (CDH) Assumption.* The CDH problem is the following: Given a finite cyclic group G, a generator $g \in G$, and group elements g^a, g^b, compute g^{ab}. The CDH assumption is that there exists no polynomial-time algorithm that can solve the CDH problem with non-negligible probability.

- *Decisional Diffie-Hellman (DDH) Assumption.* The DDH problem is the following: Given a finite cyclic group G, a generator $g \in G$, and group elements g^a, g^b, and g^c, output 0 if $g^c = g^{ab}$ and 1 otherwise. The DDH assumption is that there exists no polynomial-time algorithm that can solve the DDH problem with non-negligible advantage. The advantage of an algorithm is its success probability minus $1/2$, as one can always randomly guess with a $1/2$ success probability.

- *Random Oracle Model.* The random oracle model is an idealized security model introduced by Bellare and Rogaway [13] to analyze the security of certain natural cryptographic constructions. Roughly speaking, a random oracle is a function H: $X \rightarrow Y$ chosen uniformly at random from the set of all functions $\{h \colon X \rightarrow Y\}$ (we assume Y is a finite set). An algorithm can query the random oracle at any point

$x \in X$ and receive the value $H(x)$ in response. Random oracles are used to model cryptographic hash functions such as SHA-1. Note that security in the random oracle model does not imply security in the real world. Nevertheless, the random oracle model is a useful tool for validating natural cryptographic constructions.

2.1 Cryptographic Commitment Scheme

Informally speaking, a commitment scheme enables a prover to commit a value to a verifier such that the verifier does not know which value has been committed, and the prover cannot change its mind after having committed. In this section, we briefly describe the Pedersen commitment scheme [14] that we use throughout this thesis.

Definition 2.1.1 (The Pedersen Commitment Scheme)

Setup A trusted third party T chooses two large prime numbers p and q such that q divides $p - 1$. It is typical to have p be 1024 bits and q be 160 bits. Let g be a generator of G_q, the unique order-q subgroup of $\mathbb{Z}_p^*$. We use $x \leftarrow \mathbb{Z}_q$ to denote that x is uniformly randomly chosen from $\mathbb{Z}_q$. T picks $x \leftarrow \mathbb{Z}_q$ and computes $h = g^x \bmod p$. T keeps the value x secret and makes the values $\langle p, q, g, h \rangle$ public.

Commit The domain of the committed values is $\mathbb{Z}_q$. For the prover to commit an value $a \in \mathbb{Z}_q$, the prover chooses $r \leftarrow \mathbb{Z}_q$ and computes the commitment $c = g^a h^r \bmod p$.

Open To open a commitment c, the prover reveals a and r, and the verifier verifies whether $c = g^a h^r \bmod p$.

The above setting is slightly different from the standard setting of commitment schemes, in which the verifier runs the setup program and does a zero-knowledge proof to convince the prover that the parameters are constructed properly.

The Pedersen commitment scheme is *unconditionally hiding*: Even with unlimited computational power it is impossible for an adversary to learn any information about the value a from c, because the commitments of any two numbers in $\mathbb{Z}_q$ have exactly the same distribution. This commitment scheme is *computationally binding*: Under the DL assumption, it is computationally infeasible for an adversarial prover to open a value a' other than a in the open phase of the commitment scheme. Suppose an adversary finds a' (other than a) and r' such that $g^{a'}h^{r'} \equiv g^a h^r (\bmod\ p)$, then she can compute $\frac{a'-a}{r-r'} \bmod q$, which is $\log_g(h)$, the discrete logarithm of h with respect to the base g.

2.2 Homomorphic Encryption

A homomorphic encryption scheme [15–18] is an encryption scheme in which the plaintexts are taken from a group G, and given the encryptions of two group elements one can efficiently compute a encryption of their sum. Usually this computation involves a modular multiplication of the encryptions, we write $E(a) \cdot E(b) = E(a+b)$. It is easy to see that $E(a)^c = E(c \cdot a)$. Damgård and Jurik [18] proposed a homomorphic encryption scheme in which all users can use the same modulus when generating key pairs.

Definition 2.2.1 (Damgård-Jurik Cryptosystem) Let $n = pq$ be an RSA modulus, with $p = 2p' + 1$ and $q = 2q' + 1$ where p, q, p', q' are primes. Let g be a generator of Q_n, the group of all squares of $\mathbb{Z}_n^*$.

Key Generation Choose $\alpha \in \mathbb{Z}_\tau$ where $\tau = p'q' = |Q_n|$. The public key is then (n, q, h) with $h = g^\alpha \bmod n$ and the private key is α.

Encryption Given a plaintext $m \in \mathbb{Z}_n$, choose a random $r \in \mathbb{Z}_n$, and the ciphertext is $E(m, r) = (g^r \bmod n, (h^r \bmod n)^n (n+1)^m \bmod n^2)$.

Decryption Given a ciphertext $c = (G, H) = E(m, r)$, m can be found as

$$\begin{aligned} m &= L(H(G^\alpha \bmod n)^{-n}) \\ &= L((g^{\alpha r} \bmod n)^n (n+1)^m (g^{r\alpha} \bmod n)^{-n}) \\ &= L((n+1)^m \bmod n^2) = m \bmod n. \end{aligned}$$

Damgård-Jurik cryptosystem is a homomorphic encryption scheme. To see why, let $m, m' \in \mathbb{Z}_n$ and $r, r' \in_R \mathbb{Z}_n$, let $E(m, r) = (G, H)$ and $E(m', r') = (G', H')$. We define

$E(m,r) \cdot E(m',r')$ to be $(G \cdot G', H \cdot H')$, it is easy to verify that $E(m,r) \cdot E(m',r) = E(m + m' \bmod n, r + r')$. In the rest of this thesis, we will use $E(m)$ as a shorthand for $E(m,r)$.

Damgård-Jurik cryptosystem is *semantically secure* [18] under the Decisional Composite Residuosity Assumption and DDH assumption. The semantic security property guarantee that an eavesdropper cannot learn any information about the plaintext from the ciphertext. More precisely, given two arbitrary message m_0 and m_1, the random variables representing the two homomorphic encryptions $E(m_0)$ and $E(m_1)$ are computationally indistinguishable.

2.3 Identity-Based Encryption

The concept of Identity-Base Encryption (IBE) was first proposed by Shamir [19] in 1984, however the first usable IBE systems were discovered only recently [20, 21]. An IBE scheme is specified by following four algorithms:

Setup A Private Key Generator (PKG) takes a security parameter k and generates system parameters params and a master secret s. params is public, whereas s is private to PKG.

Extract Given any arbitrary string $id \in \{0,1\}^*$, PKG uses params, s, and id to compute the corresponding private key sk.

Encrypt It takes params, id and plaintext M as input and returns ciphertext C. We use $I(M, id)$ to denote the encryption algorithm using the identity id.

Decrypt It takes params, sk and ciphertext C as input and returns the corresponding plaintext M. We use $I^{-1}(C, sk)$ to denote the decryption algorithm using the private key sk. Of course, the decryption algorithm must satisfy the standard consistency constraint, namely for any identity id, the corresponding private key sk, and any message M, the equation $I^{-1}(I(M, id), sk) = M$ is always true.

An IBE scheme enables a sender to encrypt a message using a receiver's identity as the public key, thus avoids obtaining the public key from the receiver or a directory. Boneh and Franklin proposed an IBE scheme from weil pairing [20]. Their scheme is secure against adaptive chosen ciphertext attacks (IND-ID-CCA).

2.4 Hidden Credentials

The hidden credentials system was proposed by Holt *et al.* [22]. In the hidden credentials system, there is a trusted CA who issues credentials for users in the system. Each user in the system is assigned with a unique nym, where nym could be either a real name or a pseudonym. A hidden credential is a digital signed assertion about an attribute of a credential holder by the CA. Roughly speaking, given an IBE scheme as described in the previous section, a hidden credential $cred$ for username nym and attribute $attr$ is the private key corresponding to the identity $nym||attr$. More specifically, the hidden credentials system has following four programs:

1. $\mathbf{CA\_Create}()$: The CA runs the setup program of the IBE system and generates system parameters params and a master secret s, and publishes params. The CA also publishes a list of possible attribute names.

2. $\mathbf{CA\_Issue}(\mathbf{nym}, \mathbf{attr})$: The CA issues a credential for user with username nym and an attribute $attr$ by running the extract program of the IBE system with $id = nym||attr$, and outputs the private key sk as the credential. Given a hidden credential $cred$, we use $cred.nym$ to denote the corresponding username, and $cred.attr$ to denote the corresponding attribute in the credential.

3. $\mathbf{I}(\mathbf{M}, \mathbf{nym}||\mathbf{attr})$: This program corresponds to the encrypt algorithm of the IBE system with system parameters params, $id = nym||attr$, and plaintext M. The output of this program is C.

4. $\mathbf{I^{-1}(C, cred)}$: This function corresponds to the decrypt program of the IBE system with system parameters params, $cred$, and ciphertext C. The output of this function is M.

The hidden credentials system is secure against an adaptive chosen ciphertext attack where an attacker can obtain unlimited number of other arbitrary credentials [22]. The hidden credentials are also unforgeable. We here give a simple example of how Alice accesses Bob's resource using the hidden credentials. Suppose Bob's resource M can only be accessed by a student. Alice has a student credential $cred$, ***i.e.***, $cred.nym = Alice$ and $cred.attr = stu$. To access M, Alice sends her username $Alice$ to Bob who responds with $I(M, Alice||stu)$. Alice uses her credential $cred$ to decrypt $I(M, Alice||stu)$ and obtains M. Bob does not learn whether Alice possesses a student credential or not from the interaction.

2.5 Scrambled Circuit Evaluation

The Scrambled Circuit Evaluation (SCE) protocol was developed by Yao [23]. This protocol runs between two players: a *generator* and an *evaluator*. In the SCE protocol, the generator "scrambles" the circuit in some manner, then two players interact, the evaluator "evaluates" the scrambled circuit, and finally the evaluator sends the result of the evaluation to the generator who recovers the final result.

Let x be the evaluator's input, and y be the generator's input. Let $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be a function known to both parties. In the end, both parties learn $f(x, y)$. The SCE protocol takes the following steps:

Encrypting the circuit Assume that $E_k\,[\,M\,]$ is a semantically secure encryption function for the message M using the key k. Suppose the circuit for the function $f(x, y)$ consists of s gates $g_1, \ldots, g_s$ and t wires $w_1, \ldots, w_t$, where each gate g_i has two input wires and one output wire; we use g_i to also denote the function $\{0,1\}^2 \rightarrow \{0,1\}$ computed by the gate. The generator scrambles the circuit as follows.

1. The generator chooses $2t$ random keys $k_1^0, k_1^1, \ldots, k_t^0, k_t^1$ and assigns a pair of random keys $\{k_i^0, k_i^1\}$ to each wire w_i for $1 \leq i \leq t$.

2. For each gate g_i in the circuit, the generator constructs a table T_i as follows:

   (a) Let w_a and w_b be the input wires of gate g_i, and w_c be the corresponding output wire, where $1 \leq a, b, c \leq t$.

   (b) The generator computes the following four values:

   $$m_{0,0} = E_{k_b^0}\left[E_{k_a^0}\left[k_c^{g_i(0,0)} \,||\, 0^\sigma\right]\right] \qquad m_{0,1} = E_{k_b^1}\left[E_{k_a^0}\left[k_c^{g_i(0,1)} \,||\, 0^\sigma\right]\right]$$
   $$m_{1,0} = E_{k_b^0}\left[E_{k_a^1}\left[k_c^{g_i(1,0)} \,||\, 0^\sigma\right]\right] \qquad m_{1,1} = E_{k_b^1}\left[E_{k_a^1}\left[k_c^{g_i(1,1)} \,||\, 0^\sigma\right]\right]$$

   where $m_{x,y}$ (for $x \in \{0,1\}$ and $y \in \{0,1\}$) corresponds to the case that the input wire w_a has value x and the input wire w_b has value y, and $k_c^{g_i(x,y)}||0^\sigma$ means concatenating the random value corresponds to the wire w_c having value $g_i(x,y) \in \{0,1\}$ with a binary string of σ 0's.

   (c) The generator randomly permutes the set $\{m_{0,0}, m_{0,1}, m_{1,0}, m_{1,1}\}$ and stores it in the table T_i.

   For example, the table for the gate g_i when it is an AND gate would contain the following four entries in some random order:

   $$m_{0,0} = E_{k_b^0}\left[E_{k_a^0}\left[k_c^0 \,||\, 0^\sigma\right]\right] \qquad m_{0,1} = E_{k_b^1}\left[E_{k_a^0}\left[k_c^0 \,||\, 0^\sigma\right]\right]$$
   $$m_{1,0} = E_{k_b^0}\left[E_{k_a^1}\left[k_c^0 \,||\, 0^\sigma\right]\right] \qquad m_{1,1} = E_{k_b^1}\left[E_{k_a^1}\left[k_c^1 \,||\, 0^\sigma\right]\right]$$

   If the evaluator knows (k_a^1,k_b^1), the two keys corresponding to the 1 value in wires w_a and w_b, and tries to decrypt the four entries, the evaluator will find garbage when trying to decrypt $m_{0,0}, m_{0,1}, m_{1,0}$ and successfully decrypt $m_{1,1}$. The evaluator can tell that the decryption of $m_{1,1}$ is successful by finding the binary string 0^σ in the decrypted message. This enables the evaluator to learn k_c^1, the value corresponds to the wire w_c being 1. Of course, w_c should be 1 when both w_a and w_b are 1. If the evaluator knows (k_a^1,k_b^0), then it can successfully decrypt $m_{1,0}$ and recover k_c^0. In the other two cases, the evaluator recovers k_c^0 as well.

3. The generator sends $T_1, \ldots, T_s$ to the evaluator. The generator sends also the topology of the circuit, so that the evaluator knows which gate connects to which.

Coding the input The evaluator learns a random key for each input wire as follows.

1. For each wire w_i that corresponds to the generator's input, the generator sends k_i^0 to the evaluator if his input is 0, he sends k_i^1 if his input is 1.
2. For each wire w_j that corresponds to the evaluator's input, the generator and the evaluator engage in a 1-out-of-2 oblivious transfer protocol [24–26] in which the generator provides k_j^0 and k_j^1, and the evaluator chooses k_j^0 if her input is 0, and chooses k_j^1 otherwise.

Evaluating the circuit The evaluator evaluates the scrambled circuit gate-by-gate, starting from the circuit-input gates and ending at the circuit-output gates. Each gates g_i is evaluated as follows:

1. The evaluator can evaluate gate g_i only if she has learned one key for each of the input wires.
2. Let w_a, w_b, w_c be the corresponding input wires and output wire of gate g_i. Assume k_a^x and k_b^y are the keys the evaluator learned that correspond to wires w_a and w_b, respectively.
3. Let T_i be the table corresponding to gate g_i. The evaluator uses k_a^x and k_b^y to decrypt each entry in T_i and succeeds in the entry $m_{x,y} = E_{k_b^y}\left[E_{k_a^x}\left[k_c^{g_i(x,y)}\right]\right]$. Thus she learns $k_c^{g_i(x,y)}$, one of the two keys corresponding to the output wire w_c.

Finally, the evaluator obtains the output of the scrambled circuit, and sends it back to the generator. The generator learns $f(x, y)$ and reveals the result to the evaluator.

The SCE protocol is secure in honest-but-curious model [23, 26]. As for the security of each gate, it is necessary to construct the table using a *non-malleable* encryption scheme [27] (such as AES), to prevent the evaluator from making meaningful changes in the plaintext by changing the ciphertext. Provided that the encryption scheme is non-malleable, knowledge of one key for each of the input wires discloses only one key of the output wire. The other key is unknown to the evaluator. As for the security of the entire circuit, the oblivious transfer protocol ensures that the evaluator learns just one key per input wire, and the generator does not learn which value the evaluator chose. Therefore, the evaluator can obtain one and only one key per wire in the circuit. As the mapping between the (two) random keys of each wire and the Boolean values is unknown to the evaluator, she learn neither the type of each gate, nor any intermediate results of the original circuit.

3 OACERTS: OBLIVIOUS ATTRIBUTE CERTIFICATES

Privacy is an important concern in the use of Internet and web services. When the attribute information in a certificate is sensitive, the certificate holder may want to disclose only the information that is absolutely necessary to obtain services. Consider the following example.

Example 1 A senior citizen Alice requests from a service provider Bob a document that can be accessed freely by senior citizens. Alice wants to use her digital driver license to prove that she is entitled to free access. Alice's digital driver license certificate has fields for an identification number, expiration date, name, address, birth-date, and so on; and Alice would like to reveal as little information as possible.

In the above example, suppose Alice's digital driver license is an X.509 certificates [28], Alice first sends her request to Bob who responds with the policy that governs access to that document. Alice then sends her driver license certificate to Bob. After Bob receives the certificate from Alice and verifies it, he grants Alice access to the document. Observe that, in this scenario, Bob learns all the attribute information (*i.e.*, name, address, birth-date, *etc.*) in Alice's driver license.

Suppose Alice's digital driver license is an anonymous credential [29–33], after Bob reveals his access control policy, Alice can prove to Bob that she is a senior citizen without leaking any additional information. Now it might seem that Alice needs to reveal at least the fact that she is a senior citizen, i.e., her birth-date is before a certain date. However, even this seemingly minimal amount of information disclosure can be avoided. Suppose that the document is encrypted under a key and the encrypted document is freely available to everyone. Further suppose a protocol exists such that after the protocol is executed between Alice and Bob, Alice obtains the key if and only if the birth-date in her driver license is before a certain date and Bob learns nothing about Alice's birth-date. Under these conditions, Bob can perform access control based on Alice's attribute values while being oblivious about Alice's attribute information.

We call this *oblivious access control*, because Bob's access control policies for his resources are enforced without Bob learning any information about Alice's certified attribute values, not even whether Alice satisfies his policy or not. It is important to point out that a process that protects Alice's attributes from Bob is not only to Alice's advantage but also to Bob's: Bob no longer needs to worry about rogue insiders in his organization illicitly leaking (or selling) Alice's private information, and may even lower his liability insurance rates as a result of this. Privacy-preservation is a win-win proposition, one that is appealing even if Alice and Bob are honest and trustworthy entities.

To enable such oblivious access control, we propose Oblivious Attribute Certificates (OACerts), a scheme for using certificates to document sensitive attributes. The basic idea of OACerts is quite simple. Instead of storing attribute values directly in the certificates, a certificate authority (CA) stores the cryptographic commitments [14, 34–36] of these values in the certificates. Using OACerts, a user can select *which* attributes to use as well as *how* to use them. An attribute value in an OACert can be used in several ways: (1) by opening a commitment and revealing the attribute value, (2) by using zero-knowledge proof protocols [37–40] to prove that the attribute value satisfies a condition without revealing other information, and (3) by running a protocol so that the user obtains a message only when the attribute value satisfies a condition, without revealing any information about the attribute value. The idea of storing cryptographic commitments of attribute values in certificates was used in anonymous credentials [29–33]; however, we are not aware of prior work on the oblivious usage of such attribute values.

In Example 1, suppose that the driver-license certificate that Alice has is an OACert. With attribute values committed rather than stored in the clear in her certificates, Alice can send her certificate to Bob without revealing her birth-date or any other attribute information. Using zero-knowledge proof protocols [37–40], Alice can prove to Bob that her committed birth-date is before a certain date without revealing any other information. However, our goal is that Bob should learn nothing about Alice's birth-date, not even whether Alice is a senior citizen or not. To enable oblivious access control, we need to solve the following two-party Secure Function Evaluation (SFE) problem:

Problem 1 Let commit be a commitment algorithm, let Params be public parameters for commit, and Pred be a public predicate. Let a be a private number (Alice's attribute value), $c = \mathsf{commit}_{\mathsf{Params}}(a, r)$ be a commitment of a under the parameters Params with a random number r, and M be a private message (Bob wants Alice to see M if and only if a satisfies Pred). Alice and Bob jointly compute a family F of functions, parameterized by commit and Pred. Both parties have commit, Pred, Params, and c. Bob has private input M. Alice has private input a and r. The function F is defined as follows.

$$F[\mathsf{commit}, \mathsf{Pred}]_{Bob}(\mathsf{Params}, c, M, a, r) = \varnothing$$

$$F[\mathsf{commit}, \mathsf{Pred}]_{Alice}(\mathsf{Params}, c, M, a, r)$$

$$= \{ \begin{array}{ll} M & \text{if } c = \mathsf{commit}_{\mathsf{Params}}(a, r) \wedge \mathsf{Pred}(a) = \mathsf{true}; \\ \varnothing & \text{otherwise}. \end{array}$$

where $F[\mathsf{commit}, \mathsf{Pred}]_{Alice}$ represents Alice's output, $F[\mathsf{commit}, \mathsf{Pred}]_{Bob}$ represents Bob's output. In other words, our goal is that Bob learns nothing and Alice learns M only when her committed attribute value satisfies the predicate Pred.

The preceding problem can be solved using general solutions to two-party SFE [23, 26, 41]; however, the general solutions are inefficient, as commitment verification is done within the SFE. We propose an Oblivious Commitment Based Envelope (OCBE) scheme that solves the above two-party SFE problem efficiently. Formal definition of OCBE will be given in Section 3.3. Informally, an OCBE scheme enables a sender Bob to send an envelope (encrypted message) to a receiver Alice, such that Alice can open the envelope if and only if her committed value satisfies the predicate. An OCBE scheme is *oblivious* if at the end of the protocol the sender cannot learn any information about the receiver's committed value. An OCBE scheme is *secure against the receiver* if a receiver whose committed value does not satisfy the predicate cannot open the envelope.

We develop efficient OCBE protocols for the Pedersen commitment scheme 2.1 and six kinds of comparison predicates: $=, \neq, <, >, \leq, \geq$, as well as conjunctions and disjunctions of multiple predicates. These predicates seem to be the most useful ones for testing attribute values in access control policies. We present a protocol (called EQ-OCBE) for equality predicates and a protocol (called GE-OCBE) for greater-than-or-equal-to predicates and prove that these protocols are provably secure in the Random Oracle Model [13]. These protocols use cryptographic hash functions to efficiently derive symmetric encryption keys from a shared secret, and random oracles are used to model such usage of hash functions. We also show that it is easy to construct OCBE protocols for other comparison predicates using variants of EQ-OCBE and GE-OCBE.

3.1 Architecture of OACerts

In this section, we present the architecture of the OACerts scheme. There are three kinds of parties in the OACerts scheme: certificate authorities (CA's), certificate holders, and service providers. A CA issues OACerts for certificate holders. Each CA and each certificate holder has a unique public-private key pair. A service provider, when providing services to a certificate holder, performs access control based on the attributes of the certificate holder, as certified in OACerts.

An OACert is a digitally signed assertion about the certificate holder by a CA. Each OACert contains one or more attributes. We use $attr_1, \ldots, attr_m$ to denote the m attribute names in an OACert, and $v_1, \ldots, v_m$ to denote the corresponding m attribute values. Let $c_i = \mathsf{commit}_{\mathsf{Params}}(v_i, r_i)$ be the commitment of attribute value v_i for $1 \leq i \leq m$ with r_i being the secret random number. The attribute part of the certificate consists of a list of m entries, each entry is a tuple $\langle attr_i, c_i \rangle$. When the commitment scheme used is secure, the certificate itself does not leak any information about the sensitive attributes. Thus, an OACert's content can be made public. A certificate holder can show his OACerts to others without worrying about the secrecy of his attributes.

In many commitment schemes [14, 34, 35], the input domain is the set of integers; hence it is necessary to map an arbitrary attribute value to an integer in OACerts. For example in a digital driver license, gender can be expressed by a single bit, state can be expressed by a number from $[1, 50]$, birth-date can be expressed by the number of days between January 1st of 1900 and the date of birth. In an another example, suppose a digital student certificate contains an attribute for major. As the number of different majors is finite (and quite small in practice), we can easily encode each major with a number. There are certain attributes of which the values could be arbitrary, such as name or home address. We cannot represent those attribute values directly with integers, in this case, the CA hashes the attribute values using a collision-free hash function and commits the hash values in OACerts.

OACerts can be implemented on existing public-key infrastructure standards, such as X.509 Public Key Infrastructure Certificate [28, 42] and X.509 Attribute Certificate [43]. The commitments can be stored in X.509v3 extension fields, in which case a certificate includes also the following fields: serial number, validity period, issuer name, user name, certificate holder's public key, and so on. The distribution and revocation of OACerts can be handled using existing infrastructure and techniques. See Section 3.5 for our implementation and performance measurements of OACerts.

There are four basic protocols in the OACerts scheme:

- **CA-Setup:** A CA picks a signature scheme Sig with a public-private key pair $(K_{\mathrm{CA}}, K_{\mathrm{CA}}^{-1})$, and a commitment scheme commit with public parameters Params. The public parameters of the CA are $\{\mathsf{Sig}, K_{\mathrm{CA}}, \mathsf{commit}, \mathsf{Params}\}$.

- **Issue Certificate:** A CA uses this protocol to issue an OACert to a user. A user Alice generates a public-private key pair (K_A, K_A^{-1}) and sends to the CA a certificate request that includes her public key K_A and attributes information $(attr_1, v_1)$, ..., $(attr_m, v_m)$, signed by K_A^{-1}. After the CA verifies the correctness of $v_1, \ldots, v_m$ (most likely using off-line methods), it issues an OACert for Alice. In this process, the CA computes $c_i = \mathsf{commit}_{\mathsf{Params}}(v_i, r_i)$ and sends the certificate along with the secrets $r_1, \ldots, r_m$ to Alice. Alice stores the certificate and stores the values $(v_1, r_1), \ldots, (v_m, r_m)$ together with her private key K_A^{-1}. The role of the CA here is similar to the role of a CA in the traditional Public Key Infrastructure.

- **Alice-Bob initialization:** Alice, a certificate holder, establishes a secure communication channel with Bob, a service provider, and at the same time proves to Bob the ownership of an OACert. In this protocol, Bob checks the signature and the validity period of the certificate, then verifies that the certificate has not been revoked (using, e.g., standard techniques in [28]). Bob also verifies that Alice possesses the private key corresponding to K_A in the OACert. All these can be done using standard protocols such as TLS/SSL [44].

  Alice then requests the decryption key for an encrypted document, and Bob sends Alice his policy.

- **Alice-Bob Interaction:** Alice can show any subset of her attributes using the show attribute protocols. These protocols are executed after the show certificate protocol, through a secure communication channel between Alice and Bob. To show t attributes, Alice runs show attribute protocols t times. There are three kinds of show attribute protocols; each gives different computational and communication complexity and privacy level.

  1. *direct show:* Alice gives v_i and r_i directly to Bob, and Bob verifies $c_i = \mathsf{commit}(v_i, r_i)$. This protocol is used when Alice trusts Bob with the attribute values, or when Alice is very weak in computational power. This protocol is the most efficient one but offers the least privacy protection. Bob not only knows v_i but also can convince others that Alice has attribute v_i.

  2. *zero-knowledge show:* Alice uses zero-knowledge proofs to prove v_i satisfies some properties Bob requires, e.g., is equal to some value or belongs to some range. This kind of protocols is more expensive than the direct show, but offers better privacy protection. Bob learns whether v_i satisfies his policies, but he

cannot convince others about this. Bob also doesn't learn the exact value of v_i provided that multiple values satisfy his policies.

3. *oblivious show:* Alice interacts with Bob using OCBE protocols. Bob learns nothing about v_i. This kind of oblivious show protocols offers the best privacy protection among the three types of show protocols. Often times, it has similar or less computation than the zero-knowledge show protocols.

In practice, Alice and Bob may not share the same CA. That is, Bob may not know the CA that issues the OACerts to Alice and Bob may not trust that CA. We can handle this problem using a hierarchy of CAs with only the root of the hierarchy being trusted by Bob. For example, Alice is a student at StateU, and has a student certificate issued by College of Science (CoS) of StateU using the OACerts scheme. CoS has a valid certificate issued by StateU; and StateU is certified by Accreditation Board for Engineering and Technology (ABET). The certificate chain to prove that Alice is a valid student takes the form $\mathrm{ABET} \rightarrow \mathrm{StateU} \rightarrow \mathrm{CoS} \rightarrow \mathrm{Alice}$. There are three certificates associated with this chain, where the first two certificates are regular certificates (as there are no sensitive information in these certificates) and the last one is an OACert. Suppose Bob's policy is that only students in computer science can access the resource, and suppose Bob trusts ABET. Alice can first show the certificate chain to Bob without leaking any attribute information in her student certificate, and then run a zero-knowledge proof protocol to prove that her major is computer science.

Another practical consideration is that different CAs may use different attribute names for the same attribute. For example, Bureau of Motor Vehicles (BMV) may use DoB as the attribute name for birth-date in the driver license, whereas a Bureau of Consular Affairs may use $\mathrm{date\ of\ birth}$ as the attribute name for birth-date in the passport. Alice and Bob can use application domain specification documents [5, 6] to achieve name agreement between different attribute names. It is also possible that different CAs use different encoding methods to convert an attribute value to an integer. To address this problem, each CA publishes its encoding methods online and signs them using its private key. When

Alice shows her OACert to Bob, she also sends to Bob the encoding methods for her attributes signed by her CA. Bob can then adjust his policy based on the encoding methods. For example, in the digital driver license issued by BMV, birth-date field is encoded using the number of days between January 1st of 1900 and the actual date of birth. Suppose Bob's policy is that Alice's age must be between 30 and 40, Bob can convert his policy to be that the value of birth-date in Alice's OACert is between a and b, where a and b are birth-date values corresponding to age 30 and age 40, respectively.

3.2 Applications of OACerts

In additional to enabling oblivious access control, OACerts and OCBE are useful in the following settings.

Break policy cycles OACerts and OCBE can be used to break policy cycles (see [45] for definition) in automated trust negotiation [7,9,10,46]. Consider the following scenario where Alice and Bob want to exchange their salary certificates. Alice's policy says that she can show her salary certificate only to those whose salary is great than $100k. Similarly, Bob will reveal his certificate only to those who earn more than $80k a year. Using current trust negotiation techniques, neither Alice nor Bob is willing to present her/his certificate first. The technique developed in [45] does not work well here neither, because the salary requirement in the policies is a range, not a specific value. Such problem can be solved using OACerts and OCBE. Suppose both Alice and Bob use OACerts as their salary certificates, Alice and Bob can first exchange their OACerts without revealing their salary values, then Bob uses an OCBE scheme to send Alice his salary value together with a non-interactive proof that the value sent is indeed the value committed in the OACerts, on the condition that Alice can open them (*i.e.*, the value and the proof) only if her salary is more than $80k. Bob is certain that his salary figure is revealed to Alice only if Alice's income is more than $80k, thus Bob's policy is enforced without him knowing Alice's salary value.

Improve the efficiency of trust negotiation The goal of automated trust negotiation [7, 9, 10, 46] is to establish trust between strangers through interactive disclosure of certificates. OACerts and OCBE can simplify the trust negotiation process by reducing the rounds of interactions and the number of certificates exchanged. Consider the following scenario where Bob is a web publisher and Alice is a senior citizen who wants to get access to Bob's resource. Bob's policy requires Alice to be older than 60. On the other hand, Alice only shows her birth-date to those who are a member of Better Business Bureau (BBB). Using traditional trust negotiation techniques, Bob first shows his BBB certificate, then Alice reveals her driver license, finally Bob sends Alice his resource. The negotiation could be more complicated (and take more rounds) if there is an access control policy for Bob's BBB certificate. Using OACerts, the trust between Alice and Bob can be established in one round – Bob sends his resource using an OCBE protocol such that Alice can receive the resource if and only if she is a senior citizen.

3.3 Definition of OCBE

We now give a formal definition of OCBE. While the definition follows the usage scenario described in Section 3.1 in general, it abstracts away some of the details in the scenario that have been solved using OACerts and focuses on the parts that still need to solved by the OCBE protocol.

Definition 3.3.1 (OCBE) An Oblivious Commitment-Based Envelope (OCBE) scheme is parameterized by a commitment scheme commit. It involves a sender S, a receiver R, and a trusted CA, and has the following phases:

CA-Setup CA takes a security parameter t and outputs the following: the public parameters Params for commit, a set $\mathcal{V}$ of possible values, and a set $\mathcal{P}$ of predicates. Each predicate in $\mathcal{P}$ maps an element in $\mathcal{V}$ to either true or false. The domain of $\mathsf{commit}[\mathsf{Params}]$ contains $\mathcal{V}$ as a subset.

CA-Commit R chooses a value $a \in \mathcal{V}$ (R's attribute value) and sends to CA. CA picks a random number r and computes the commitment $c = \mathsf{commit}_{\mathsf{Params}}(a, r)$. CA gives c and r to R, and c to S.

Recall that in the actual usage scenario, CA does not directly communicate with R. Instead, CA stores the commitment c in R's OACert certificate. The certificate is then sent by R to S, enabling S to have c as if it is sent from CA. Here we abstract these steps away to have CA sending c to S. We stress that CA does *not* participate in the interactions between S and R.

Initialization S chooses a message $M \in \{0, 1\}^*$. S and R agree[1] on a predicate $\mathsf{Pred} \in \mathcal{P}$.

Now S has Pred, c, and M. R has Pred, c, a, and r.

Interaction S and R run an interactive protocol, during which an envelope containing an encryption of M is delivered from S to R.

Open After the interaction phase, if $\mathsf{Pred}(a)$ is true, R outputs the message M; otherwise, R does nothing.

Let an *adversary* be a probabilistic interactive Turing Machine [47]. An OCBE scheme must satisfy the following three properties. It must be sound, oblivious, and semantically secure against the receiver.

Sound An OCBE scheme is *sound* if in the case that $\mathsf{Pred}(a)$ is true, the receiver can output the message M with overwhelming probability, *i.e.*, the probability that the receiver cannot output M is negligible.

Oblivious An OCBE scheme is *oblivious* if the sender learns nothing about a, *i.e.*, no adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the game described

[1] The main effect of having both the sender and the receiver to affect the predicate is that in the security definitions both an adversarial sender and an adversarial receiver can choose the predicate they want to attack on.

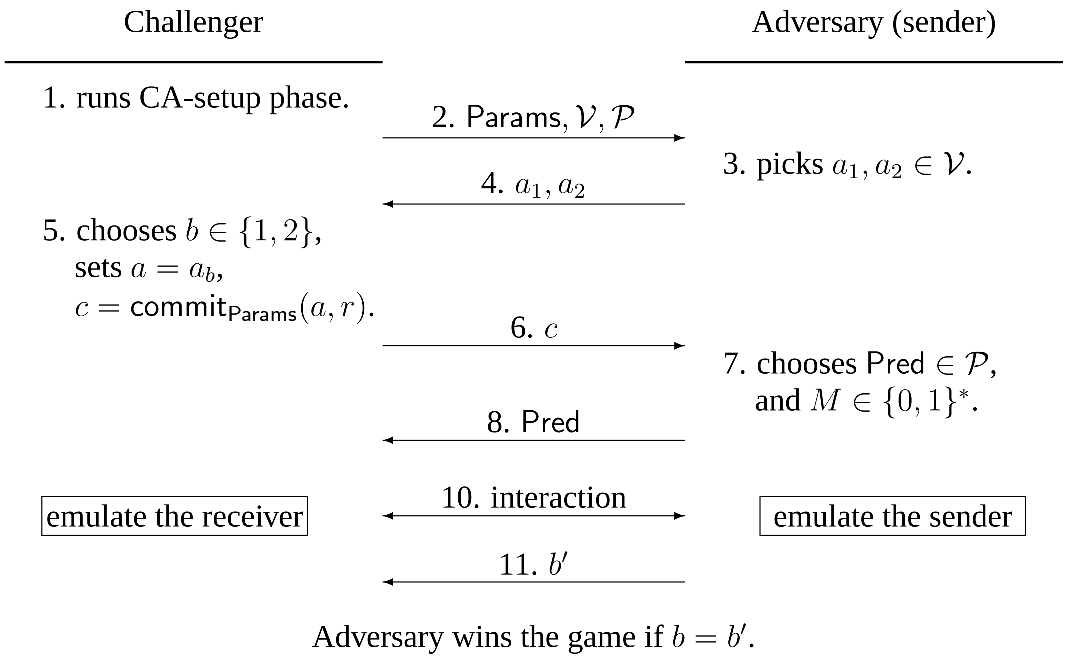
Challenger
Adversary (sender)
1. runs CA-setup phase.
2. Params, $\mathcal{V}, \mathcal{P}$
3. picks $a_1, a_2 \in \mathcal{V}$.
4. a_1, a_2
5. chooses $b \in \{1, 2\}$,
sets $a = a_b$,
$c = \mathsf{commit}_{\mathsf{Params}}(a, r)$.
6. c
7. chooses $\mathsf{Pred} \in \mathcal{P}$,
and $M \in \{0, 1\}^*$.
8. Pred
emulate the receiver
10. interaction
emulate the sender
11. b'
Adversary wins the game if $b = b'$.

Figure 3.1. The attacker game for OCBE's oblivious property. We allow the adversary to pick a predicate Pred and two attribute values a_1, a_2 of her choice; yet the adversary still should not be able to distinguish a receiver with attribute a_1 from one with attribute a_2.

in Figure 3.1 where the challenger emulates CA and the receiver, and the adversary emulates the sender. In other words, an OCBE scheme is *oblivious* if for every probabilistic interactive Turing Machine $\mathcal{A}$, $|\ \Pr[\mathcal{A}$ wins the game in Figure 3.1$] - \frac{1}{2}\ | \leq f(t)$, where f is a negligible function in t.

Secure against the receiver An OCBE scheme is *secure against the receiver* if the receiver learns nothing about M when $\mathsf{Pred}(a)$ is false, *i.e.*, no adversary $\mathcal{A}$ has a non-negligible advantage against the challenger in the game described in Figure 3.2 where the challenger emulates CA and the sender, and the adversary emulates the receiver.

We now argue that OCBE is an adequate solution to the two-party SFE problem in Problem 1, by showing intuitively that the security properties defined for OCBE suffice to prove that the scheme protects the privacy of the participants in the malicious model [26]. Observe that our definitions allow arbitrary adversaries, rather than just those following

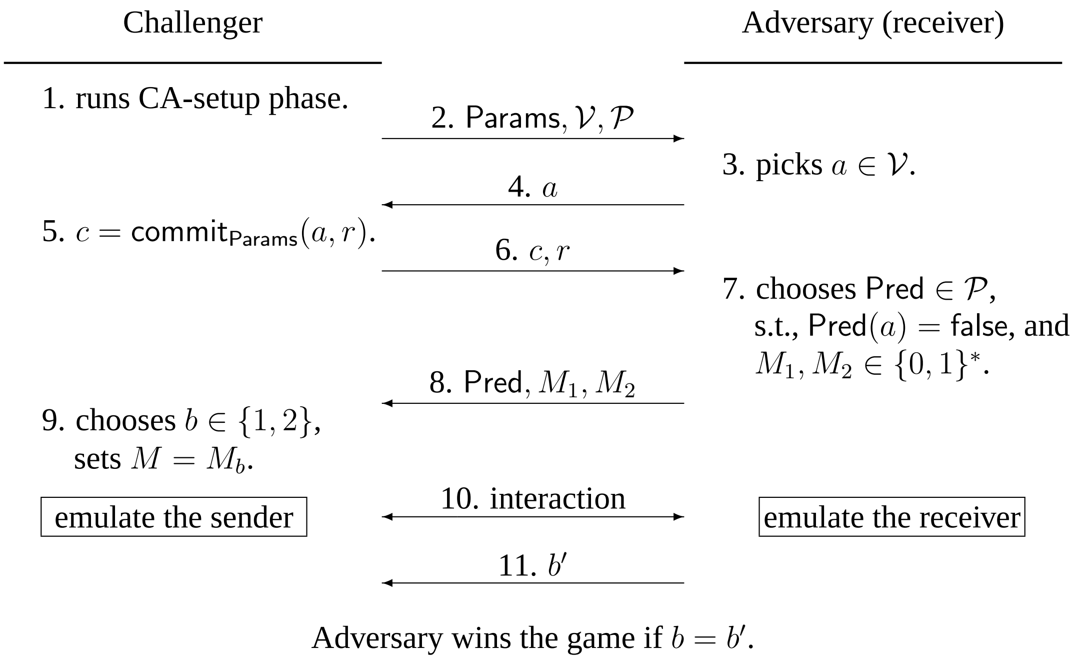
Challenger
Adversary (receiver)
1. runs CA-setup phase.
2. $\mathsf{Params}, \mathcal{V}, \mathcal{P}$
3. picks $a \in \mathcal{V}$.
4. a
5. $c = \mathsf{commit}_{\mathsf{Params}}(a, r)$.
6. c, r
7. chooses $\mathsf{Pred} \in \mathcal{P}$, s.t., $\mathsf{Pred}(a) = \mathsf{false}$, and $M_1, M_2 \in \{0,1\}^*$.
8. Pred, M_1, M_2
9. chooses $b \in \{1,2\}$, sets $M = M_b$.
emulate the sender
10. interaction
emulate the receiver
11. b'
Adversary wins the game if $b = b'$.

Figure 3.2. The attacker game for OCBE's security property against the receiver. Even if we give the adversary the power to pick two equal-length messages M_1 and M_2 of her choice, she still cannot distinguish an envelope containing M_1 from one containing M_2. This formalizes the intuitive notion that the envelope leaks no information about its content.

the protocol (semi-honest adversaries). The oblivious property guarantees that the sender's view of any protocol run can be simulated using just the sender's input, because one can simulate a protocol run between the sender and receiver, and no polynomially bounded sender can figure out the receiver's input. Soundness and security against the receiver guarantee that the receiver's view can be simulated using just the receiver's input and output. If the receiver's committed value a satisfies Pred, then the message M is in the output, one can therefore simulates the sender S. If the receiver's committed value a does not satisfy Pred, one can simulate the sender with a arbitrary message M' and no polynomially bounded receiver can tell the difference.

The security properties defined for OCBE guarantee also the correctness [26] of the OCBE protocol against malicious receivers. Our security definitions do not cover the correctness of the protocol against malicious senders, *i.e.*, if the receiver's value does not satisfy the predicate, a malicious sender may trick the receiver to output the message M which violates the correctness of the protocol[2]. However, this malicious behavior does not make sense in the applications. If a malicious sender does not want to send the message M, she can choose not to participate in the protocol; on the other hand, if a malicious sender wants the receiver to see M without satisfying her policy; she can choose to send M directly rather than participating in the protocol.

We assume that the interaction phase of the OCBE scheme is executed on top of a previously established private communication channel between the sender and the receiver. Recall that the certificate holder establishes an SSL channel with the service provider using OACerts described in Section 3.1.

Note that the OCBE scheme itself does not have the non-transferability property. That is, a legitimate receiver, whose attribute value satisfies a sender's predicate, can share the values a, r, and c to others so that a non-legitimate receiver who knows a, r, and c can successfully obtain the sender's message. However, we stress that the OCBE protocol should always be used together with the disclosure of OACerts (see Section 3.1 for the usage of OACerts). In other words, the receiver has to show that c is certified in his

[2] In such case, the views of the sender and receiver cannot be simulated in the ideal model.

OACerts and that he owns the OACerts. In order for a non-legitimate receiver to access the sender's message, the non-legitimate receiver has to know not only a, r, c from a legitimate receiver but also the private key to the legitimate receiver's OACert. Therefore, non-transferability is guaranteed in our scheme.

3.4 OCBE Protocols

In this section, we present two OCBE protocols using the Pedersen commitment scheme, one for equality predicates, the other for greater-than-or-equal-to predicates. We then sketch how to construct OCBE protocols for other comparison predicates. All arithmetic in this section is assumed to be $\bmod p$ unless otherwise specified.

3.4.1 EQ-OCBE: An OCBE Protocol For $=$ Predicates

Our EQ-OCBE protocol runs a Diffie-Hellman style key-agreement protocol [48] with the twist that the receiver can compute the shared secret if and only if the receiver's committed value a is equal to a_0.

Protocol 1 (EQ-OCBE) Let $\mathcal{E}$ be a semantically secure symmetric encryption scheme with keyspace $\{0,1\}^s$. Let $H : G_q \rightarrow \{0,1\}^s$ be a cryptographic hash function that extracts a key for $\mathcal{E}$ from an element in the group G_q, the order-q subgroup of $\mathbb{Z}_p^*$. EQ-OCBE involves a sender S, a receiver R, and a trust CA.

CA-Setup CA takes a security parameter t and runs the setup algorithm of the Pedersen commitment scheme to create $\mathsf{Params} = \langle p, q, g, h \rangle$. CA also outputs $\mathcal{V} = \mathbb{Z}_q$ and $\mathcal{P} = \{\mathsf{EQ}_{a_0} \mid a_0 \in \mathcal{V}\}$, where $\mathsf{EQ}_{a_0}\colon \mathcal{V} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ is a predicate such that $\mathsf{EQ}_{a_0}(a)$ is true if $a = a_0$ and false if $a \neq a_0$.

CA-Commit R chooses an integer $a \in \mathcal{V}$ and sends to CA. CA picks $r \leftarrow \mathbb{Z}_q$ and computes the commitment $c = g^a h^r$. CA gives c and r to R, and c to S.

Initialization S chooses a message $M \in \{0,1\}^*$. S and R agree on a predicate $\mathsf{EQ}_{a_0} \in \mathcal{P}$.

Now S has EQ_{a_0}, c, and M. R has EQ_{a_0}, c, a, and r.

Interaction S picks $y \leftarrow \mathbb{Z}_q^*$, computes $\sigma = (cg^{-a_0})^y$, and then sends to R the pair $\langle \eta = h^y, C = \mathcal{E}_{H(\sigma)}[M] \rangle$.

Open R receives $\langle \eta, C \rangle$ from the interaction phase. If $\mathsf{EQ}_{a_0}(a)$ is true, R computes $\sigma' = \eta^r$, and decrypts C using $H(\sigma')$.

To see that EQ-OCBE is sound, observe that when $\mathsf{EQ}_{a_0}(a)$ is true,

$$\sigma = (cg^{-a_0})^y = (g^a h^r g^{-a_0})^y = (g^{a-a_0} h^r)^y = (h^r)^y = (h^y)^r = \eta^r = \sigma'.$$

Therefore the sender and receiver share the same symmetric key.

Also observe that the interaction phase of the EQ-OCBE protocol is one-round; it involves only one message from the sender to the receiver. In the interaction and open phases, the sender does two exponentiations and the receiver does one exponentiation.

The key idea of EQ-OCBE is that if the receiver's committed value a is equal to a_0, the sender can compute $cg^{-a_0} = g^{a-a_0}h^r = h^r$. The sender now holds h^r such that the receiver knows the value r. This achieves half of the Diffie-Hellman key-agreement protocol [48], with h as the base. The sender then does the other half by sending h^y to the receiver. Thus both the sender and receiver can compute $\sigma = (cg^{-a_0})^y = h^{ry}$. If the receiver's committed value a is not equal to a_0, then it is presumably hard for him to compute $\sigma = (cg^{-a_0})^y$ from h^y and cg^{-a_0}. The receiver cannot effectively compute $\log_h(cg^{-a_0})$, because if the receiver is able to find a number $r' = \log_h(cg^{-a_0})$, he can break the binding property of the commitment scheme, *i.e.*, he finds a (a_0, r') pair such that $g^{a_0}h^{r'} = g^a h^r$.

Theorem 3.4.1 *EQ-OCBE is oblivious.*

Proof The interaction phase involves only one message from the sender to the receiver. Among what the sender sees, the only piece of information that is related to the receiver's attribute value a is the commitment c. As the Pedersen commitment scheme is unconditionally hiding; c does not leak *any* information about a. Thus EQ-OCBE is oblivious even against an infinitely powerful adversary. ■

Theorem 3.4.2 *Under the CDH assumption on G_q, the order-q subgroup of $\mathbb{Z}_p^*$, and when H is modeled as a random oracle, EQ-OCBE is secure against the receiver.*

Proof EQ-OCBE uses a semantically secure symmetric encryption algorithm. When H is modeled as a random oracle, EQ-OCBE is secure against the receiver when no receiver whose committed value is not equal to a_0 can compute with non-negligible probability $\sigma = (cg^{-a_0})^y$, the secret that the sender uses to derive the encryption key. More precisely, EQ-OCBE is secure against the receiver if no polynomial-time adversary wins the following game against the challenger with non-negligible probability (this game is instantiated from the game in Figure 3.2 with details from the EQ-OCBE protocol): The challenger runs the setup phase and sends $\mathsf{Params} = \langle p, q, g, h \rangle$ and the descriptions of $\mathcal{V}$ and $\mathcal{P}$ to the adversary. The adversary picks an integer $a \in \mathcal{V}$. The challenger chooses $r \leftarrow \mathbb{Z}_q$ and computes the commitment of a as $c = g^a h^r$, and gives r and c to the adversary. The adversary responds with an equality predicate EQ_{a_0} such that $\mathsf{EQ}_{a_0}(a)$ is false. The challenger then picks $y \leftarrow \mathbb{Z}_q^*$ and sends to the adversary h^y. The adversary then outputs σ, and the adversary wins the game if $\sigma = (cg^{-a_0})^y$.

Given an attacker $\mathcal{A}$ that wins the above game with probability ϵ, we construct another attacker $\mathcal{B}$ that solves the CDH problem in G_q with the same probability. $\mathcal{B}$ does the following:

1. $\mathcal{B}$, when given $p, q, h \in G_q, h^x, h^y$, gives $\mathsf{Params} = \langle p, q, h^x, h \rangle$ and the descriptions of $\mathcal{V} = \mathbb{Z}_q$ and $\mathcal{P} = \{\mathsf{EQ}_{a_0} \mid a_0 \in \mathcal{V}\}$ to $\mathcal{A}$. Let g denote h^x.

2. $\mathcal{B}$ receives an integer $a \in \mathbb{Z}_q$ from $\mathcal{A}$, picks $r \leftarrow \mathbb{Z}_q$, computes $c = (h^x)^a h^r$, and sends r and c to $\mathcal{A}$.

3. $\mathcal{B}$ receives an equality predicate EQ_{a_0} from $\mathcal{A}$ where $a \neq a_0$, and sends h^y to $\mathcal{A}$.

4. $\mathcal{B}$ receives σ from $\mathcal{A}$, computes $\delta = \sigma h^{-ry}$, and outputs $\delta^{(a-a_0)^{-1} \bmod q}$.

   When $\mathcal{A}$ wins the game, $\sigma = (cg^{-a_0})^y = (g^{a-a_0}h^r)^y = (g^y)^{a-a_0}h^{ry}$, then $\delta = \sigma h^{-ry} = (g^y)^{a-a_0} = (h^{xy})^{a-a_0}$. $\mathcal{B}$ outputs $\delta^{(a-a_0)^{-1} \bmod q} = h^{xy}$.

$\mathcal{B}$ succeeds in solving the CDH problem if $\mathcal{A}$ wins the above game, *i.e.*, successfully computes $(cg^{-a_0})^y$. ■

3.4.2 GE-OCBE: An OCBE Protocol For $\geq$ Predicates

In this section, we present an OCBE protocol (GE-OCBE) for the Pedersen commitment scheme with greater-than-or-equal-to predicates. The basic idea of the GE-OCBE protocol is as follows. Let ℓ be an integer such that $2^\ell < q/2$. Let a and a_0 be two numbers in $[0..2^\ell - 1]$, and let $d = ((a - a_0) \bmod q)$. Let $c = g^a h^r$ be a commitment of a where r is known to the receiver, then $cg^{-a_0} = g^{a-a_0}h^r = g^d h^r$ is a commitment of d that the receiver knows how to open. Notice that if $a \geq a_0$ then $d \in [0..2^\ell - 1]$, otherwise $d \notin [0..2^\ell - 1]$.

If $a \geq a_0$, the receiver generates ℓ new commitments $c_0, \ldots, c_{\ell-1}$, one for each of the ℓ bits of d. The sender picks a random encryption key k and split it into ℓ secrets $k_0, \ldots, k_{\ell-1}$. Then the sender and receiver run a "bit-OCBE" protocol for each commitment, *i.e.*, if c_i is a bit-commitment, the receiver obtains k_i, otherwise he gets nothing, while the sender learns nothing about the value committed under c_i.

Protocol 2 (GE-OCBE) Let $\mathcal{E}$ be a semantically secure symmetric encryption scheme with keyspace $\{0,1\}^s$. Let $H : G_q \rightarrow \{0,1\}^s$ and $H' : \{0,1\}^{s\ell} \rightarrow \{0,1\}^s$ be two cryptographic hash functions. Our GE-OCBE protocol involves a sender S, a receiver R, and a trust CA.

CA-Setup CA takes two parameters, a security parameter t and a parameter ℓ (which specifies the desired range of the attribute values). CA runs the setup algorithm of the Pedersen commitment scheme to create $\mathsf{Params} = \langle p, q, g, h \rangle$ such that $2^\ell < q/2$. CA also outputs $\mathcal{V} = [0..2^\ell - 1]$ and $\mathcal{P} = \{\mathsf{GE}_{a_0} \mid a_0 \in \mathcal{V}\}$, where $\mathsf{GE}_{a_0} \colon \mathcal{V} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ is a predicate such that $\mathsf{GE}_{a_0}(a)$ is true if $a \geq a_0$ and false otherwise.

CA-Commit R chooses an integer $a \in \mathcal{V}$ and sends to CA. CA picks $r \leftarrow \mathbb{Z}_q$ and computes the commitment $c = g^a h^r$. CA gives c and r to R, and c to S.

Initialization S chooses a message $M \in \{0,1\}^*$. S and R agree on a predicate $\mathsf{GE}_{a_0} \in \mathcal{P}$.

Now S has GE_{a_0}, c, and M. R has GE_{a_0}, c, a, and r.

Interaction Let $d = ((a - a_0) \bmod q)$, $\mathsf{GE}_{a_0}(a) = \mathsf{true}$ if and only if $d \in [0..2^\ell - 1]$. Note that $cg^{-a_0} = g^d h^r$ is a commitment of d that R can open.

1. R picks $r_1, \ldots, r_{\ell-1} \leftarrow \mathbb{Z}_q$ and sets $r_0 = r - \sum_{i=1}^{\ell-1} 2^i r_i \bmod q$. When $\mathsf{GE}_{a_0}(a) = \mathsf{true}$, let $d_{\ell-1} \ldots d_1 d_0$ be the binary representation of d, *i.e.*, $d = d_0 2^0 + d_1 2^1 + \cdots + d_{\ell-1} 2^{\ell-1}$. When $\mathsf{GE}_{a_0}(a) = \mathsf{false}$, R randomly picks $d_1, d_2, \ldots, d_{\ell-1} \leftarrow \{0,1\}$, and sets $d_0 = d - \sum_{i=1}^{\ell-1} 2^i d_i \bmod q$. R computes, for $0 \le i \le \ell - 1$, the commitment $c_i = \mathsf{commit}(d_i, r_i) = g^{d_i} h^{r_i}$. R sends $c_0, \ldots, c_{\ell-1}$ to S.

2. S verifies that $cg^{-a_0} = \prod_{i=0}^{\ell-1} (c_i)^{2^i}$. S randomly chooses ℓ symmetric keys $k_0, \ldots, k_{\ell-1} \in \{0,1\}^t$ and sets $k = H'(k_0 || \cdots || k_{\ell-1})$. S picks $y \leftarrow \mathbb{Z}_q^*$, computes $\eta = h^y$ and $C = \mathcal{E}_k[M]$. For each $0 \le i \le \ell - 1$, S computes $\sigma_i^0 = (c_i)^y$, $\sigma_i^1 = (c_i g^{-1})^y$, $C_i^0 = H(\sigma_i^0) \oplus k_i$, and $C_i^1 = H(\sigma_i^0) \oplus k_i$. S sends to R the tuple $\langle \eta, C_0^0, C_0^1, \ldots, C_{\ell-1}^0, C_{\ell-1}^1, C \rangle$.

Open R receives $\langle \eta, C_0^0, C_0^1, \ldots, C_{\ell-1}^0, C_{\ell-1}^1, C \rangle$ from the interaction phase. If $\mathsf{GE}_{a_0}(a)$ is true, $d = \sum_{i=0}^{\ell-1} 2^i d_i$ where $d_i \in \{0,1\}$. For each $0 \le i \le \ell - 1$, R computes $\sigma_i' = \eta^{r_i}$, and obtains $k_i' = H(\sigma_i') \oplus C_i^{d_i}$. R then computes $k' = H'(k_0' || \cdots || k_{\ell-1}')$, and decrypts C using k'.

To see that the GE-OCBE protocol is sound, observe that when $\mathsf{GE}_{a_0}(a)$ is true, $d_0, \ldots, d_{\ell-1}$ are either 0 or 1. If the receiver follows the protocol, the sender will succeed in verifying $\prod_{i=0}^{\ell-1} (c_i)^{2^i} = \prod_{i=0}^{\ell-1} (g^{d_i} h^{r_i})^{2^i} = g^d h^r = cg^{-a_0}$. For each $0 \le i \le \ell - 1$, if $d_i = 0$, $\sigma_i^0 = (c_i)^y = (g^{d_i} h^{r_i})^y = (h^y)^{r_i} = \eta^{r_i} = \sigma_i'$, the receiver can compute $k_i = C_i^0 \oplus H(\sigma_i')$; if $d_i = 1$, $\sigma_i^1 = (c_i g^{-1})^y = (g^{d_i - 1} h^{r_i})^y = (h^y)^{r_i} = \eta^{r_i} = \sigma_i'$, the receiver can compute $k_i = C_i^1 \oplus H(\sigma_i')$. As $k = H'(k_0 || \cdots || k_{\ell-1})$, the receiver can successfully obtain k. Thus the sender and receiver share the same symmetric key k if $\mathsf{GE}_{a_0}(a)$ is true.

The interaction phase of the GE-OCBE protocol is two rounds. The receiver does about 2ℓ exponentiations. The sender does about ℓ exponentiations (observe that σ_i^1 can be computed as $\sigma_i^0 g^{-y}$, where g^{-y} needs to be computed only once).

We briefly sketch the idea why the receiver cannot obtain M if $\mathsf{GE}_{a_0}(a)$ is false. If the receiver follows the protocol, then $d_1, \ldots, d_{\ell-1} \in \{0,1\}$ and $d_0 \notin \{0,1\}$. The receiver can successfully compute $k_1, \ldots, k_{\ell-1}$, but fails to compute k_0 because he can compute neither $\sigma_0^0 = (c_0)^y = (g^{d_0} h^r)^y$ nor $\sigma_0^1 = (c_0 g^{-1})^y = (g^{d_0 - 1} h^r)^y$. Even if the receiver does not follow the protocol, it is impossible for him to find $d_0, \ldots, d_{\ell-1} \in \{0,1\}$ and $r_0, \ldots, r_{\ell-1}$ such that $cg^{-a_0} = \prod_{i=0}^{\ell-1} (c_i)^{2^i}$ and $c_i = g^{d_i} h^{r_i}$. Suppose the receiver finds such $d_0, \ldots, d_{\ell-1} \in \{0,1\}$ and $r_0, \ldots, r_{\ell-1}$; let $d' = \sum_{i=0}^{\ell-1} d_i 2^i \in [0..2^\ell - 1]$ and $r' = \sum_{i=0}^{\ell-1} r_i 2^i \pmod q$, then

$$g^{a-a_0} h^r = cg^{-a_0} = \prod_{i=0}^{\ell-1} (c_i)^{2^i} = \prod_{i=0}^{\ell-1} (g^{d_i} h^{r_i})^{2^i} = g^{\sum_{i=0}^{\ell-1} d_i 2^i} h^{\sum_{i=0}^{\ell-1} r_i 2^i} = g^{d'} h^{r'}.$$

As $a - a_0 \notin [0..2^\ell - 1]$ and $d' \in [0..2^\ell - 1]$, $d' \neq a - a_0$, the receiver is able to find $a - a_0$, r, d', and r' such that $g^{a-a_0} h^r = g^{d'} h^{r'}$, which breaks the binding property of the Pedersen commitment scheme.

Theorem 3.4.3 *GE-OCBE is oblivious.*

Proof Consider the game for the oblivious property of OCBE (in Figure 3.1), let us examine what an adversary would see in the case of GE-OCBE. The adversary sees a commitment c and ℓ commitments $c_0, \ldots, c_{\ell-1}$ such that $cg^{-a_0} = \prod_{i=0}^{\ell-1} (c_i)^{2^i}$. The joint distribution of $c, c_0, \ldots, c_{\ell-1}$ is independent of whether the challenger picked a_0 or a_1, as $c, c_1, \ldots, c_{\ell-1}$ are totally random (because of the random choices of $r, r_1, \ldots, r_{\ell-1}$), and c_0 is always equal to $cg^{-a_0} \prod_{i=1}^{\ell-1} (c_i)^{-2^i}$. GE-OCBE is oblivious even against an infinitely powerful adversary. ■

Theorem 3.4.4 *Under the CDH assumption on G_q, the order-q subgroup of $\mathbb{Z}_p^*$, and when H and H' are modeled as random oracles, GE-OCBE is secure against the receiver.*

Proof GE-OCBE uses a semantically secure symmetric encryption algorithm. When H' is modeled as a random oracle, EQ-OCBE is secure against the receiver when no receiver whose committed value a does not satisfy GE_{a_0} can compute with non-negligible probability $k_0||\ldots||k_{\ell-1}$, the secret that the sender uses to derive the encryption key k. In other words, if $\mathsf{GE}_{a_0}(a)$ is false, we need to show that no receiver can compute $k_0, \ldots, k_{\ell-1}$ with non-negligible probability. Recall that the receiver is given $C_i^0 = H(\sigma_i^0) \oplus k_i$ and $C_i^1 = H(\sigma_i^1) \oplus k_i$, when H is also modeled as a random oracle, the receiver has to know either σ_i^0 or σ_i^1 to recover k_i.

GE-OCBE is secure against the receiver if no polynomial-time adversary wins the following game against the challenger with non-negligible probability (this game is instantiated from the game in Figure 3.2 with details from the GE-OCBE protocol): The challenger runs the setup phase and sends $\mathsf{Params} = \langle p, q, g, h\rangle$ and the descriptions of $\mathcal{V}$ and $\mathcal{P}$ to the adversary. The adversary picks an integer $a \in \mathcal{V}$. The challenger chooses $r \leftarrow \mathbb{Z}_q$ and computes the commitment of a as $c = g^a h^r$, and gives r and c to the adversary. The adversary responds with a greater-than-or-equal-to predicate GE_{a_0} such that $\mathsf{GE}_{a_0}(a)$ is false. The adversary outputs ℓ commitments $c_0, \ldots, c_{\ell-1}$ such that $cg^{-a_0} = \prod_{i=0}^{\ell-1}(c_i)^{2^i}$. The challenger then picks $y \leftarrow \mathbb{Z}_q^*$ and sends to the adversary h^y. The adversary then outputs $\sigma_0, \ldots, \sigma_{\ell-1}$ and $d_0, \ldots, d_{\ell-1} \in \{0, 1\}$, and the adversary wins the game if each $0 \leq i \leq \ell - 1$, $\sigma_i = (c_i g^{-d_i})^y$ holds.

Given an attacker $\mathcal{A}$ that wins the above game with probability ϵ, we construct another attacker $\mathcal{B}$ that solves the CDH problem in G_q with the same probability. $\mathcal{B}$ does the following:

1. $\mathcal{B}$, when given $p, q, h \in G_q, h^x, h^y$, gives $\mathsf{Params} = \langle p, q, h^x, h\rangle$ and the descriptions of $\mathcal{V} = \mathbb{Z}_q$ and $\mathcal{P} = \{\mathsf{GE}_{a_0} \mid a_0 \in \mathcal{V}\}$ to $\mathcal{A}$. Let g denote h^x.

2. $\mathcal{B}$ receives an integer $a \in \mathbb{Z}_q$ from $\mathcal{A}$, picks $r \leftarrow \mathbb{Z}_q$, computes $c = (h^x)^a h^r$, and sends r and c to $\mathcal{A}$.

3. $\mathcal{B}$ receives a great-than-or-equal-to predicate GE_{a_0} from $\mathcal{A}$ where $a < a_0$. $\mathcal{B}$ computes $d = ((a - a_0) \bmod q)$.

4. $\mathcal{B}$ receives ℓ commitments $c_0, \ldots, c_{\ell-1}$ where $cg^{-a_0} = \prod_{i=0}^{\ell-1}(c_i)^{2^i}$, and sends h^y to $\mathcal{A}$.

5. $\mathcal{B}$ receives $\sigma_0, \ldots, \sigma_{\ell-1}$, and $d_0, \ldots, d_{\ell-1}$ from $\mathcal{A}$. $\mathcal{B}$ computes $\delta = \prod_{i=0}^{\ell-1} (\sigma_i)^{2^i}$ and $d' = \sum_{i=0}^{\ell-1} d_i 2^i$, and outputs $(\delta h^{-ry})^{(d-d')^{-1} \bmod q}$.

   When $\mathcal{A}$ wins the game, $\sigma_i = (c_i g^{-d_i})^y$, then

$$\begin{aligned} \delta &= \prod_{i=0}^{\ell-1} (\sigma_i)^{2^i} = \prod_{i=0}^{\ell-1} ((c_i g^{-d_i})^y)^{2^i} \\ &= (g^{-d'} \prod_{i=0}^{\ell-1} (c_i)^{2^i})^y = (g^{-d'} c g^{-a_0})^y = (g^{d-d'} h^r)^y = g^{(d-d')y} h^{ry}. \end{aligned}$$

   $\mathcal{B}$ outputs $(\delta h^{-ry})^{(d-d')^{-1} \bmod q} = (g^{(d-d')y})^{(d-d')^{-1} \bmod q} = g^y = h^{xy}$.

$\mathcal{B}$ succeeds in solving the CDH problem if $\mathcal{A}$ wins the above game, *i.e.*, successfully computes $(c_0 g^{-d_0})^y, \ldots, (c_{\ell-1} g^{-d_{\ell-1}})^y$, where $cg^{-a_0} = \prod_{i=0}^{\ell-1} (c_i)^{2^i}$, and $d_0, \ldots, d_{\ell-1} \in \{0, 1\}$. ■

3.4.3 OCBE Protocols for Other Predicates

In this section, we first present two logical combination OCBE protocols, one for $\wedge$ (AND-OCBE), the other for $\vee$ (OR-OCBE). Then we describe OCBE protocols for comparison predicates: $>$ (GT-OCBE), $\leq$ (LE-OCBE), $<$ (LT-OCBE), $\neq$ (NE-OCBE). Finally, we present an OCBE protocol for range predicates (RANGE-OCBE). Instead of formally presenting these protocols, we briefly sketch the ideas. We use $OCBE(\mathsf{Pred}, a, M)$ to denote an OCBE protocol with predicate Pred and committed value a, the receiver outputs M if $\mathsf{Pred}(a)$ is true. Similar techniques have been used before in [30, 49].

1. **AND-OCBE**: Suppose there exists OCBE protocols for Pred_1 and Pred_2, the goal is to build an OCBE protocol for the new predicate $\mathsf{Pred} = \mathsf{Pred}_1 \wedge \mathsf{Pred}_2$. An $OCBE(\mathsf{Pred}_1 \wedge \mathsf{Pred}_2, a, M)$ can be constructed as follows: In the interaction phase, the sender picks two random keys k_1 and k_2 and sets $k = H(k_1 || k_2)$, where H is a cryptographic hash function. The sender then runs the interaction phases of $OCBE(\mathsf{Pred}_1, a, k_1)$ and $OCBE(\mathsf{Pred}_2, a, k_2)$ with the receiver. Finally, the sender sends $\mathcal{E}_k[M]$ to the receiver. The receiver can recover M in the open phase only if both $\mathsf{Pred}_1(a)$ and $\mathsf{Pred}_2(a)$ are true.

2. **OR-OCBE**: An $OCBE(\mathsf{Pred}_1 \vee \mathsf{Pred}_2, M)$ can be constructed as follows: In the interaction phase, the sender picks a random key k. The sender then runs the interaction phases of $OCBE(\mathsf{Pred}_1, a, k)$ and $OCBE(\mathsf{Pred}_2, a, k)$ with the receiver. Finally, the sender sends $\mathcal{E}_k[M]$ to the receiver. The receiver can recover M in the open phase if either $\mathsf{Pred}_1(a)$ or $\mathsf{Pred}_2(a)$ is true.

3. **GT-OCBE**: For integer space, $a > a_0$ is equivalent to $a \geq a_0 + 1$. An $OCBE(>_{a_0}, a, M)$ protocol is equivalent to an $OCBE(\geq_{a_0+1}, a, M)$ protocol.

4. **LE-OCBE**: The idea of LE-OCBE protocol is similar to the GE-OCBE protocol. Observe that $a \leq a_0$ if and only if $d = ((a_0 - a) \bmod q) \in [0..2^\ell - 1]$. Let $c = g^a h^r$ be a commitment of a, then $g^{a_0} c^{-1} = g^{(a_0-a) \bmod q} h^{-r \bmod q}$ is a commitment of d such that the receiver knows how to open. The LE-OCBE protocol uses the same method as in GE-OCBE.

5. **LT-OCBE**: For integer space, $a < a_0$ is equivalent to $a \leq a_0 - 1$. An $OCBE(<_{a_0}, a, M)$ protocol is equivalent to an $OCBE(\leq_{a_0-1}, a, M)$ protocol.

6. **NE-OCBE**: $a \neq a_0$ is equivalent to $(a > a_0) \vee (a < a_0)$. Therefore, an $OCBE(\neq_{a_0}, a, M)$ can be built as $OCBE(>_{a_0} \vee <_{a_0}, a, M)$.

7. **RANGE-OCBE**: $a_0 \leq a \leq a_1$ is equivalent to $(a \geq a_0) \wedge (a \leq a_1)$. Therefore, a RANGE-OCBE can be built as $OCBE(\geq_{a_0} \wedge \leq_{a_1}, a, M)$.

3.4.4 MOCBE: Multi-attribute OCBE

OCBE guarantees that, for the receiver to receive a message, her attribute committed in her OACert must satisfy the sender's policy. In many scenarios, access control policies are based on multiple attributes rather than one. For example, a policy may require that the receiver either has GPA more than 3.0 or is older than 21. This requirement involves two attribute a_1 (GPA) and a_2 (age), and the predicate for the sender is $(a_1 > 3.0) \vee (a_2 > 21)$. It is natural to extend OCBE to support multiple attributes, called MOCBE.

In this subsection, we present constructions of MOCBE for two types of multi-attribute comparison predicates which we believe are useful in practice. Let $\diamond$ denote a comparison operation where $\diamond \in \{=, \neq, <, >, \leq, \geq\}$. Our constructions use the Pedersen commitment scheme and use the OCBE protocols as sub-protocols.

Linear Relation Predicates The linear relation predicates $\mathsf{Pred}(a_1, \ldots, a_n)$ take the form of $a_1b_1 + \cdots + a_nb_n \quad \diamond \quad e$, where $b_1, \ldots, b_n$, and e are public integers from $\mathcal{V}$. In other words, $\mathsf{Pred}(a_1, \ldots, a_n)$ is true if $a_1b_1 + \cdots + a_nb_n \quad \diamond \quad e$ is true, and is false otherwise. The MOCBE protocol of this type of predicates can be built as follows: Since the Pedersen commitment scheme is a homomorphic commitment scheme, the sender and receiver each can compute the commitment of $a_1b_1 + \cdots + a_nb_n$ (denoted as x) by computing $c_1^{b_1}c_2^{b_2} \cdots c_n^{b_n}$ (denote as c). Now both the sender and the receiver have c, the receiver knows how to open the commitment c, and we want the receiver to obtain the sender's message if and only if x (the value committed in c) satisfies $x \diamond e$. We reduce the MOCBE protocol to the OCBE protocols for comparison predicates.

General Comparison Predicates The idea of this construction comes from [49]. The predicate $\mathsf{Pred}(a_1, \ldots, a_n)$ is specified as a boolean circuit with n input and one output, each input i is associate with a predicate $a_i \diamond e_i$ where e_i is an integer in $\mathcal{V}$. The circuit consists of AND gates and OR gates; each gate has two or more inputs and one output. Intuitively, a receiver makes an input true if $a_i \diamond e_i$ is true. A receiver satisfies the predicate if it makes the output of the circuit true. The MOCBE protocol is as follows:

1. For each $i = 1..n$, the sender chooses a random key k_i and runs an OCBE protocol with the receiver, sending k_i in an envelope that can be opened only when $a_i \diamond e_i$ is true.

2. The sender computes the keys associated with (the output of) each gate as follows, starting from the input of the circuit. For an AND gate, let $k^{(1)}, k^{(2)}, \ldots, k^{(m)}$ be the keys associated with the m inputs, then the key corresponding to the output is $k = k^{(1)} \oplus \ldots \oplus k^{(m)}$. For an OR gate, let $k^{(1)}, k^{(2)}, \ldots, k^{(m)}$ be the keys associated

with the m inputs. The sender chooses a random key k as the output key. The sender then encrypts k under each of $k^{(1)}, k^{(2)}, \ldots, k^{(m)}$, and sends the m ciphertexts to the receiver.

3. The sender encrypts the message M using the key associated with the circuit output and sends the ciphertext C to the receiver.

It is not hard to see that if the receiver's attributes $a_1, \ldots, a_n$ satisfy the predicate $\mathsf{Pred}(a_1, \ldots, a_n)$, then the receiver can obtain the key associated with the circuit output. Thus the receiver is able to decrypt C and obtain M.

3.5 Implementation and Performance

We have implemented a toolkit that generates X.509 certificates [28] that are also OACerts using Java v1.4.2 SDK and JCSI PKI Server Library [50]. In our implementation, both the parameters of the Pedersen commitment scheme and commitments of certificate holder's attributes are encoded in the X.509v3 extension fields. Recall that the parameters of the Pedersen commitment scheme are $\langle p, q, g, h \rangle$; they are large integer numbers. The commitments can also be viewed as large integers. We convert each of these integers into an octet string and bind it with an unique object identifier (OID) [28], and place them (octet string and OID) in the extension fields as a non-critical extension. Note that attribute name is not encoded in the certificate. The CA can publish a list of attribute names and their corresponding OID, so that service providers know which commitment corresponds to which attribute. Our OACerts can be recognized by OpenSSL.

We implemented also the OCBE protocols and zero-knowledge proof protocols [35–38] in Java with Java 2 Platform v1.4.2 SDK. We use the Pedersen commitment scheme with security parameters $p = 1024$ bits and $q = 160$ bits. Thus the size of a commitment is 1024 bits, or 128 bytes. We set the attribute values in OACerts to be unsigned long, *i.e.*, $\ell = 32$. In the implementation of the OCBE protocols, we use MD5 as the cryptographic hash function, AES as the symmetric key encryption scheme. Given an arbitrary size message, MD5 outputs a 128-bit message digest. In our setting, M is typically a 16-byte symmetric key, the size of $\mathcal{E}[M]$ is also 16 bytes using AES in ECB mode. In EQ-OCBE, η is 128 bytes (1024 bits) and C is 16 bytes, the total size of communication is 144 bytes. We ran our implementation on a 2.53GHz Intel Pentium 4 machine with 384MB RAM running RedHat Linux 9.0. We simulate the certificate holder and service provider on the same machine. With p of size 1024 bits and q of size 160 bits in the Pedersen commitment scheme, and $\ell = 32$, the performance of two zero-knowledge proof protocols and two OCBE protocols is summarized in Table 3.1.

Table 3.1
Running time and size of communication on a 2.53GHz Intel Pentium 4 running RedHat Linux. Security parameters are $\ell = 32$, $p = 1024$ bits, and $q = 160$ bits.

	execution time	communication size
Zero-knowledge proof that $a = a_0$	28 ms	168 bytes
Zero-knowledge proof that $a \geq a_0$	2.2 s	15 KB
EQ-OCBE	75 ms	144 bytes
GE-OCBE	0.9 s	5.1 KB

4 POLICY-HIDING ACCESS CONTROL USING OACERTS

In attribute-based access control, as attribute information may be sensitive, the certificates that contain attribute data need protection just as other resources do. Often times, the policies for determining who can access the resources are sensitive also and need protection as well. Consider the following example.

Example 2 Bob is a bank offering certain special-rate loans and Alice would like to know whether she is eligible for such a loan before she applies. Alice has a digital driver license certificate issued by the state authority; the certificate contains her birth-date, address, and other attribute data. Alice has also an income certificate issued by her employer documenting her salary and the starting date of her employment. Bob determines whether Alice is eligible for a special-rate loan based on Alice's attribute information. For example, Bob may require that one of the following two conditions holds: (1) Alice is over 30 years old, has an income of no less than $43K, and has been in the current job for over six months; (2) Alice is over 25 years old, has an income of no less than $45K, and has been in the current job for at least one year.

Bob is willing to reveal that his loan-approval policy uses the applicant's birth-date, current salary, and the length of the current employment; however, Bob considers the detail of his policy to be a trade secret and does not want to reveal it to others. Alice is interested in this loan and would like to go forward; however, she wants to reveal as little information about her attributes as possible. In particular, Bob shouldn't learn anything about her address (which is also in her driver license) or learn her actual birth-date. Ideally, Alice wants Bob to know whether she is eligible for the loan, but nothing else.

In the above example, the policy is a commercial secret, and knowledge of Bob's policy would compromise Bob's strategy and invite unwelcome imitators. In other examples, the motivation for hiding the policy is not necessarily protection from an evil adversary, but simply the desire to prevent legitimate users from gaming the system; *e.g.*, changing their behavior based on their knowledge of the policy. This is particularly important for policies that are not incentive-compatible in economic terms.

Motivated by the preceding applications, we introduce and study the problem of *policy-hiding access control*. In this framework, Bob has a private policy and Alice has several sensitive certificates. In the end, Bob learns whether Alice's attributes in her certificates satisfy his policy but nothing else about her attribute values; at the same time, Alice does not learn Bob's policy except for what attributes are required for his policy.

One may tempt to use existing general solutions to the two-party Secure Function Evaluation (2-SFE) [23, 26, 41] (*e.g.*, Yao's scramble circuit protocol [23]) for policy-hiding access control. That is, Alice inputs her certificates and Bob inputs his policy; and they run a 2-SFE protocol to evaluate Bob's policy on Alice's attributes in her certificates. Such approach does not work well because (1) the function to compute in 2-SFE is public, whereas the function (Bob's policy) in policy-hiding access control is private; (2) as Alice needs to input her certificates into 2-SFE, certificate verification, which involves verifying digital signatures, needs to be done as a part of 2-SFE circuit evaluation. This is extremely inefficient. Observe that Alice is not allowed to input her attribute values directly (instead of her certificates), because, Alice otherwise can input arbitrary faked attribute values at her will[1].

To avoid verifying certificates within circuit evaluation, we use OACerts described in Chapter 3. We introduce the notion of *Certified Input Private Policy Evaluation (CIPPE)*, which enables policy-hiding access control using OACerts. Formal definition of CIPPE will be given in Section 4.2. In CIPPE, Alice has private inputs $x_1, x_2, \cdots, x_n$, Bob has a private function f drawn from a family $\mathcal{F}$ of functions (usually f outputs 'yes' or 'no'; however, we allow functions that output more than one bit of information), and Alice and Bob share $c_1, c_2, \cdots, c_n$, where c_i is a cryptographic commitments of x_i, for $1 \leq i \leq n$. The objective of CIPPE is for both Alice and Bob to learn the result of $f(x_1, \cdots, x_n)$.

[1] In SFE, there is no way to prevent a dishonest party from changing its local input before the protocol execution.

Bob should not learn anything about $x_1, \ldots, x_n$; and Alice should not learn more than the fact that $f \in \mathcal{F}$.

We develop a CIPPE protocol for certain families of functions that we believe are useful for expressing policies. Our solution uses Yao's scrambled circuit protocol [23, 51]. When a circuit is scrambled, the operation in each gate is hidden; however, the topological structure of the circuit is not. Therefore, Alice could infer some information about Bob's policy by looking at the scrambled circuit if Bob constructs the circuit in the naive way. To protect Bob's private function, we develop an efficient approach to construct circuits with uniform topology that can compute certain functions families. To ensure that Alice can evaluate the scrambled circuit only with her attribute values as committed in her certificates, we develop an efficient and provably secure Committed-Integer based Oblivious Transfer (CIOT) protocol. The computation and communication complexity of the proposed CIPPE protocol is close to the complexity of the scramble circuit protocol that computes $f(x_1, \ldots x_n)$ where f is public, and $x_1, \ldots, x_n$ are not committed. The CIPPE protocol is efficient; and we believe it can be deployed in practice (see [51] for an implement of the scramble circuit protocol by Malkhi *et al.*).

The rest of this chapter is organized as follows. We first describe how CIPPE can be used to enable policy-hiding access control in Section 4.1. Then we give a formal definition of CIPPE in Section 4.2. In the next two sections, we present two building blocks that we build for CIPPE, one is circuit construction of policy functions with uniform topology, the other is the CIOT protocol. In Section 4.5 we give an efficient construction for CIPPE.

4.1 Using OACerts and CIPPE for Policy-Hiding Access Control

In this section, we present a high-level framework for policy-hiding access control using CIPPE. We describe how the policy-hiding access control in Example 2 can be enabled. In what follows, we use commit to denote the commitment algorithm of a commitment scheme. Let Params denote the public parameters for commit. To be secure, a commitment scheme cannot be deterministic; thus a commitment of a value a also depends on an auxiliary input, a secret random value r. We use $c = \mathsf{commit}_{\mathsf{Params}}(a, r)$ to denote a commitment of a. Figure 4.1 depicts how CIPPE can be used in the policy-hiding access control. We observe that the two CA's are involved only in issuing certificates to Alice. When Alice is interacting with various servers such as Bob, the CA's are not involved and can be off-line. Note that Kantarcioglu and Clifton have proposed a similar privacy protection model in [52].

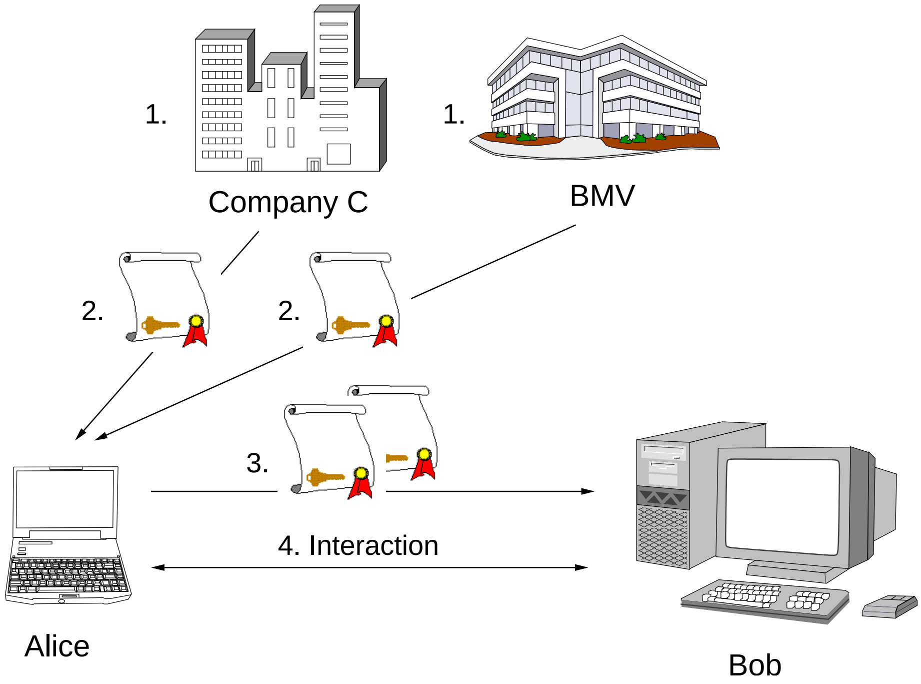

Figure 4.1. An example of policy-hiding access control procedures between Alice and Bob.

1. **CA Setup.** Let Bureau of Motor Vehicles (BMV) be the CA who issues digital driver licenses. BMV runs the CA setup program, *i.e.*, BMV picks a signature scheme, a commitment scheme denoted by commit, a pair of public/private keys, and the public parameters for the commitment scheme, Params. Let Company C be Alice's employer, the CA that issues an income certificate for Alice. Company C runs the CA setup program analogously.

2. **Alice-CA Interaction.** In this phase, Alice obtains two OACerts, one from BMV and the other from Company C. Alice applies for a digital driver license certificate from BMV as follows. BMV first verifies the correctness of her attribute values through some (possibly off-line) channels, then issues an OACert for Alice. The OACert is signed using the BMV's key and contains Alice's public key, BMV's public key, and a commitment for each attribute value that is to be included in the certificate. For example, let x be Alice's birth-date (encoded as an integer), BMV generates a random number r, computes $c = \mathsf{commit}_{\mathsf{Params}}(x, r)$, and stores c in the OACert. The BMV sends the signed OACert to Alice, together with all the secret random values that have been used. Similarly Alice obtains an income certificate from her employer Company C.

3. **Alice-Bob Setup.** Alice applies for a special-rate loan from Bob. Bob reveals that the loan policy takes at most three attributes: birth-date, current salary, and the length of current employment. Alice shows her driver license OACert and income OACert to Bob. Alice then proves the ownership of her OACerts using the usual techniques [28]. Recall that OACerts can be used as regular digital certificates (*e.g.*, X.509 certificates) except the attribute values are stored in the committed form.

4. **Alice-Bob Interaction.** Alice and Bob run an interaction protocol, where Alice inputs her attribute values and secret random values she has stored from Phase 2 (Alice-CA Interaction) and Bob inputs his private policy function. In the end, both Alice and Bob learn whether Alice satisfies Bob's policy without getting other information about Alice's attributes or Bob's policy.

4.2 Definition of Certified Input Private Policy Evaluation

We now give a formal definition of CIPPE, which allows us to prove our protocol for CIPPE is secure.

Definition 4.2.1 (CIPPE) A CIPPE scheme is parameterized by a commitment scheme commit. A CIPPE scheme involves a client C, a server S, and a trusted CA, and has the following four phases:

CA Setup CA takes a security parameter σ and another parameter ℓ (which specifies the desired range of the attribute values), and outputs public parameters Params for commit. The domain of commit contains $[0..2^\ell - 1]$ as a subset. CA sends Params to C and S.

Client-CA Interaction C chooses n values $x_1, \ldots, x_n \in [0..2^\ell - 1]$ (these are C's attribute values) and sends them to CA. For each i, $1 \leq i \leq n$, CA generates a new random number r_i and computes the commitment $c_i = \mathsf{commit}_{\mathsf{Params}}(x_i, r_i)$. CA gives c_i and r_i to C, and c_i to S.

Recall that in the actual usage scenario in Section 4.1, CA does not directly communicate with S. Instead, CA verifies C's attribute values before computing the commitments and storing $c_1, \ldots, c_n$ into C's OACerts. The OACerts are then sent by C to S, enabling S to have the commitment values as if they were sent from CA. Here we abstract these steps away to have CA sending c_i to S. We stress that CA does *not* directly participate in the policy-hiding access control process between C and S.

Client-Server Setup S chooses a family $\mathcal{F}$ of functions and sends the description of $\mathcal{F}$ to C (this models the fact that $\mathcal{F}$ is public knowledge). Each f in $\mathcal{F}$ maps n ℓ-bit integers to a bit, *i.e.*, $f : ([0..2^\ell - 1])^n \rightarrow \{0, 1\}$. S chooses a function $f \in \mathcal{F}$ privately.

Now S has $c_1, \ldots, c_n$, and f. C has $c_1, \ldots, c_n$, $x_1, \ldots, x_n$, and $r_1, \ldots, r_n$.

Client-Server Interaction C and S run an interactive protocol. In the end, both C and S output $F(x_1, r_1, \ldots, x_n, r_n, f)$, where F takes $c_1, \ldots, c_n$, and $\mathcal{F}$ as parameters, and is defined as

$$F(x_1, r_1, \ldots, x_n, r_n, f)$$
$$= \begin{cases} f(x_1, \ldots, x_n) & \text{if } \ f \in \mathcal{F} \ \ \wedge c_i = \mathsf{commit}(a_i, r_i) \text{ for each } i \\ \varnothing & \text{otherwise}. \end{cases}$$

When both C and S are honest, C and S will output $f(x_1, \ldots, x_n)$ in this phase.

To avoid unnecessarily cluttering the exposition, in Definition 4.2.1 we assume that there is only one CA in a CIPPE scheme, and that $x_1, \ldots, x_n$ are equal-length and are committed under the same commitment parameters. The definition of CIPPE can be modified to support multiple CA's, different input lengths, and different commitment parameters. As a matter of fact, we can easily adjust our CIPPE protocol to support the situation in which each x_i is committed under a different set of commitment parameters.

Notion of Security

The security definitions we use follow [26, 53, 54]. We consider security against three kinds of adversaries. An *adversary* is a probabilistic interactive Turing Machine [47]. A *honest-but-curious* adversary is an adversary who follows the prescribed protocol, and attempts to learn more information than allowed from the execution. A *weak-honest* adversary [54] is an adversary who may deviate arbitrarily from the protocol, as long as her behavior appears honest to parties executing the protocol. A *malicious* adversary is an adversary who may behave arbitrarily. When we consider malicious adversaries, there are certain things we cannot prevent: an adversary (1) may refuse to participate in the protocol, (2) may substitute its local input with something else, and (3) may abort the protocol prematurely. When we consider weak-honest, we cannot prevent an adversary from substituting her local input.

The security of a CIPPE protocol is analyzed by comparing what an adversary can do in the protocol to what she can do in the ideal model with a Trusted Third Party (TTP).

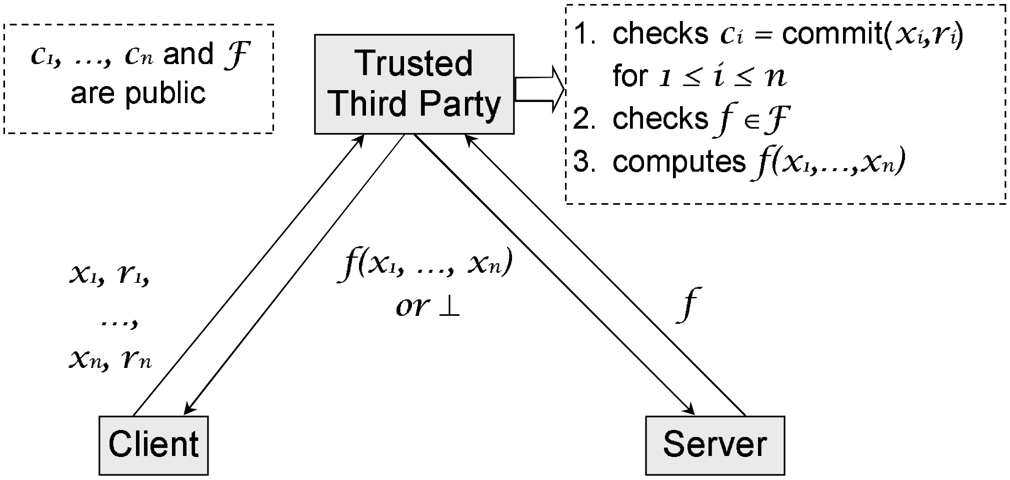
$c_1, \ldots, c_n$ and $\mathcal{F}$ are public
Trusted Third Party
1. checks c_i = commit(x_i,r_i) for $1 \leq i \leq n$
2. checks $f \in \mathcal{F}$
3. computes $f(x_1,\ldots,x_n)$
x_1, r_1, …, x_n, r_n
$f(x_1, \ldots, x_n)$ or $\perp$
f
Client
Server

Figure 4.2. Ideal model for the CIPPE protocol

In the ideal model, as depicted in Figure 4.2, the client sends her private input x_i and r_i, for $1 \leq i \leq n$, to the TTP, and the server sends his private input f to the TTP. The TTP verifies that $c_i = \mathsf{commit}(x_i, r_i)$ for each i and $f \in \mathcal{F}$, computes $f(x_1, \ldots, x_n)$, sends the result back to the client and the server. If the verification fails, the TTP simply outputs a special symbol $\varnothing$.

The ideal model differs for honest-but-curious adversaries, weak-honest adversaries, and malicious adversaries. In the ideal model for honest-but-curious adversaries, an honest party outputs her output from the TTP, whereas an honest-but-curious party outputs an arbitrary function from her initial input and the output she obtained from the TTP. The ideal model for weak-honest adversaries is similar to the ideal model for honest-but-curious adversaries, but differs in that a weak-honest adversary can substitute her input before sending to the TTP. In the ideal model for malicious adversaries, a malicious adversary can terminate the protocol prematurely, even at a stage when she has received her output and the other party has not.

Definition 4.2.2 (Security) Let F be the function the client and server compute in the interaction phase of a CIPPE scheme. Let Π be the CIPPE protocol for computing F. We model the client and server as a pair of admissible probabilistic polynomial-time machines, where at least one of them is honest. Protocol Π *securely computes* F if for every pair of admissible probabilistic polynomial-time machines (C^*, S^*) in real model, there exists a pair of admissible probabilistic polynomial-time machines (C, S) in the ideal model, such that the joint execution of F under (C^*, S^*) in the real model is computationally indistinguishable from the joint execution of F under (C, S) in the ideal model.

Our construction for CIPPE is provably secure in the honest-but-curious model and the weak-honest model. The server's privacy is guaranteed against any malicious client. A malicious server may learn additional information about a client's attributes; however, this additional information is limited to at most one bit and such malicious behavior will be detected by the client (see Section 4.5 for the detailed construction).

4.3 Building Circuits That Have Uniform Topological Structure

When a circuit is scrambled (refer to Chapter 2.5), the operation in each gate is hidden; however, the topological structure of the circuit is not. Therefore, the client could infer some information about the server's function by looking at the scrambled circuit if the server constructs the circuit in a naive way. To protect the server's private function, we present an approach to construct circuits that can compute a family of functions and have the same topological structure.

Function definition

We propose a family $\mathcal{F}$ of functions that can express many policy functions in real applications. We define $\mathcal{F}$ as follows. $\mathcal{F}$ has four parameters ℓ, n, m, and λ. Each function f in $\mathcal{F}(\ell, n, m, \lambda)$ takes m parameters $y_1, \ldots, y_m \in [0..2^\ell - 1]$ and n inputs $x_1, \ldots, x_n \in [0..2^\ell - 1]$, and maps them to $\{0, 1\}$. Let $f(x_1, \ldots, x_n) = p(x_{i_1} \; \mathsf{op}_1 \; y_1, \; x_{i_2} \; \mathsf{op}_2 \; y_2, \; \cdots, \; x_{i_m} \; \mathsf{op}_m \; y_m)$, where $1 \leq i_1, i_2, \ldots, i_m \leq n$, each op_i is one of the following predicates $\{=, \neq, >, <, \geq, \leq\}$, and p is a disjunctive (or conjunctive) normal form in which the number of disjuncts (or conjuncts) is no more than λ.

If the server chooses a function f from the family $\mathcal{F}(\ell, n, m, \lambda)$ of functions, the client should not be able to distinguish f from any other functions in the family. For instance, consider Example 2, Bob (the bank) can set $n = 3, m = 8, \lambda = 4$, and the policy function is of the form:

$$\begin{aligned} f(x_1, x_2, x_3) \quad &= \quad (x_1 \geq 30 \wedge x_2 \geq 43000 \wedge x_3 > 6) \ \vee \\ & \quad\quad (x_1 \geq 25 \wedge x_2 \geq 45000 \wedge x_3 > 12), \end{aligned}$$

where x_1 denotes age, x_2 denotes annual income in dollars, and x_3 denotes length of current employment in months. Alice learns that x_1, x_2, and x_3 are used for comparison at most 8 times, she would not learn information such as which values they are compared with, and how many times each attribute is compared.

If Bob builds a circuit for $f(x_1, x_2, x_3)$ in a naive fashion, Alice can learn from the topology of the circuit how many times each x_i is compared, what these comparison operators are, and some information about the structure of the policy function. One technical difficulty in hiding such information is that each attribute may be used in multiple comparisons, and we want to hide the number of times it is used. A straightforward way to do this is to use m circuits, each of which select one input from the n inputs. This is not efficient as it needs $O(nm)$ gates. Our construction uses results from the literature on permutation and multicast switching networks (see, for example [55–59]). Some of these networks may be useful for constructing circuits for families of functions beyond the ones considered in this chapter.

Basic circuit components

We introduce three basic circuit components that will be used in our construction. We depict them in Figure 4.3.

1. *Comparison circuit.* Given two ℓ-bit integers x and y, the comparison circuit computes $x = y$, $x \neq y$, $x > y$, or $x < y$. Observe that $x \geq y$ and $x \leq y$ can be represented as $x > y - 1$ and $x < y + 1$, respectively. Let $x_{\ell-1} \ldots x_1 x_0$ be the binary representation of x and $y_{\ell-1} \ldots y_1 y_0$ be the binary representation of y.

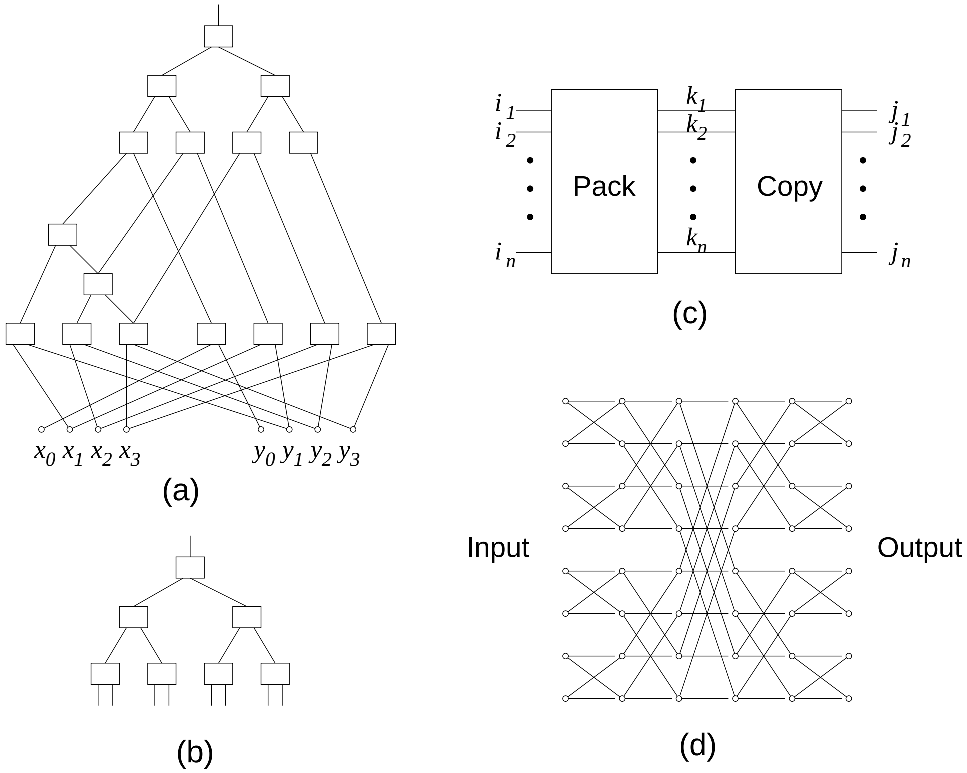

Figure 4.3. Basic circuit components: (a) the structure of 4-bit comparison circuits, (b) the structure of 8-input logical operation circuits, (c) the high-level schema for a generalizer circuit, (d) an (8,8)-generalizer,

- Circuit for $x > y$ is
  $\bigvee_{i=0}^{\ell-1}\left(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{\ell-1}(x_j = y_j)\right)$
- Circuit for $x < y$ is
  $\bigvee_{i=0}^{\ell-1}\left(\neg x_i \wedge y_i \wedge \bigwedge_{j=i+1}^{\ell-1}(x_j = y_j)\right)$
- Circuit for $x = y$ is $\bigwedge_{i=0}^{\ell-1}(x_i = y_i)$
- Circuit for $x \neq y$ is $\bigvee_{i=0}^{\ell-1}(x_i \neq y_i)$

Note that the circuits for $x > y$ and $x < y$ have the same topology. To make the structure of all comparison circuits uniform, we modify the circuits for $x = y$ and $x \neq y$ by adding some "dummy" gates. For example, the comparison circuit for $x = y$ could be $\bigwedge_{i=0}^{\ell-1}\left((x_i = y_i) \wedge \bigwedge_{j=i+1}^{\ell-1} g(x_i, y_i)\right)$ where $g(x_i, y_i)$ always outputs

1. Figure 4.3(a) shows the structure of 4-bit comparison circuits. Note that each ℓ-bit comparison circuit requires $O(\ell)$ gates ($5\ell - 4$ gates).

2. *Logical operation circuit.* Given m Boolean inputs $a_1, \ldots, a_m$, the logical circuit computes $\bigvee_{i \in S} a_i$ or $\bigwedge_{i \in S} a_i$ where $S \subseteq \{1, 2, \ldots, m\}$. We can use a binary tree structure to implement the m-input logical circuit. For example, to compute the logical formula $\bigvee_{i \in S} b_i$, every gate in the binary tree computes $\vee$; if $i \in S$ we give the corresponding wire value a_i, otherwise, set value 0. Figure 4.3(b) shows a 8-bit logical operation circuit. Note that the m-input logical circuits require $O(m)$ gates ($m - 1$ gates).

3. *Generalizer circuit.* An (n, n)-generalizer is an n-input and n-output switching network, it passes each input i to zero or more outputs. The existence of (n, n)-generalizer with $O(n)$ gates is demonstrated nonconstructively by Pipenger [55]. Ofman [60] gives a construction of a generalizer using the schema shown in Figure 4.3(c). In his construction, the network consists of two parts: a pack network and a copy network. The pack network packs those inputs having requests to consecutive positions. The copy network copies inputs to multiple outputs. The network proposed by Ofman [60] requires $3n \log n$ gates. Thompson [56] improved Ofman's work and gives a construction using $2n \log n$ gates. The Thompson's construction uses a reversed butterfly network concatenated with a butterfly network. Figure 4.3(d) is the Thompson's construction of a $(8, 8)$-generalizer.

Our construction

Our construction takes the following three stages.

1. *Copy Stage.* The copy stage takes n ℓ-bit integers $x_1, \ldots, x_n$ and outputs m ℓ-bit integers in which each x_i is copied to output v_i times where $v_i \geq 0$ and $\sum v_i = m$. To build the copy stage in circuit, we construct ℓ identical (n, m)-generalizers, one

for each bit. A (n, m)-generalizer can be implemented by $\lceil \frac{m}{n} \rceil$ numbers of (n, n)-generalizer. This stage needs $O(\ell m \log n)$ gates (about $2\ell m \log n$ gates).

2. *Comparison Stage.* The comparison stage takes m ℓ-bit integers and makes m comparisons. This stage consists of m comparison circuits, one for each (x, y) pair. This stage needs $O(\ell m)$ gates (about $5\ell m$ gates).

3. *Logical Computation Stage.* Observe that all the disjunctive normal forms where the number of conjunctions is no more than λ can be expressed as $\bigvee_{j=1}^{\lambda}(\bigwedge_{i \in S_j} a_i)$, where $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, m\}$. Such disjunctive normal forms can be implemented using λ m-input logical operation circuits and one λ-input logical operation circuits. For each m-input logical operation circuit, the input consists of the m output bits from the comparison stage, the output is connected to the input wire of the last λ-input logical operation circuit. The conjunctive normal forms can be implemented analogously. This stage needs $O(\lambda m)$ gates (about λm gates).

Figure 4.4 shows the structure of circuits that can compute the family $\mathcal{F}(3, 3, 4, 4)$ of functions. For the family $\mathcal{F}(\ell, n, m, \lambda)$ of functions, our circuit construction needs $O(\ell m \log n + \lambda m)$ gates (around $(2 \log n + 5)\ell m + \lambda m$ gates).

4.4 A Committed-Integer Based Oblivious Transfer Protocol

To build a CIPPE protocol using the scrambled circuit protocol, we have to ensure that the client gets the keys of the input wires corresponding to her committed input. We present a Committed-Integer based Oblivious Transfer (CIOT) protocol to achieve this. A CIOT protocol involves a sender and a receiver. The receiver has a committed ℓ-bit integer x, the sender has ℓ pairs of values $(k_1^0, k_1^1), \cdots, (k_\ell^0, k_\ell^1)$, and both the sender and receiver share the commitment of x. In the end of the protocol, the receiver learns exactly one key in each pair; furthermore, the keys she learns correspond to the bits in x. The main idea of CIOT is as follows. Using the commitment of x, the receiver generates ℓ new commitments, one for each bit of x. Then the sender and receiver run a modified version of non-interactive oblivious transfer protocol [61, 62] for each commitment.

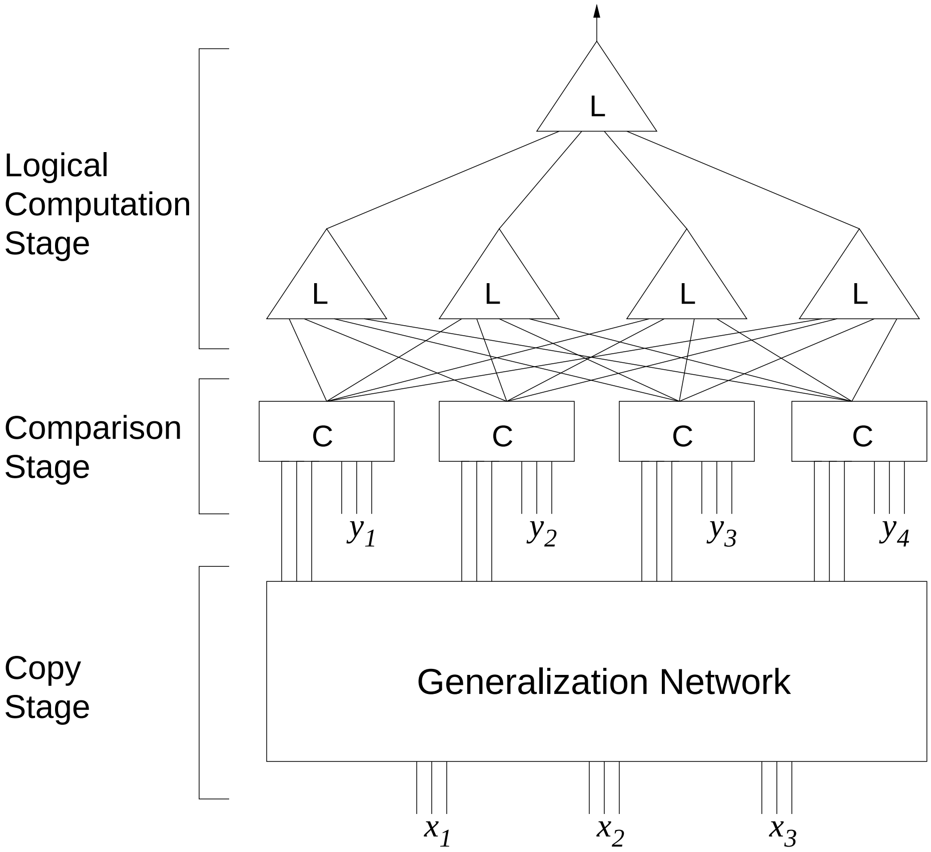

Figure 4.4. An example circuit structure for the family $\mathcal{F}$ of functions with parameters $\ell = 3$, $n = 3$, $m = 4$, and $\lambda = 4$. There are 4 comparison circuits in the comparison stage, and 5 logical operation circuits in the logical computation stage.

Protocol 3 (CIOT Protocol) Let $\langle p, q, g, h \rangle$ be the public parameters of the Pedersen commitment scheme. All arithmetic in this section is $\bmod\ p$ unless specified otherwise. Let x be an integer in $[0..2^\ell - 1]$, and $x_{\ell-1} \dots x_1 x_0$ be the binary representation of x, ***i.e.***, $x = x_0 2^0 + x_1 2^1 + \cdots + x_{\ell-1} 2^{\ell-1}$. Let $c = \mathsf{commit}(x, r) = g^x h^r$ be the commitment of x with a random $r \in \mathbb{Z}_q$.

Input The receiver has x and r, and the sender has ℓ pairs of integers $(k_0^0, k_0^1), \dots,$ $(k_{\ell-1}^0, k_{\ell-1}^1)$. Both the sender and receiver have c.

Output The receiver learns $k_0^{x_0}, \dots, k_{\ell-1}^{x_{\ell-1}}$. The sender learns nothing.

1. The receiver decomposes c into ℓ commitments, one for each bit of x. More specifically, the receiver randomly picks $r_1, \ldots, r_{\ell-1} \in \mathbb{Z}_q$ and sets $r_0 = r - \sum_{i=1}^{\ell-1} 2^i r_i \mod q$. The receiver computes $c_i = \mathsf{commit}(x_i, r_i) = g^{x_i} h^{r_i}$ for $i = 0, 1, \ldots, \ell - 1$, and gives them to the sender. The sender checks that $\prod_{i=0}^{\ell-1} (c_i)^{2^i} = c$. Observe that $\prod_{i=0}^{\ell-1} (c_i)^{2^i} = \prod_{i=0}^{\ell-1} (g^{x_i} h^{r_i})^{2^i} = g^{\sum_{i=0}^{\ell-1} x_i 2^i} h^{\sum_{i=0}^{\ell-1} r_i 2^i} = g^x h^r = c$.

2. For $i = 0, 1, \ldots, \ell - 1$, the sender calculates $K_i^0 = \langle p, q, h, c_i \rangle$ and $K_i^1 = \langle p, q, h, c_i g^{-1} \rangle$. Using the ElGamal encryption scheme [63] (modified to have messages from a subgroup [64]), the sender sends to the receiver two ciphertexts $E_{K_i^0}(k_i^0) = (h^{y_i}, k_i^0 c_i^{y_i})$ and $E_{K_i^1}(k_i^1) = (h^{z_i}, k_i^1 (c_i g^{-1})^{z_i})$, where y_i and z_i are chosen at uniform random from $\mathbb{Z}_q$ by the sender. The receiver can obtain $k_i^{x_i}$ as follow: If x_i equals 0, then $c_i = h^{r_i}$, the receiver knows the private key corresponding to K_i^0 (the private key is r_i), therefore she can decrypt $E_{K_i^0}(k_i^0)$ to recover k_i^0. If x_i equals 1, then $c_i g^{-1} = h^{r_i}$, the receiver knows the private key corresponding to K_i^1, she can decrypt $E_{K_i^1}(k_i^1)$ to recover k_i^1.

Both the sender and receiver need $O(\ell)$ modular exponentiation, *i.e.*, the sender needs 2ℓ modular exponentiation, and receiver needs 4ℓ modular exponentiation. The security properties of the CIOT protocol are given by the following theorems.

Theorem 4.4.1 *The sender does not learn anything from the CIOT protocol.*

Proof The CIOT protocol consists of two phases: a bit-commitment phase and an oblivious transfer phase. The sender learns nothing about x from the oblivious transfer phase, as the receiver does not send any information to the sender during that phase. Thus, all the information the sender learns about x is from the bit-commitment phase. In the bit-commitment phase, the sender learns the commitments $c_0, \ldots, c_{\ell-1}$. Observe that $c_0 = c \prod_{i=1}^{\ell-1} (c_i)^{-2^i}$; therefore, c_0 can be computed from $c, c_1, \ldots, c_{\ell-1}$ and does not leak any additional information. Recall that $r, r_1, \ldots, r_{\ell-1}$ are chosen uniformly randomly from $\mathbb{Z}_q$; the distributions of $c, c_1, \ldots, c_{\ell-1}$ are exactly the same as the distribution of any commitment under the Pedersen commitment scheme. Thus $c, c_1, \ldots, c_{\ell-1}$ leak nothing about their corresponding committed values $x, x_1, \ldots, x_{\ell-1}$. Therefore, the sender does not learn anything about x. In other words, for any $x, y \in [0..2^\ell - 1]$ (let c_x, c_y be the corresponding commitments), and for any adversary executing the sender's part, the views that the adversary sees when the receiver inputs (x, c_x) and when the receiver inputs (y, c_y) are perfectly indistinguishable. ■

Theorem 4.4.2 *Under the DDH assumption and the DL assumption on G_q, the order-q subgroup of $\mathbb{Z}_p^*$, the receiver learns at most one value per (k_i^0, k_i^1) pair.*

Proof Suppose an adversarial receiver learns both k_i^0 and k_i^1 for some given i, where $0 \leq i \leq \ell - 1$. Under the DDH assumption, the ElGamal encryption scheme is semantically secure [64]. Therefore, the adversary knows the private keys corresponding to the ElGamal public keys $K_i^0 = \langle p, q, h, c_i \rangle$ and $K_i^1 = \langle p, q, h, c_i g^{-1} \rangle$. In other words, the adversary knows r where $h^r = c_i$, and r' where $h^{r'} = c_i g^{-1}$. Thus, the adversary knows r and r' where $h^r = gh^{r'}$; she can effectively compute $\log_g(h) = (r - r')^{-1} \bmod q$, which contradicts the DL assumption. ■

Theorem 4.4.3 *Under the DDH assumption and the DL assumption on G_q, the order-q subgroup of $\mathbb{Z}_p^*$, if the receiver learns ℓ keys, these values must be $k_0^{x_0}, \ldots, k_{\ell-1}^{x_{\ell-1}}$.*

Proof By Theorem 4.4.2, if an adversarial receiver learns ℓ keys, she learns exactly one key per (k_i^0, k_i^1) pair. Suppose she learns $k_0^{y_0}, \ldots, k_{\ell-1}^{y_{\ell-1}}$, where $y_i \in \{0, 1\}$ for $i \in [0..\ell - 1]$ and there exist at least one j such that $x_j \neq y_j$. Therefore, $\sum_{i=0}^{\ell-1} y_i 2^i \neq \sum_{i=0}^{\ell-1} x_i 2^i = x$. Under the DDH assumption, the adversary knows the private keys corresponding to the ElGamal public keys $K_i^{y_0}, \ldots, K_i^{y_{\ell-1}}$; thus she knows t_i for each $i \in [0..\ell - 1]$ such that $g^{y_i} h^{t_i} = c_i$. As

$$g^x h^r = c = \prod_{i=0}^{\ell-1} (c_i)^{2^i} = \prod_{i=0}^{\ell-1} (g^{y_i} h^{t_i})^{2^i} = g^{\sum_{i=0}^{\ell-1} y_i 2^i} h^{\sum_{i=0}^{\ell-1} t_i 2^i} = g^y h^t,$$

where y denotes $\sum_{i=0}^{\ell-1} y_i 2^i$ and t denotes $\sum_{i=0}^{\ell-1} t_i 2^i \pmod{q}$, the receiver knows x, r, y, and t such that $g^x h^r = g^y h^t$. The receiver can efficiently compute $\log_g(h)$, which contradicts the DL assumption. ■

In the CIOT protocol, a malicious receiver may learn the inputs that do not correspond to bits of her committed x. However, in this case, she cannot learn a key for every input wire of the circuit. Therefore in this case she cannot compute the output of the scrambled circuit.

4.5 The CIPPE Protocol

We now give the CIPPE protocol which follows Definition 4.2.1, and specify what each participant does in each step.

Protocol 4 (CIPPE Protocol) The CIPPE protocol involves a client C, a server S, and a trusted CA, and has the following four phases:

CA Setup CA takes a security parameter σ and a setup parameters ℓ as input. CA runs the Pedersen commitment setup algorithm to create $\mathsf{Params} = \langle p, q, g, h \rangle$ such that $2^\ell < q$, and sends it to C and S.

Client-CA Interaction C chooses n integers $x_1, \ldots, x_n \in [0..2^\ell - 1]$ and sends them to CA. For each x_i, $1 \leq i \leq n$, CA picks $r_i \in_R \mathbb{Z}_q$ and computes the commitment $c_i = (g^{x_i} h^{r_i} \bmod p)$. CA gives c_i and r_i to C, and c_i to S.

Client-Server Setup S takes three parameters ℓ, n, m, and λ as input, and outputs the family $\mathcal{F}$ of functions as defined in Section 4.3. S sends the description of $\mathcal{F}$ to C, then chooses a private function $f \in \mathcal{F}$.

Now S has $c_1, \ldots, c_n$, and f. C has $c_1, \ldots, c_n$, $x_1, \ldots, x_n$, and $r_1, \ldots, r_n$.

Client-Server Interaction The steps are as follows.

1. *Scrambling the circuit:* S constructs a circuit that computes the function f using the technique described in Section 4.3, then scrambles the circuit. S gives the scrambled circuit to C.

2. *Committing the output:* Let wire w_t denote the unique output wire of the scrambled circuit, and (k_t^0, k_t^1) denote the corresponding keys of w_t. S sends $\langle \eta_0 = E_{k_t^0}\left[\, 0^\sigma \,\right], \eta_1 = E_{k_t^1}\left[\, 1^\sigma \,\right] \rangle$ to C.

3. *Coding the input:* For each x_i where $1 \leq i \leq n$, there are ℓ corresponding input wires in the scrambled circuit. C and S run the CIOT protocol in which C inputs x_i, r_i, and c_i; and S inputs c_i and ℓ pairs of keys that correspond to the ℓ input wires. In the end of this step, C learns one key per input wire; furthermore, each key corresponds to a bit in C's committed input.

4. *Evaluating the circuit:* After Step 3, C possesses enough information to evaluate the scrambled circuit independently. C evaluates the circuit and obtains k, the key of the output wire. Recall that C receives $\langle \eta_0, \eta_1 \rangle$ from S in step 2, C tries to decrypt η_0 and η_1 using key k. If C fails in decrypting both of them, she outputs $\varnothing$ and aborts; this happens only when S intentionally misbehaves. If C succeeds in decrypting η_0 and gets 0^σ, she outputs 0. Otherwise, if C succeeds in decrypting η_1 and gets 1^σ, she outputs 1.

5. *Notifying the result:* C sends k to S, enabling S to output 0 if $k = k_t^0$ and output 1 if $k = k_t^1$.

The purpose of committing the output in step 2 is to achieve the fairness of the computation. The client and server need $O(\ell n)$ modular exponentiation and $O(\ell m \log n + \lambda m)$ symmetric key encryptions. More precisely, the server needs around $2\ell n$ modular exponentiation and the client needs around $4\ell n$ modular exponentiation, both the client and server need $(16 \log n + 40)\ell m + 8\lambda m$ symmetric key encryptions.

The CIPPE protocol is complete in the sense that if both C and S follow the protocol, C will get proper keys of the input wires, and will be able to evaluate the scrambled circuit correctly. We now briefly discuss the security properties of our CIPPE protocol and the intuitions underlying these properties.

- The CIPPE protocol is secure against honest-but-curious adversaries. This property follows from the fact that the scrambled circuit protocol is secure in the honest-but-curious model [23].

  An honest-but-curious server cannot learn any information about a client's private input, as the server is not able to obtain any information about the client's committed values from the CIOT protocol. An honest-but-curious client cannot learn any information about the a server's private function f, because the client learns only the topology of the circuit which is public information.

- The CIPPE protocol is secure against weak-honest adversaries. As we described in Chapter 4.4, for the client to evaluate the scrambled circuit, the client has to get one input value for each of her input wires to the circuit. Furthermore, if the client gets an input value for each of the input wires, these input values must correspond to her committed attribute values. Therefore, the client can neither learn the server's private function f nor change the result of the computation, as long as the server follows the protocol.

  A weak-honest server cannot build a circuit that has different topology than expected, because the client can detect such deviant behavior immediately. The server can build a circuit that computes a function other than f, we denote the function computed by the circuit as f'. Because f is the server's private input, it is essentially same as if the server inputs f' instead of f. In other words, such adversarial behavior in the real model can be simulated by the execution of computing f' in the ideal model.

- The CIPPE protocol is secure against a malicious client. However, a malicious server may learn (at most) one extra bit of information. The server can do this by constructing a scrambled circuit that would fail for some of the client's input. While executing step 4 of Protocol 2, the client may fail during the evaluation of the circuit or fail to decrypt η_0 and η_1. At the end of the protocol, the client may send (k_t^0, k_t^1), or nothing to the server. Thus the server can classify the client's input into one of three subsets determined by the circuit. When the server is honest, the server can classify the client's input into only two subsets. In other words, a malicious server may learn at most one bit of extra information. However, the client would know that the server has cheated when the circuit evaluation fails.

In the example scenario we consider, if the bank is detected to be dishonest by Alice, the bank's reputation may suffer. This small extra gain does not seem to warrant such malicious behavior in the kind of electronic commerce scenarios we consider.

5 A PRIVACY-PRESERVING TRUST NEGOTIATION PROTOCOL

In traditional trust negotiation [7, 8, 11, 12, 65, 66] the notion of sensitive credential protection has been well studied. In these schemes, each sensitive credential has an access control policy – a credential is used (or revealed) only when the other party satisfies the policy for that credential. This does not prevent sensitive credential leakage, but it does allow the user to control the potential leakage of her credentials. The privacy-preserving attribute-based access control schemes in [22, 67, 68] and in Chapter 4 did not reveal credentials but could not handle policies for credentials (i.e., they dealt with the easier special case where each credential's access control policy was unconditionally "true"). This work is the first to combine the techniques for privacy-preserving access control with the notion of policies for sensitive credentials. These credential policies have to be considered sensitive as well, because otherwise the server (or client) can game the system in many ways. For example, if the client knows the access control policies for the server's credentials then she will know the path of least resistance to unlock certain credentials and thus she will be able to probe more easily.

We organize this chapter as follows. We begin with a detailed description of our contributions in Section 5.1. We review trust negotiation and propose a new definition of trust negotiation that supports policy cycles in Section 5.2. Next, we formally introduce our approach to trust negotiation in Section 5.3. We present our protocol for privacy-preserving trust negotiation in Section 5.4. We give efficiency improvements for our base scheme in section 5.5. Finally, we give a sketch of the proof of security in Section 5.6.

5.1 Our Contributions

We introduce a protocol for privacy-preserving trust negotiation, where the client and server each input a set of credentials along with an access control policy for each of their credentials. The protocol determines the set of usable credentials between the client and the server, and then will process the resource or service request based on the client's usable credentials. A credential is *usable* if its access control policy has been satisfied by the other party. Our protocol is complicated by the fact that: (1) the policies for sensitive credentials may themselves be sensitive and therefore cannot be revealed, (2) the client should not learn information about which of her credentials or the server's credentials are usable, and (3) the server should not learn information about which of his credentials or the client's credentials are usable. The rationale for requirement (1) was given in the previous section. Requirements (2) and (3) are because, if the client or server were to learn which of its credentials are usable, then this would reveal more information about the other party's credential set and thus facilitate probing attacks. The technical contributions of this chapter include:

1. We develop a new privacy-preserving trust negotiation protocol and several novel cryptographic protocols for carrying it out. One of the challenges is the distinction between having a credential and being able to use that credential (when its access control policy has been satisfied), while requiring that "not having" a credential be indistinguishable from "having but being unable to use" a credential.

2. We propose a *reverse eager trust negotiation strategy* (denoted as RE strategy) that handles arbitrary policy cycles, whereas the existing traditional trust-negotiation strategies (such as the eager strategy [7]) are inherently unable to handle such cycles (even if these strategies were properly implemented in this framework).

5.2 Trust Negotiation: Review and Discussion

In trust negotiation [7–9, 11, 12, 65, 66], the disclosure of a credential s is controlled by an access control policy p_s that specifies the prerequisite conditions that must be satisfied in order for credential s to be disclosed. Typically, the prerequisite conditions are a set of credentials $C \subseteq \mathcal{C}$, where $\mathcal{C}$ is the set of all credentials. As in [7, 8, 11, 65, 66], the policies in this chapter are modeled using propositional formulas. Each policy p_s takes the form

$s \leftarrow \phi_s(c_1, \ldots, c_k)$ where $c_1, \ldots, c_k \in \mathcal{C}$ and $\phi_s(c_1, \ldots, c_k)$ is a normal formula consisting of literals c_i, the Boolean operators $\vee$ and $\wedge$, and parentheses (if needed). In this chapter, s is referred to as the target of p_s, and $\phi_s(c_1, \ldots, c_k)$ is referred to as the policy function of p_s.

Given a set of credentials $\mathcal{C}' \subseteq \mathcal{C}$ and a policy function $\phi_s(c_1, \ldots, c_k)$, we denote $\phi_s(\mathcal{C}')$ as the value of the normal formula $\phi_s(x_1, \ldots, x_k)$ where $x_i = 1$ if and only if $c_i \in \mathcal{C}'$ (otherwise $x_i = 0$). For example, if $\phi_s = (c_1 \wedge c_2) \vee c_3$, then $\phi_s(\{c_1, c_2, c_4\}) = 1$ and $\phi_s(\{c_1, c_4\}) = 0$. Policy p_s is *satisfied* by a set of credentials $\mathcal{C}' \subseteq \mathcal{C}$ if and only if $\phi_s(\mathcal{C}') = 1$. During trust negotiation, one can disclose credential s if $\phi_s(\mathcal{C}') = 1$ where $\mathcal{C}'$ is the set of credentials that she has received from the other party.

A trust negotiation protocol is normally initiated by a client requesting a resource from a server. The negotiation consists of a sequence of credential exchanges. Trust is established if the initially requested resource is granted and all policies for disclosed credentials are satisfied [7, 65]. In this case, the negotiation between the client and server is a *successful* negotiation, and otherwise, it is a *failed* negotiation. We give the formal definition for traditional trust negotiation as follows:

Definition 5.2.1 (Traditional Trust Negotiation) *Let $\mathcal{C}_S$ and $\mathcal{P}_S$ ($\mathcal{C}_C$ and $\mathcal{P}_C$) be the sets of credentials and policies possessed by a negotiating server (client). The negotiation is initiated by a request for $s \in \mathcal{C}_S$* [1] *from the client. The goal of trust negotiation is to find a* credential disclosure sequence $(c_1, \ldots, c_n = s)$*, where $c_i \in \mathcal{C}_S \cup \mathcal{C}_C$, and such that for each c_i, $1 \leq i \leq n$, the policy for c_i is satisfied by the credentials already disclosed, i.e., $\phi_{c_i}(\bigcup_{j<i} c_j) = 1$. If the client and server find a credential disclosure sequence, the negotiation succeeds, otherwise, it fails.*

The sequence of disclosed credentials depends on the decisions of each party; these decisions are referred to as a strategy. A strategy controls which credentials are disclosed, when to disclose them, and when to terminate a negotiation [66]. Several negotiation strategies are proposed in [7, 65, 66]. For example, in the eager strategy [7], two parties

[1] For simplicity, we model service s as a credential. In order to obtain s, the client has to have credentials that satisfy ϕ_s.

take turns disclosing a credential to the other side as soon as the access control policy for that credential is satisfied. Each negotiator iteratively executes the pseudo-code in Figure 5.1. The negotiation succeeds if s appears in the output (*i.e.*, $s \in \mathcal{M}$), and it fails if the size of the credential disclosure sequence does not increment after one round of execution (*i.e.*, $\mathcal{M} = \emptyset$). Note that any negotiation using the eager strategy takes at most $\min(n_S, n_C)$ rounds, where n_S and n_C are the sizes of $\mathcal{C}_S$ and $\mathcal{C}_C$, respectively. The following is an example of trust negotiation using the eager strategy.

```
$\mathrm{eager\text{-}strategy}(\mathcal{D}, \mathcal{C}, \mathcal{P}, s)$
    $\mathcal{D} = \{c_1, \ldots, c_k\}$: the credential disclosure sequence.
    $\mathcal{C}$: the local credentials of this party.
    $\mathcal{P}$: the local policies of this party.
    $s$: the service to which access was originally requested.
Output:
    $\mathcal{M}$: the set of new released credentials.
Pre-condition:
    $s$ has not been disclosed.
Procedure:
    $\mathcal{M} = \emptyset$;
    For each credential $c \in \mathcal{C}$
        let $c$'s policy be $p_c : c \leftarrow \phi_c$;
        if $\phi_c(\mathcal{D}) = 1$, then $\mathcal{M} = \mathcal{M} \cup \{c\}$;
    $\mathcal{M} = \mathcal{M} - \mathcal{D}$;
    return $\mathcal{M}$.
```

Figure 5.1. Pseudocode for the eager strategy

Example 3 Suppose the client and server have the following policies:

$Client$	$Server$
$p_{c_1} : c_1 \leftarrow s_1$	$p_s : s \leftarrow c_5 \vee (c_2 \wedge c_4)$
$p_{c_2} : c_2 \leftarrow s_2 \wedge s_3$	$p_{s_1} : s_1 \leftarrow c_4$
$p_{c_3} : c_3 \leftarrow s_1 \vee s_2$	$p_{s_2} : s_2 \leftarrow c_1$
$p_{c_4} : c_4 \leftarrow \mathsf{true}$	$p_{s_3} : s_3 \leftarrow \mathsf{true}$

where s denotes the server's service, $\{s, s_1, s_2, s_3\}$ denote the set of server's credentials, $\{c_1, c_2, c_3, c_4\}$ denotes the set of the client's credentials. Using the eager strategy, the client begins by revealing credential c_4, as the policy function for c_4 is true (thus it is trivially satisfied). The server then discloses s_3 (which can be revealed freely) and s_1 (which requires the earlier receipt of c_4). The exchange of credentials continues as the final disclosure sequence is $\{c_4, s_1, s_3, c_1, c_3, s_2, c_2, s\}$. Note that all policies for disclosed credentials have been satisfied.

Although the cryptographic contributions of this chapter will make it possible to implement the eager strategy in the framework considered, we do not pursue this approach because it fails to handle policy cycles. In fact, if there is a policy cycle, the trust negotiation will fail under Definition 1. We now propose a new definition of trust negotiation that supports policy cycles.

Definition 5.2.2 (Cycle-Tolerant Trust Negotiation) *Let $\mathcal{C}_S$ and $\mathcal{P}_S$ ($\mathcal{C}_C$ and $\mathcal{P}_C$) be the sets of credentials and policies possessed by a negotiating server (client). The negotiation is initiated by a request for $s \in \mathcal{C}_S$ from the client. The negotiation between the client and server succeeds if there exists usable credential sets $\mathcal{U}_S \subseteq \mathcal{C}_S$ and $\mathcal{U}_C \subseteq \mathcal{C}_C$ for the server and client respectively, such that (1) $s \in \mathcal{U}_S$, (2) $\forall c \in \mathcal{U}_S$, $\phi_c(\mathcal{U}_C) = 1$, and (3) $\forall c \in \mathcal{U}_C$, $\phi_c(\mathcal{U}_S) = 1$. Otherwise, the negotiation fails.*

Note that the above definition allows for many possible $\mathcal{U}_C, \mathcal{U}_S$ solution pairs, and does not capture any notion of minimality for such pairs: Some solution pair may be a proper subset of some other pair, and either of them is considered acceptable. This is fine in the framework of this chapter, because at the end of the negotiation nothing is revealed about the specific $\mathcal{U}_C$,$\mathcal{U}_S$ pair, i.e., neither party can distinguish which pair was responsible for access or whether that pair was minimal or not. *It also implies that the trust negotiation strategy we design need not make any particular effort at zeroing in on a particular pair (e.g., a minimal one).*

Example 4 Suppose the client and server have the following policies:

Client	*Server*
$p_{c_1} : c_1 \leftarrow s_2$	$p_s : s \leftarrow c_5 \vee (c_2 \wedge c_4)$
$p_{c_2} : c_2 \leftarrow s_2 \wedge s_3$	$p_{s_1} : s_1 \leftarrow c_6$
$p_{c_3} : c_3 \leftarrow s_6$	$p_{s_2} : s_2 \leftarrow c_1$
$p_{c_4} : c_4 \leftarrow \mathsf{true}$	$p_{s_3} : s_3 \leftarrow c_4$

where s denotes the server's service, $\{s, s_1, s_2, s_3\}$ denote the set of server's credentials, $\{c_1, c_2, c_3, c_4\}$ denotes the set of the client's credentials. Under Definition 1, the negotiation between the client and server would fail as there is a policy cycle between c_1 and s_2, and there exists no credential disclosure sequence ending with s. However, under Definition 2, the negotiation succeeds, as $\mathcal{U}_C = \{c_1, c_2, c_4\}$ and $\mathcal{U}_S = \{s, s_2, s_3\}$ is a solution pair.

If the trust negotiation between the client and server can succeed in Definition 1, it will also succeed in Definition 2. Let U be the set of credentials in the final credential disclosure sequence in Definition 1, then $U \cap C_C$ is a usable credential set for the client and $U \cap C_S$ is a usable credential set for the server such that these two credential sets satisfy the definition in Definition 2. However, it is not vice-versa, that is, success of negotiation in Definition 2 does not imply negotiation success in Definition 1 (*e.g.*, see Example 4). In the next section, we describe a reverse eager (RE) strategy that efficiently determines whether the negotiation can succeed (under Definition 2) given $\mathcal{C}_S$, $\mathcal{P}_S$, $\mathcal{C}_C$, and $\mathcal{P}_C$. Then, we will give a privacy-preserving trust negotiation protocol that securely implements the RE strategy *without* revealing $\mathcal{C}_S$ and $\mathcal{P}_S$ to the client and *without* revealing $\mathcal{C}_C$ and $\mathcal{P}_C$ to the server.

5.3 Our Approach

We begin this section with an intuitive, informal presentation of our approach. The eager strategy for trust negotiations can be thought of as one of "progressively incrementing the usable set": The set of usable credentials is initially set to the unconditionally usable credentials, and each iteration adds to it credentials that have just (in that iteration) become known to be usable. It is, in other words, a conservative approach, whose motto is that *a credential is not usable unless proved otherwise*: The iterative process stops when no more credentials are added to the usable set. This conservatism of the eager approach is also why using that strategy would lead us to deadlock on cycles. Our overall strategy is the opposite, and can be viewed as a "reverse eager" strategy: Initially all credentials are temporarily considered to be usable, and each iteration *decreases* the set of usable credentials (of course the decrease is achieved implicitly, so as not to violate privacy – more on these implementation details is given in the next section). Note that, because of the "optimism" of the RE strategy (in that a credential is tentatively usable, until proven otherwise), cycles no longer cause a problem, because a "self-reinforcing" cycle's credentials will remain usable (whereas it deadlocked in the eager strategy). This RE strategy (the details of which are given later) is made possible by the fact that we carry out the iterative process in a doubly blinded form, so that *neither party learns anything* (not only about the other party's credentials, but also about their use policies for these credentials). The RE strategy and blinded evaluations work hand in hand: The former is useless without the latter, and it should not be used outside of this particular framework. Note that because the credentials and policies are blinded, it is acceptable for the RE strategy to find maximal usable credential sets rather than minimum ones.

The rest of this section gives a more precise presentation by first introducing the notation that will be used throughout the rest of the chapter, then defining our problem and giving a more detailed overview of our approach.

5.3.1 Notation and Definitions

Before describing the details of our approach, it is necessary to give a more formal notation than the intuitive terminology of the previous section.

- We use s to denote the server's service or resource that the client requests. Without loss of generality, we model s as a credential.
- We use $\mathcal{C}_C$ (resp., $\mathcal{C}_S$) to denote the set of the client's (resp., the server's) hidden credentials. We use n_C and n_S to denote the size of $\mathcal{C}_C$ and $\mathcal{C}_S$, respectively. Referring to Example 4, $\mathcal{C}_C = \{c_1, c_2, c_3, c_4\}$ and $n_C = 4$.
- We use $\mathcal{P}_C$ (resp., $\mathcal{P}_S$) to denote the set of the client's (resp., server's) policies.
- We use $R(p_i)$ to denote the set of credentials relevant to (i.e., that appear in) the policy function of the policy p_i. For example, if the policy function for p_i takes the form of $\phi_i(c_1, \ldots, c_k)$, then $R(p_i) = \{c_1, \ldots, c_k\}$.
- We use $R(\mathcal{P}_C)$ (resp. $R(\mathcal{P}_S)$) to denote the union of all the $R(p_i)$'s over all p_i in $\mathcal{P}_C$ (resp. $\mathcal{P}_S$), *i.e.*, $R(\mathcal{P}_C) = \bigcup_{p_i \in \mathcal{P}_C} R(p_i)$. We use m_C and m_S to denote the size of $R(\mathcal{P}_C)$ and $R(\mathcal{P}_S)$, respectively. Referring to Example 4, $R(\mathcal{P}_S) = \{c_1, c_2, c_4, c_5, c_6\}$ and $m_S = 5$.
- We use $\mathcal{U}_C$ (resp., $\mathcal{U}_S$) to denote the set of the client's (resp., the server's) credentials whose policies *are presumed to have been satisfied* (i.e., these are the currently-believed usable credentials); as stated earlier, these sets will decrease from one iteration to another. Initially, $\mathcal{U}_C = \mathcal{C}_C$ and $\mathcal{U}_S = \mathcal{C}_S$, and throughout the iterative process we have $\mathcal{U}_C \subseteq \mathcal{C}_C$ and $\mathcal{U}_S \subseteq \mathcal{C}_S$.

5.3.2 Problem Definition

The goal of this chapter is to develop a solution such that the client and server are able to learn whether trust can be established without either party revealing to the other party anything about their own private credentials and policies (other than, unavoidably, what can be deduced from the computed answer). We formalize the *privacy-preserving trust negotiation* problem as follows.

Problem 2 The server inputs $\mathcal{C}_S$ and $\mathcal{P}_S$ and the client inputs $\mathcal{C}_C$, $\mathcal{P}_C$, and a request for the server's service s. In the end, both the client and server learn whether the client's access to s can be granted based on their credentials and policies, without revealing their sensitive credentials and policies to the other party. In other words, they want to know whether the trust negotiation between the client and server succeeds under Definition 2 without leaking other information, except for n_C, n_S, m_C, and m_S.

Having stated the problem, we will now discuss the information revealed by the protocol. The values n_C and n_S reveal the number of credentials that the client and server respectively have and the values m_C and m_S reveal the size of all policies for all credentials for the client and the server. We do not view this as a problem because the parties can pad their list or their policies with dummy credentials. We now list the security properties required of a solution (a more detailed version is given in Section 5.6).

1. *Correctness*: If trust can be successfully negotiated, then both the client and server should output `true` with overwhelming probability if they follow the protocol.

2. *Robustness against malicious adversaries*: If the trust negotiation fails, then both the client and server should output `false` even if one of the participants is malicious (*i.e.*, behaves arbitrarily) with overwhelming probability.

3. *Privacy-preservation*: The client and server should not learn anything about the other party's private input (credentials and policies) or intermediate results (usable credential sets), other than what can be deduced from the yes/no outcome of the negotiation.

5.3.3 Overview of Our Approach

As described earlier, our overall strategy for privacy-preserving trust negotiation is the RE strategy. During each round of the RE strategy, a negotiator blindly (i.e., without actually learning the outcome) checks which of their presumed-usable local credentials are in fact not usable (according to whether the policy for it has ceased to be satisfied based on the the new presumed-usable credential set of the other party). After this, the negotiator blindly decreases their own local presumed-usable credential set accordingly. Recall that we use $\mathcal{U}_C$ ($\mathcal{U}_S$) to denote the set of the client's (server's) credentials that are presumed usable, *i.e.*, at a particular stage of the iterative process, for each credential in $\mathcal{U}_C$ ($\mathcal{U}_S$), the corresponding usability policy is currently satisfied (although it may cease to be so in a future iteration). We present the RE strategy in Figure 5.2.

```
reverse-eager-strategy$(\mathcal{C}, \mathcal{P}, \mathcal{U}_O)$
    $\mathcal{C}$: the local credentials of this party.
    $\mathcal{P}$: the local policies of this party.
    $\mathcal{U}_O$: the credentials used by the other party.
Output:
    $\mathcal{U}$: the local credentials that can be used.
Procedure:
    $\mathcal{U} = \mathcal{C}$;
    For each credential $c \in \mathcal{C}$
        let $c$'s policy be $p_c : c \leftarrow \phi_c$;
        if $\phi_c(\mathcal{U}_O) = 0$, then $\mathcal{U} = \mathcal{U} - \{c\}$;
    return $\mathcal{U}$.
```

Figure 5.2. Pseudocode for the RE strategy

Our approach to privacy-preserving trust negotiation is to implement the RE strategy in a secure way. We give the high-level description of our protocol in Figure 5.3. In it, the server first initializes $\mathcal{U}_S$. Then the client and server run a secure version of the RE strategy protocol to update $\mathcal{U}_C$ and $\mathcal{U}_S$ iteratively for n rounds, where $n = \min(n_C, n_S)$

(recall that the trust negotiation using the eager strategy takes at most n rounds). In the end, if $s \in \mathcal{U}_S$ (i.e., s can be used), the negotiation succeeds, otherwise, it fails.

```
privacy-preserving-trust-negotiation($s, \mathcal{C}_C, \mathcal{P}_C, \mathcal{C}_S, \mathcal{P}_S$)
Output:
    true or false
Procedure:
    Initialize $\mathcal{U}_S$;
    For $i = 1, \ldots, \min(n_C, n_S)$
        $\mathcal{U}_C = $ reverse-eager-strategy($\mathcal{C}_C, \mathcal{P}_C, \mathcal{U}_S$);
        $\mathcal{U}_S = $ reverse-eager-strategy($\mathcal{C}_S, \mathcal{P}_S, \mathcal{U}_C$);
    If $s \in \mathcal{U}_S$, output true, otherwise, output false.
```

Figure 5.3. High-level description of privacy-preserving trust negotiation

Clearly, $\mathcal{U}_C$ and $\mathcal{U}_S$ should not be known to either the client or the server. Thus $\mathcal{U}_C$ and $\mathcal{U}_S$ need to be maintained in such a way that the values of $\mathcal{U}_C$ and $\mathcal{U}_S$: (1) are unknown to the client and server and (2) cannot be modified by a malicious client or server. We maintain $\mathcal{U}_C$ in the following split way: For each $c \in \mathcal{C}_C$, the client generates two random numbers $r_c[0]$ and $r_c[1]$, and the server learns one of them, denoted as r_c. If $c \in \mathcal{U}_C$, then $r_c = r_c[1]$, otherwise $r_c = r_c[0]$. The client does not learn which value the server obtains, and so by splitting $\mathcal{U}_C$ in this way, the client does not learn $\mathcal{U}_C$. Furthermore, the server does not learn anything about $\mathcal{U}_C$, as the values he obtains from the client look random to him. We maintain $\mathcal{U}_S$ in an analogous way. Our protocol will keep this form of splitting as an invariant through all its steps. This does not solve all privacy problems of the negotiation, but it will be one of the guiding principles of our protocol.

5.3.4 Proof of RE Strategy

We now provide a proof of the correctness of the RE strategy for trust negotiations. That is, we prove that at the end of the RE negotiation every unusable credential has been marked as such (the other credentials correctly retain their initial label of “usable”).

So not only does RE not produce a minimal usable credential set pair $\mathcal{C}_C, \mathcal{C}_S$, in fact it will produce a maximal pair in the sense that every credential (whether essential or not) is kept usable unless marked otherwise. As stated earlier, this is justified by the indistinguishability to either party of any two solution pairs.

Throughout this section, we use $\mathcal{C}_{X,i}$, $X \in \{C, S\}$, to denote the usable credential set of the client (if $X = C$) or of the server (if $X = S$) after iteration i of the RE negotiation has completed. We use $\mathcal{C}_{X,0}$ to denote the initial (prior to iteration 1) usable credential set (which equals $\mathcal{C}_X$). We use $\bar{X}$ to denote $\{C, S\} - X$.

Letting $\mathcal{C}(X)$ denote the correct usable credentials for X, our goal is therefore to prove that, after the last iteration i of the RE negotiation, we have $\mathcal{C}_{X,i} = \mathcal{C}(X)$ and $\mathcal{C}_{\bar{X},i} = \mathcal{C}(\bar{X})$. Note that $\mathcal{C}_{X,i} = f_X(\mathcal{C}_{X,i-1}, \mathcal{C}_{\bar{X},i-1})$ for some monotonic function f_X. (Although in fact $\mathcal{C}_{X_i}$ depends only on $\mathcal{C}_{\bar{X},i-1}$ and not on $\mathcal{C}_{X,i-1}$, it does no harm to give a more general proof, as we do below, for the case when it can depend on both.)

The next lemma proves the intuitive fact that an iteration i cannot cause an unusable credential to become usable.

Lemma 1 $\mathcal{C}_{X,i} \subseteq \mathcal{C}_{X,i-1}$, *for* $i = 1, 2, \ldots$.

Proof By induction on i. For the basis of the induction, $i = 1$, the claim trivially holds because, prior to iteration 1, all the credentials of each party are in their initial usable set $\mathcal{C}_{X,0}$. For $i = 2$, the claim also holds because $\mathcal{C}_{\bar{X},1} \subseteq \mathcal{C}_{\bar{X},0}$, $\mathcal{C}_{X,2} = f_X(\mathcal{C}_{X,1}, \mathcal{C}_{\bar{X},1})$, and $\mathcal{C}_{X,1} = f_X(\mathcal{C}_{X,0}, \mathcal{C}_{\bar{X},0})$, thus we have $\mathcal{C}_{X,2} \subseteq \mathcal{C}_{X,1}$. We now turn our attention to the inductive step, $i > 1$. Observe that

1. during iteration i, $\mathcal{C}_{X,i}$ is computed based on $\mathcal{C}_{X,i-1}$ and $\mathcal{C}_{\bar{X},i-1}$, i.e., $\mathcal{C}_{X,i} = f_X(\mathcal{C}_{X,i-1}, \mathcal{C}_{\bar{X},i-1})$;
2. during iteration $i-1$, $\mathcal{C}_{X,i-1}$ is computed based on $\mathcal{C}_{X,i-2}$ and $\mathcal{C}_{\bar{X},i-2}$, i.e., $\mathcal{C}_{X,i-1} = f_X(\mathcal{C}_{X,i-2}, \mathcal{C}_{\bar{X},i-2})$;
3. by the induction hypothesis we have $\mathcal{C}_{X,i-1} \subseteq \mathcal{C}_{X,i-2}$, and $\mathcal{C}_{\bar{X},i-1} \subseteq \mathcal{C}_{\bar{X},i-2}$

The above facts (1), (2), and (3), together with the monotonicity of the function f_X, imply that $\mathcal{C}_{X,i} \subseteq \mathcal{C}_{X,i-1}$. ■

A corollary of the above lemma is that, to prove the correctness of RE, it suffices to show that for every credential c of party X, c is unusable if and only if there is some iteration i after which $c \notin \mathcal{C}_{X,i}$. The next lemma proves the "if" part. Recall that $\mathcal{C}(X)$ denote the correct usable credentials for X.

Lemma 2 *For every i, we have $\mathcal{C}(X) \subseteq \mathcal{C}_{X,i}$.*

Proof By induction on i. The basis, $i = 0$, is trivial because $\mathcal{C}_{X,0} = \mathcal{C}_X$. For the inductive step, $i > 0$, we assume that credential c was removed by iteration i (i.e., that $c \in \mathcal{C}_{X,i-1}$ and $c \notin \mathcal{C}_{X,i}$), and we show that it must then be the case that $c \notin \mathcal{C}(X)$. Observe that

1. $c \notin f_X(\mathcal{C}_{X,i-1}, \mathcal{C}_{\bar{X},i-1})$;
2. by the induction hypothesis, we have $\mathcal{C}(X) \subseteq \mathcal{C}_{X,i-1}$ and $\mathcal{C}(\bar{X}) \subseteq \mathcal{C}_{\bar{X},i-1}$.

The above (1) and (2), together with the monotonicity of f_X, imply that $c \notin f_X(\mathcal{C}(X), \mathcal{C}(\bar{X}))$, i.e., that $c \notin \mathcal{C}(X)$. ■

The above lemma proved that every c removed by the RE negotiation deserves to be removed (the "if" part). To complete the proof, we need to prove the "only if" part: That every unusable credential will eventually be marked as such by the RE negotiation. That is, we need to prove that every $c \notin \mathcal{C}(X)$ will, for some i, be removed by iteration i. This is proved in the next lemma.

Lemma 3 *For every $c \notin \mathcal{C}(X)$, there is an iteration i for which $c \in \mathcal{C}_{X,i-1}$ and $c \notin \mathcal{C}_{X,i}$.*

Proof For every credential c, let the *level* of c be defined as follows:

- If c is unconditionally usable then $level(c) = 1$.
- If the usability policy for c is p_c then $level(c) = 1 + \max\{level(v) : v \in R(p_c)\}$. (Recall that $R(p_c)$ is the set of credentials relevant to policy p_c.)

We claim that a credential $c \notin \mathcal{C}(X)$ is removed after at most $level(c)$ iterations, i.e., that for some $i \leq level(c)$ we have $c \in \mathcal{C}_{X,i-1}$ and $c \notin \mathcal{C}_{X,i}$. This is established by a straightforward induction on $level(c)$, whose details we omit. ■

5.4 Protocol for Privacy-Preserving Trust Negotiation

5.4.1 Building Blocks

We now describe two building blocks, one for blinded policy evaluation, the other for equality test for array elements. These building blocks will later be used in the secure RE strategy protocol.

Blinded policy evaluation

The goal of the blinded policy evaluation is for Bob to evaluate Alice's policy without learning her policy. Alice should learn nothing about Bob's input nor the output of the evaluation. We define the input and output for this blinded policy evaluation in Figure 5.4.

Input: Alice has a private policy function $\phi : \{0,1\}^k \rightarrow \{0,1\}$, two random numbers t_0 and t_1, and k pairs of values $\{r_1[0], r_1[1]\}, \ldots, \{r_k[0], r_k[1]\}$. Bob has k values $r_1, \ldots, r_k$ where $r_i \in \{r_i[0], r_i[1]\}$.

Output: Bob learns $t_{\phi(r_1 \overset{?}{=} r_1[1], \ldots, r_k \overset{?}{=} r_k[1])}$. Alice learns nothing.

Figure 5.4. Input and output of blinded policy evaluation

The protocol for blinded policy evaluation was given in [67, 69]. In most cases, it requires a polynomial amount of communication, and works for a family of policy functions.

Equality test for array elements

In an equality test for array elements, Alice has a private array $\langle x_1, \ldots, x_n \rangle$ and Bob has a private array $\langle y_1, \ldots, y_n \rangle$. They want to learn whether there exists an index i such that $x_i = y_i$. The result of the equality test is known to neither Alice nor Bob. We define the input and output for this protocol in Figure 5.5.

Input: Bob has n values $\langle y_1, y_2, \ldots, y_n \rangle$. Alice has n values $\langle x_1, x_2, \ldots, x_n \rangle$ and has two random numbers t_0 and t_1.

Output: Bob learns t_1 if and only if there $\exists\ i \in [1..n]$ such that $x_i = y_i$, and learns t_0 otherwise. Alice learns nothing.

Figure 5.5. Input and output of equality test for array elements

This equality test can be implemented by a scrambled circuit evaluation protocol [23, 51]. The protocol requires $O(\rho^2 n)$ communication and computation, where ρ is the maximum bit-length of each x_i and y_i or the security parameter (whichever is larger). We give an efficiency improvement that reduces that communication and computation requirement to $O(\rho n)$ (that is of independent interest) in Section 5.5.

5.4.2 Secure RE Strategy Protocol

The goal of the secure RE strategy protocol is to securely implement the RE strategy in Figure 5.2. We denote the participants of this protocol by Alice and Bob, where Alice is either the client or the server and Bob is the opposite role. In this section, we introduce a protocol to compute $\text{secure-reverse-eager-strategy}(\mathcal{C}_A, \mathcal{P}_A, \mathcal{C}_B, \mathcal{U}_B)$ (the items subscripted by A are Alice's values and those subscripted by B are Bob's values), where the output is $\mathcal{U}_A$ in the split-form described earlier. The careful reader may notice a discrepancy between this and the RE strategy defined earlier. Note that in this case $\mathcal{U}_B$ represents an array of Boolean values marking which credentials are usable, whereas in the previous case it represented the actual credentials. A credential c of Alice's is not usable if Bob's usable credentials do not satisfy Alice's usability policy for c.

Protocol 5 (Secure RE Strategy Protocol) The protocol details are given as follows.

Input Bob inputs: (1) a set of credentials, $\mathcal{C}_B$, which we denote by $b_1, \ldots, b_n$ and (2) his share of $\mathcal{U}_B$, which we denote by ordered pairs $(r_1^B[0], r_1^B[1]), \ldots, (r_n^B[0], r_n^B[1])$. Alice inputs: (1) a set of credentials, $\mathcal{C}_A$, which we denote by $a_1, \ldots, a_m$, (2) a set of policies for these credentials, $\mathcal{P}_A$, which we denote by $p_1, \ldots, p_m$, and (3) her share of $\mathcal{U}_B$, which we denote by $r_1^B[d_1^B], \ldots, r_n^B[d_n^B]$ (note d_i^B is 1 if Bob can use b_i and is 0 otherwise).

Output Alice learns her share of the updated $\mathcal{U}_A$ which is denoted by ordered pairs $(r_1^A[0], r_1^A[1]), \ldots, (r_m^A[0], r_m^A[1])$. Bob learns his share of the updated $\mathcal{U}_A$ which is denoted by $r_1^A[d_1^A], \ldots, r_m^A[d_m^A]$, where $d_i^A = p_i(\mathcal{U}_B)$.

Protocol Steps The steps are as follows.

1. *Determine which credentials in Alice's policies Bob has and can use*: Suppose that the credentials in $R(\mathcal{P}_A)$ are $c_1, \ldots, c_k$. Alice randomly generates k ordered pairs: $(t_1[0], t_1[1]), \ldots, (t_k[0], t_k[1])$. For each credential c_i, Alice and Bob engage in the following steps:

   (a) Alice picks a random number x, and sends $m = I(x, c_i)$, the Identity-Based Encryption (IBE) of x based on the hidden credential c_i, to Bob.

   (b) Bob decrypts m using each of his hidden credentials, and obtains $d_1, \ldots,$ d_n, where $d_i = I^{-1}(m, b_i)$.

   (c) Alice creates a vector $\vec{a}_1 = \langle x + r_1^B[d_1^B], \ldots, x + r_n^B[d_n^B] \rangle$ and Bob creates a vector $\vec{a}_2 = \langle d_1 + r_1^B[1], \ldots, d_n + r_n^B[1] \rangle$. Alice and Bob engage in an equality test protocol for array elements where they each input their own array and Alice inputs $t_i[0]$ and $t_i[1]$. At the end of the protocol, Bob obtains $t_i[x_i]$. Note that x_i is 1 if and only if $c_i \in \mathcal{U}_B$ and Bob has c_i (that is Bob can use the credential and he actually has it) and is 0 otherwise.

2. *Compute $\mathcal{U}_A$:* For each credential a_i, Alice and Bob engage in the following steps:

   (a) Alice randomly generates an ordered pair $(r_i^A[0], r_i^A[1])$.

   (b) Alice and Bob securely evaluate p_i using blinded policy evaluation. Alice inputs $p_i, (r_i^A[0], r_i^A[1]), \{(t_1[0], t_1[1]), \ldots, (t_k[0], t_k[1])\}$ and Bob inputs $\{t_1[x_1], \ldots, t_k[x_k]\}$. At the end of the protocol Bob obtains $r_1^A[d_1^A]$.

3. *Alice and Bob produce $\mathcal{U}_A$:* Alice learns $(r_1^A[0], r_1^A[1]), \ldots, (r_m^A[0], r_m^A[1])$ and Bob learns $r_1^A[d_1^A], \ldots, r_m^A[d_m^A]$

Intuition of Correctness/Security: In Step 1 of the protocol, Bob will learn $t_i[1]$ if he has credential c_i and he can use it, and otherwise he learns $t_i[0]$. Note that these values were generated by Alice. The first part of this (i.e., Bob has c_i) is captured by the value x; that is, Bob is able to obtain x if and only if he has c_i. Furthermore, if Bob's credential b_j is c_i, then $d_j = x$ in Step 1b. The second part of this (i.e., Bob can use c_i) is captured by the set $\mathcal{U}_B$; that is, Alice will have $r_i^B[1]$ if Bob can use c_i can she will have $r_i^B[0]$ otherwise. Putting these pieces together implies that "b_j equals c_i and Bob can use b_j" if and only if $x + r_j^B[d_j^B] = d_j + r_j^B[1]$. Thus the equality test for array elements protocol computes the desired value.

In Step 2 of the protocol Alice and Bob learn their shares of $\mathcal{U}_A$, that is Alice will learn a pair $(r_i^A[0], r_i^A[1])$ and Bob will learn $r_i^A[1]$ if and only if Alice can use credential a_i and he will learn $r_i^A[0]$ otherwise. Note that Alice can use credential a_i only if Bob's usable credential (computed in Step 1) satisfies Alice's policy for a_i. However, this is exactly what the blinded policy evaluation in Step 2 does.

Proof of Correctness/Security: A more detailed proof sketch is given in Section 5.6.

Cost analysis Steps 1(a)-1(c) are performed k times. Step 1(c) requires $O(n\rho^2)$ (where ρ is a security parameter) communication. Thus Step 1 requires $O(kn\rho^2)$ communication, but this can be reduced to $O(kn\rho)$ if the protocol in Section 5.5.1 is used for Step 1(c). Assuming that the policies can be computed with circuits that are linear in the number of credentials, Step 2 requires $O(mk\rho)$ communication. Now k is m_A, n is n_B, and m is n_A, and so this protocol requires $O(m_A\rho(n_A + n_B))$ communication (assuming policies can be computed by a circuit of size linear in the number of bits of their inputs).

5.4.3 Privacy-Preserving Trust Negotiation Protocol

We now "put the pieces together" and give the overall protocol for privacy-preserving trust negotiation.

Protocol 6 (Privacy-Preserving Trust Negotiation Protocol) We now describe the protocol as follows.

Input The client has $\mathcal{C}_C$ and $\mathcal{P}_C$. The server has $\mathcal{C}_S$ (call these credentials $s_1, \ldots, s_{n_S}$) and $\mathcal{P}_S$. Furthermore, s_1 is the service that the client requested.

Output If the trust negotiation between the client and server can succeed, then both the client and server output true, otherwise, they output false.

Protocol Steps The steps are as follows.

1. *Initialize $\mathcal{U}_S$.* For each credential $s_i \in \mathcal{C}_S$, the server picks two random numbers $\{r_i^S[0], r_i^S[1]\}$. The server sends $r_i^S[1]$ to the client. The client calls this value $r_i^S[x_i]$
2. For $i = 1, \ldots, \min(n_C, n_S)$:
   (a) The client and server run the secure RE strategy protocol to obtain $\mathcal{U}_C = \text{secure-reverse-eager-strategy}(\mathcal{C}_C, \mathcal{P}_C, \mathcal{C}_S, \mathcal{U}_S)$ in split form.
   (b) The server and client run the secure RE protocol to obtain $\mathcal{U}_S = \text{secure-reverse-eager-strategy}(\mathcal{C}_S, \mathcal{P}_S, \mathcal{C}_C, \mathcal{U}_C)$ in split form.
3. *Output result.* To determine whether $s_1 \in \mathcal{U}_S$, the server sends a hash of $r_1^S[1]$ to the client. The client checks if the hash of $r_1^S[x_1]$ matches this value; if it is a match then the client proves this to the server by sending $r_1^S[x_1]$ to the server

(and both parties output true), and if it is not a match the client terminates the protocol (and both parties output false).

Intuition of Correctness/Security: In Step 1 of the protocol, the server sets its set of usable credentials to all of its credentials (recall that the RE strategy protocol assumes everything is usable initially and that things are removed from this set).

In Step 2 of the protocol, the client and the server take turns updating their usable credential sets based on the other party's usable set. Once a set ceases to change then the usable sets will cease changing and we will have computed the maximal usable credential set. Note that since we are assuming monotonic policies this will take at most $\min\{n_C, n_S\}$ rounds to compute this set.

Finally, as we model the service as a credential s_1, the client will have $r_1^S[1]$ after Step 3 if and only if s_1 is in the $\mathcal{U}_S$.

Proof of Correctness/Security: A more detailed proof sketch is given in Section 5.6.

Cost analysis Step 2 of the protocol is executed $\min\{n_C, n_S\}$ (call this value n) times. An individual execution requires $O(\rho(m_C + m_S)(n_C + n_S))$ communication and thus the protocol requires $O(n\rho(m_C + m_S)(n_C + n_S))$ communication.

5.5 Efficiency Improvements

5.5.1 A More Efficient Equality Test for Array Elements

In this section, we introduce a more efficient protocol for the equality test for array elements. This protocol is related to the protocol proposed by [70] for secure set intersection. Note that this protocol requires only $O(n\rho + \rho^2)$ communication (instead of $O(n\rho^2)$ communication). We give the proof sketch of correctness and security in Section 5.6.

Protocol 7 (Secure Equality Test Protocol for Array Elements) The input and output of this protocol can be found in Figure 5.5. The protocol steps are as follows.

1. Alice and Bob both choose semantically secure homomorphic encryption schemes E_A and E_B that share a modulus M and exchange public parameters.

2. Alice creates a polynomial P that encodes the x values where the constant coefficient is 1 (which can be done since this arithmetic is modular). In other words she finds a polynomial $P(x) = \eta_n x^n + \eta_{n-1} x^{n-1} + \cdots + \eta_1 x + 1$ where $P(x_i) = 0$ for all x_i. She sends to Bob $E_A(\eta_n), \ldots, E_A(\eta_1)$.

3. Bob chooses a value k_B uniformly from $\mathcal{Z}_M^\star$. For each y_i, Bob chooses a value $q_{B,i}$ uniformly from $\mathcal{Z}_M^\star$ and he computes $(E_A(P(y_i)))^{q_{B,i}} E_A(k_B + y_i) = E_A(q_{B,i} P(y_i) + k_B + y_i)$ (call this value $E_A(\alpha_i)$). Bob sends to Alice $E_A(\alpha_1), \ldots, E_A(\alpha_n), E_B(k_B)$.

4. Alice decrypts the values to obtain $\alpha_1, \ldots, \alpha_n$. She then computes $x_1 - \alpha_i, \ldots, x_n - \alpha_n$ She checks for duplicate values, and if there are duplicates she replaces all extra occurrences of a value by a random value. Alice chooses a value k_A uniformly from $\mathcal{Z}_M^\star$. For each of the values $x_i - \alpha_i$ she chooses $q_{A,i}$ uniformly from $\mathcal{Z}_M^\star$ and then she computes $(E_B(k_B) E_B(x_i - \alpha_i))^{q_{A_i}} E_B(k_A)$= $E_B((x_i + k_B - \alpha_i) q_{A,i} + k_A)$ (we will call this value $E_B(\beta_i)$). Alice sends to Bob $E_B(\beta_1), \ldots, E_B(\beta_n)$.

5. Bob decrypts the values to obtain $\beta_1, \ldots, \beta_n$. Bob then creates a polynomial Q that encodes these values where the constant coefficient is 1. In other words Bob finds a polynomial $Q(x) = \gamma_n x^n + \gamma_{n-1} x^{n-1} + \cdots + \gamma_1 x + 1$ where $Q(\beta_i) = 0$ for all β_i. Bob sends to Alice $E_B(\gamma_n), \ldots, E_B(\gamma_1)$.

6. Alice chooses two values k and q_A uniformly from $\mathcal{Z}_M^\star$ and computes $E_B(Q(k_A) q_A + k)$ and sends this value to Bob.

7. Bob decrypts this value to obtain k'. Alice and Bob engage in a scrambled circuit evaluation of an equality circuit where Alice is the generator with input k and she sets the encodings for the output wire to t_0 for the negative encoding and to t_1 for the positive encoding and Bob is the evaluator with input k'.

5.5.2 Reducing the Number of Rounds

A possible criticism of our protocol for trust negotiation is that it requires $O(\min\{n_C, n_S\})$ rounds. The RE strategy requires this many rounds in the worst case, but in practice it requires much less (it requires rounds proportional to the length of the longest policy chain). Our protocol can be modified to stop as soon as the usable credential sets cease changing. However, this is not recommended as it would leak additional information, and this information allows for additional probing. For example, if the negotiation requires 5 rounds then both parties can deduce that the other party does not satisfy at least 4 of their credentials. Thus, from a privacy standpoint terminating after the usable credential sets cease changing is not a good idea. Another option is to limit the number of rounds to some reasonable constant. This does not have privacy problems, but it could cause the negotiation to succeed when credentials do not satisfy policies. However, if there is domain-specific knowledge that bounds the longest credential chain, then this is a viable option.

5.6 Security Proofs

We now discuss the security of our protocols. We first define what is meant by security. We then briefly sketch components of the proof of security.

5.6.1 Definition of Security

The security definition we use is similar to the standard model from the secure multi-party computation literature [26, 53]. The security of our protocol is analyzed by comparing what an adversary can do in our protocol against what an adversary can do in an ideal implementation with a trusted oracle. Specifically, we will show our protocol is no worse than this ideal model by showing that for any adversary in our model there is an adversary in the ideal model that is essentially equivalent. Thus if the ideal model is acceptable (in terms of security), then our protocols must also be acceptable.

Defining the ideal model for private trust negotiation is tricky. First, the ideal model has to be defined such that there are no “violations of security” that are achievable in this ideal model; otherwise, there could be “violations of security” in our protocols. Furthermore, the ideal model must be defined in such a way as to allow useful trust negotiation to take place; otherwise it and our protocols will not be useful. This is further complicated by the fact that the RE strategy does not make sense in a non-private setting (as one cannot revoke knowledge from another party). Thus we define a fictitious environment where the parties have ”chronic amensia” about the other party’s credentials. In such an environment the RE strategy is plausible, and so our ideal model simulates this environment.

We now informally define an ideal model implementation of our scheme. In the ideal model the client sends $\mathcal{C}_C$ and $\mathcal{P}_C$ to the trusted oracle, and the server sends $\mathcal{C}_S$, $\mathcal{P}_S$, and s to the oracle. We model $\mathcal{P}_C$ and $\mathcal{P}_S$ as arbitrary PPT algorithms. These algorithms will simulate the parties’ behavior during the RE strategy. Thus these algorithms should be viewed as control algorithms that: (1) define which credentials to use during each round, (2) define the access control policies (which we model as PPT algorithms over the other party’s currently usable credentials) for its credentials during each round, and (3) can force the oracle to terminate. We stress that these algorithms cannot do the above operations based upon the state of the negotiation. For example, they cannot force the oracle to terminate when a specific credential becomes unusable. The oracle will simulate the RE strategy using the access control policies defined by each party’s control algorithm. At the end of the negotiation the oracle will inform the client and the server whether access is granted.

5.6.2 Sketch of the Security Proof

We will now sketch part of the proof and we focus only on one specific aspect of the system. We focus on the secure reverse eager strategy protocol (which is the key component of our system). We first show that if Alice is honest, then Bob cannot influence the outcome of the protocols so that he unrightfully keeps one of Alice’s credentials usable.

Lemma 4 *In the secure RE strategy protocol: If Alice is honest and after the protocol a specific credential a_i (with policy p_i) is in $\mathcal{U}_A$, then Bob has a credential set $\mathcal{C}_B$ such that $p_i(\mathcal{C}_B)$ is true.*

Proof (sketch) Because step 2 is done by SCE and Alice is an honest generator, by Lemma 5 all that we must show is that after step 1, Bob learns $t_i[1]$ only when he has credential a_i. By way of contradiction, suppose Bob does not have credential a_i, and that he learns $t_i[1]$ in Step 1c. By Lemmas 6 and 7, Bob only learns $t_i[1]$ when there is a match in the arrays created by Alice and Bob in Step 1c. If there is a match, then Bob must be able to learn x with a non-negligible probability. In other words, Bob can learn x from $I(x, c_i)$ where c_i is a credential Bob does not have. This implies that he can invert the IBE encryption with non-negligible probability, but this contradicts that the IBE encryption scheme is secure. ■

Lemma 5 *In scrambled circuit evaluation: If the generator is honest and the evaluator learns at most one encoding for each input wire, then the evaluator learns at most one encoding for the output wire; furthermore this encoding is the correct value.*

Proof We omit the details of this lemma, but similar lemmas are assumed in the literature. ■

Lemma 6 *In the circuit-version of the equality test for array elements: If Alice is honest, Bob learns t_1 only when there is an index i such that $x_i = y_i$.*

Proof Since Alice is the generator of the circuit and is honest, Bob will input a set of y values and will learn t_1 only when one of his y values matches one of Alice's x values (by Lemma 5). ■

Lemma 7 *In the new version of the equality test for array elements (Section 5.5.1): If Alice is honest, Bob learns t_1 only when there is an index i such that $x_i = y_i$.*

Proof By way of contradiction, suppose Bob learns t_1 and there is no match in their arrays. In Step 7 of the protocol Bob must know the value k (by Lemma 5). Thus in Step

5 of the protocol, Bob must be able to generate a non-zero polynomial of degree n that has k_A as a root, but this implies he knows k_A with non-negligible probability. This implies that in Step 3, Bob can generate values $\alpha_1, \ldots, \alpha_n$ such that there is an α value that is $x_i + k_B$. This implies Bob knows x_i with non-negligible probability, and this implies that there is a match in the arrays. ■

The above only shows one part of the proof. We must also show that if Alice is honest, Bob cannot learn whether he made a specific credential usable (he can force a credential to be unusable, but this has limited impact). Furthermore, we must show that if Bob is honest that Alice does not learn which of her credentials are usable (other than what can be deduced from her policies; i.e., a globally usable credential will definitely be usable). We now show that the protocol is correct, that is if the parties are honest, then the correct usable set is computed.

Proof In step 1 of the protocol, Bob learns a value $t_i[x_i]$ where x_i is 1 if Bob has credential c_i and can use it. There are 3 cases to consider:

1. *Bob does not have* c_i: In Step 1b of the protocol, Bob will not learn the value x, and thus there will not be a match in Step 1c (with very high probability). Since there is no match in the array, Bob will learn $t_i[0]$, which is correct.

2. *Bob has* c_i *but cannot use it.* Suppose $b_j = c_i$ and Alice has $r_j^B[0]$. In this case, $d_j = x$, but Bob's vector entry will be $x + r_j^B[1]$ and Alice's will be $x + r_j^B[0]$. Since there is no match in the array, Bob will learn $t_i[0]$, which is correct.

3. *Bob has* c_i *and can use it.* Suppose $b_j = c_i$ and Alice has $r_j^B[1]$. In this case, $d_j = x$, but Bob's vector entry will be $x + r_j^B[1]$ and Alice's will be $x + r_j^B[1]$. Since there is a match in the array, Bob will learn $t_i[1]$, which is correct.

In step 2 of the protocol, Alice and Bob securely evaluate p_i based upon which credentials are in $\mathcal{U}_B$. If $p_i(\mathcal{U}_B)$ is true, then Bob will learn $r_i^A[1]$ (signifying that Alice can use a_i) and otherwise he will learn $r_i^A[0]$ (signifying that Alice cannot use a_i). ■

6 A TRUST NEGOTIATION FRAMEWORK FOR CRYPTOGRAPHIC CREDENTIALS

A number of cryptographic credential schemes and associated protocols have been developed to address the privacy problems in ATN. Oblivious signature based envelope [45], hidden credentials [22, 68], and secret handshakes [71] can be used to address the policy cycle problem. OACerts (see Chapter 3), private credentials [30], and anonymous credentials [29, 31–33] 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. CIPPE (see Chapter 4) 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 negotiation 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 chapter, 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 [28,42]) 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 [5, 6, 72].

- 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 [9, 12].

The rest of this chapter is organized as follows. We first review several credential schemes and associated protocols that can be used in ATN in Section 6.1. In Section 6.2, we present the language ATNL. In Section 6.3 we present our negotiation protocol.

6.1 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 [30], anonymous credentials [29, 31–33] and OACerts in Chapter 3, 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 [35, 37, 39, 40] 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) [45], hidden credentials [22], or secret handshakes [71,73]. 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. Hidden credentials and secret handshakes share a 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 OCBE in Chapter 3. 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:* In CIPPE in Chapter 4, 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 private credentials [30] and anonymous credentials [29, 31–33], such as the multi-show unlinkable property, anonymous property, etc. Some of these properties require anonymous communication channels [74, 75] to be useful. In this chapter, 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.

6.2 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 [5, 6, 72]. We first give an example trust negotiation scenario in ATNL, then describe the syntax of ATNL in detail.

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:

$$\begin{array}{lll}
\ell 1: & \mathrm{SBA.businessLicense} & \longleftarrow \mathrm{BookSt} \\
\ell 2: & \mathrm{BBB.goodSecProcess} & \longleftarrow \mathrm{BookSt}
\end{array}$$

$BookSt$'s policies:

$$\begin{array}{llll}
m1: & \mathrm{BookSt.discount(phoneNum} = x_3) & \longleftarrow & \mathrm{StateU.student(program} = x_1) \\
 & & & \cap\ \mathrm{BookSt.DoB(val} = x_2) \\
 & & & \cap\ \mathsf{Any}.\mathrm{phoneNum(val} \Rightarrow x_3)\ ; \\
 & & & ((x_1 = \text{‘cs’}) \wedge (x_2 > \text{‘01/01/1984’})) \\
m2: & \mathrm{BookSt.DoB(val} = x) & \longleftarrow & \mathrm{BMV.driverLicense(DoB} = x) \\
m3: & \mathrm{BookSt.DoB(val} = x) & \longleftarrow & \mathrm{Gov.passport(DoB} = x) \\
m4: & \mathsf{disclose}(\mathsf{ac}, \mathrm{SBA.businessLicense}) & \longleftarrow & \mathsf{true} \\
m5: & \mathsf{disclose}(\mathsf{ac}, \mathrm{BBB.goodSecProcess}) & \longleftarrow & \mathsf{true}
\end{array}$$

Figure 6.1. The credentials and policies of BookSt

BookSt's credentials and policies are given in Figure 6.1. 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 (as of January 1, 2005), and willing to provide a phone number. Since the discount is a resource, the head of this policy, $\mathrm{BookSt.discount(phoneNum} = x_3)$, 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 $\mathrm{BookSt.discount(phoneNum} = x_3)$, 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: $\mathrm{StateU.student(program} = x_1) \ \cap\ \mathrm{BookSt.DoB(val} = x_2)$
$\cap\ \mathsf{Any}.\mathrm{phoneNum(val} \Rightarrow x_3)$

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 $\mathrm{StateU.student(program} = x_1)$, one must provide a credential (or a credential chain) showing that one is certified by StateU to be a student; $\mathrm{program} = x_1$ means that the value of the $\mathrm{program}$ field is required to satisfy additional constraints. In $\mathsf{Any}.\mathrm{phoneNum(val} \Rightarrow x_3)$, 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.

Alice's credentials and policies are given in Figure 6.2. 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

Alice's credentials:

$$\begin{array}{lll}
n1: & \mathrm{StateU.student} & \longleftarrow \mathrm{CoS.student} \\
n2: & \mathrm{CoS.student}(\mathrm{program} = \text{‘cs’}, \mathrm{level} = \text{‘sophomore’}) & \longleftarrow \mathrm{Alice} \\
n3: & \mathrm{BMV.driverLicense}(\mathrm{name} = \mathsf{commit}(\text{‘Alice’}), & \\
 & \qquad \mathrm{DoB} = \mathsf{commit}(\text{‘03/07/1986’})) & \longleftarrow \mathrm{Alice}
\end{array}$$

Alice's attribute declarations:

$$\begin{array}{llllllll}
o1: & \mathrm{phoneNum} & = & \text{‘(123)456-7890’} & :: & & :: & \textsf{sensitive} \\
o2: & \mathrm{DoB} & = & \text{‘03/07/1986’} & :: & \mathrm{BMV.License}(\mathrm{DoB}) & :: & \textsf{sensitive} \\
o3: & \mathrm{program} & = & \text{‘cs’} & :: & \mathrm{CoS.student}(\mathrm{program}) & :: & \textsf{non-sensitive} \\
o4: & \mathrm{level} & = & \text{‘sophomore’} & :: & \mathrm{CoS.student}(\mathrm{level}) & :: & \textsf{non-sensitive}
\end{array}$$

Alice's policies:

$$\begin{array}{llll}
p1: & \textsf{disclose}(\textsf{ac}, \mathrm{CoS.student}) & \longleftarrow & \mathrm{SBA.businessLicense} \\
p2: & \textsf{disclose}(\textsf{full}, \mathrm{DoB}) & \longleftarrow & \mathrm{BBB.goodSecProcess} \\
p3: & \textsf{disclose}(\textsf{full}, \mathrm{phoneNum}) & \longleftarrow & \mathrm{BBB.goodSecProcess} \\
p4: & \textsf{disclose}(\textsf{range}, \mathrm{DoB}, \mathrm{year}) & \longleftarrow & \mathsf{true} \\
p5: & \textsf{disclose}(\textsf{ac}, \mathrm{BMV.driverLicense}) & \longleftarrow & \mathsf{true}
\end{array}$$

Figure 6.2. The credentials and policies of Alice

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 birth-date 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 ($\ell1$, $\ell2$), 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 6.1.

The Syntax

Figure 6.3 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 6.3.

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.

Credentials and Roles

Two central concepts that ATNL takes from RT [5, 6] 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

⟨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⟩ ::= ⟨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 6.3. 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.

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.

For example, BookSt's credential ($\ell 1$) in Figure 6.1 is a membership credential. It means SBA issued a business license certificate for BookSt. Alice's credential ($n1$) in Figure 6.2 is a delegation credential. It says that StateU delegates its authority over identifying students to CoS. Alice's membership credential ($n2$) in Figure 6.2 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 $\mathrm{BMV.driverLicense}(\mathrm{name} = \mathsf{commit}(\text{‘Alice’}), \mathrm{DoB} = \mathsf{commit}(\text{‘03/07/1986’}))$ 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 [14, 36][1].

If a credential is a regular certificate, such as an X.509 certificate [28], 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 creden-

[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.

tial, 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.

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 6.2 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

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

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 $\mathrm{Gov.passport(BirthDate)}$.

[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 birth-date, whereas Gov uses $\mathrm{BirthDate}$ as the field name. Name agreement for different field names can be achieved using application domain specification documents [5, 6].

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 6.2, 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.

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, 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 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*.

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 6.1). 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 6.2, BookSt's policy ($m2$) in Figure 6.1 is a policy statement with no constraint. It states that BookSt considers a driver license from BMV to provide adequate documentation of birth-date. 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 $\mathrm{BookSt.student(program} = x_1)$, $\mathrm{BookSt.DoB(val} = x_2)$, and $\mathsf{Any}.\mathrm{phoneNum(val} \Rightarrow x_3)$. 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 $\mathsf{Any}.\mathrm{phoneNum(val} \Rightarrow x_3)$ 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.

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.

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. In practice, a negotiator may limit the number of times that the private policy evaluation is computed on a particular attr to prevent such attack.

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 $\mathsf{disclose}(\mathsf{range}, \mathrm{score}, 50)$, 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.

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.

6.3 The Extended Trust Target Graph 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 [9,12], 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.

6.3.1 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 $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle$, 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.

[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 *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 $\langle V : \mathrm{attr} \overset{?}{\twoheadleftarrow} S\rangle$, 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.

6.3.2 Edges in a Trust-Target Graph

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

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

- A *policy control edge* takes the form $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : B_i.R_i \overset{?}{\twoheadleftarrow} S\rangle$, where i is in $1..k$, and k is greater than 1.

- An *attribute edge* takes the form $\langle V : A.R \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle V : \text{attr} \overset{?}{\twoheadleftarrow} S\rangle$, 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 $\langle V : e \overset{?}{\twoheadleftarrow} S\rangle \hookleftarrow \langle opp(V) : \text{policy-id} \overset{?}{\twoheadleftarrow} V\rangle$, 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.

6.3.3 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 field in a role node or an intersection node, there is a *field state*. 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 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.

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

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

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 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.

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`.

Field state

The field state for each field in a (role or intersection) node has each entry initially unknown. The values of a given node's field states are copied from its children or its grandchildren, as they become available. Field-state entry values are copied from the corresponding field states in delegation-credential children or intersection children. If the given node 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. Otherwise, if the current node has an attribute child, depending on the attribute state of the attribute goal, the opponent reveals the attribute value accordingly. For example, if the full entry in the attribute child is `true`, then the opponent reveals the full value of the field and the value is added to a set of values 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 contain a reference to the current role target, as well as a reference to the corresponding attribute in that role target. This aids in associating constraints and the sources of the values of variables contained therein. 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 false value 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.

6.3.4 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 6.1, 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 of fields in trust targets, and attribute states of attribute goals are not transmitted in messages; instead, each negotiation party infers them independently.

6.3.5 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 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.

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, $\langle K_V : K.r \overset{?}{\twoheadleftarrow} K_S \rangle$, if $K.r$ is a dummy role (defined in a policy statement), the target is opponent-processed, which means that the opponent cannot further 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 a field state for each field. Initially, each field state has three entries, one for the full entry, one for the bit entry, and one for the range entry. The values of these entries are set to be empty.

- 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 add a field state for each field. Initially, each field state has a full entry, a bit entry, and a range entry. The values of these entries are set to be empty.

- 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.

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.

1. Processing $T = \langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle$

(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 $T \longleftarrow\!\!\prec$ $\langle V : \mathrm{policy\text{-}id} \overset{?}{\twoheadleftarrow} S \rangle$.

(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 $T = \langle V : \text{policy-id} \overset{?}{\twoheadleftarrow} S \rangle$

(a) Let $[\text{pre-cond-1 !}]\ B_1.R_1 \cap \cdots \cap B_k.R_k\ ;\ [[\text{pre-cond-2 !}]\ \psi(x_1, \ldots, x_n)]$ be the policy body corresponding to policy-id. If pre-cond-1 is a role, say $A_1.R_1$, V can add a policy control edge $T \leftarrowtail \langle V : A_1.R_1 \overset{?}{\twoheadleftarrow} S \rangle$.

(b) After (a) is done and $\langle V : A_1.R_1 \overset{?}{\twoheadleftarrow} S \rangle$ is satisfied, V can add a policy expansion edge $T \leftarrowtail \langle V : B_1.R_1 \cap \cdots \cap B_k.R_k \overset{?}{\twoheadleftarrow} S \rangle$. V can also do so in the case that there is no precondition for the intersection.

(c) Suppose there is a constraint for this policy. If pre-cond-2 is a role, say $A_2.R_2$, V can add a policy control edge $T \leftarrowtail \langle V : A_2.R_2 \overset{?}{\twoheadleftarrow} S \rangle$.

(d) After (c) is done and $\langle V : A_2.R_2 \overset{?}{\twoheadleftarrow} S \rangle$ 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 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), if the constraint is private, V runs a private policy evaluation protocol with S to evaluate the constraint using the location information stored in the bit entries of the field states. 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.

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 $T = \langle V : A.R \overset{?}{\twoheadleftarrow} S \rangle$

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

2. Processing $T = \langle V : \text{attr} \overset{?}{\twoheadleftarrow} S \rangle$

(a) If there exists a policy statement full-id with head $\textsf{disclose}(\textsf{full}, \text{attr})$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{full-id} \overset{?}{\twoheadleftarrow} V \rangle$. 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 $\textsf{disclose}(\textsf{bit}, \text{attr})$, S can add an attribute control edge $T \leftarrowtail \langle S : \text{bit-id} \overset{?}{\twoheadleftarrow} V \rangle$. 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 P to be the parent node of T. Once

the bit entry of the attribute state becomes $\mathtt{true}$, S writes the identity of P to the bit entry of the field state in P.

(c) If there exists a policy statement $\mathrm{range\text{-}id}$ with head $\mathsf{disclose}(\mathsf{range}, \mathrm{attr}, \mathrm{precision})$, S can add an attribute control edge $T \longleftarrow\!\!\prec \langle S\!:\!\mathrm{range\text{-}id} \overset{?}{\twoheadleftarrow} V\rangle$. S adds a range entry with the precision parameter to the attribute state and sets its value to be $\mathtt{unknown}$. If the attribute control child has been satisfied, S sets the range entry of the attribute state to be $\mathtt{true}$. 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.

6.3.6 Example of The ETTG Protocol

We now give an example that illustrates the ATNL language and the ETTG protocol. This is a simple instance of the ETTG protocol and illustrates the usage of the first three properties described in Section 6.1. Referring to the bookstore example in Section 6.2, we depict the final TTG in Figure 6.4. 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

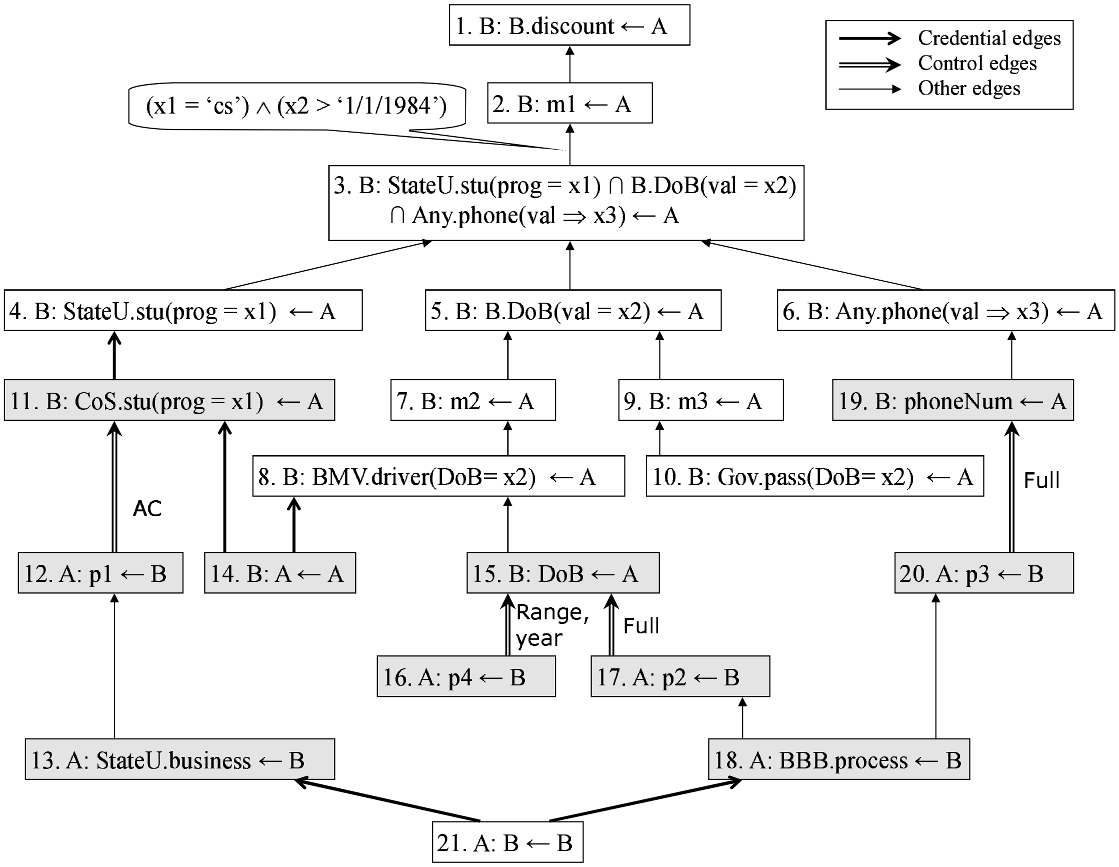
1. B: B.discount ← A
Credential edges
Control edges
Other edges
(x1 = ‘cs’) ∧ (x2 > ‘1/1/1984’)
2. B: m1 ← A
3. B: StateU.stu(prog = x1) ∩ B.DoB(val = x2) ∩ Any.phone(val ⇒ x3) ← A
4. B: StateU.stu(prog = x1) ← A
5. B: B.DoB(val = x2) ← A
6. B: Any.phone(val ⇒ x3) ← A
11. B: CoS.stu(prog = x1) ← A
7. B: m2 ← A
9. B: m3 ← A
19. B: phoneNum ← A
8. B: BMV.driver(DoB= x2) ← A
10. B: Gov.pass(DoB= x2) ← A
AC
Full
12. A: p1 ← B
14. B: A ← A
15. B: DoB ← A
20. A: p3 ← B
Range, year
Full
16. A: p4 ← B
17. A: p2 ← B
13. A: StateU.business ← B
18. A: BBB.process ← B
21. A: B ← B

Figure 6.4. 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.

(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 is 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.

7 RELATED WORK

7.1 Automated Trust Negotiation

Automated trust negotiation was introduced by Winsborough et al. [7], 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. [10] 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. [8] and Yu and Winslett [11] studied the problem of protecting contents of policies as well as credentials.

On the aspect of system architecture for trust negotiation, Hess et al. [76] 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. [46] introduced the TrustBuilder architecture for trust negotiation systems.

The problem of leaking attribute information was recognized by Winsborough and Li [9], Seamons et al. [77], and Yu and Winslett [78]. Winsborough and Li [9, 12, 79] 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 [80] 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 [81] proposed a framework for regulating service access and information release on the web. Their framework supports both certified attributes and uncertified attributes. Bertino, Ferrari, and Squicciarini proposed Trust-χ [82–85], a comprehensive XML-based framework for trust negotiations, specifically conceived for a peer-to-peer environment. Trust-χ presents a number of innovative features, such as the support for protection of sensitive policies, the use of trust tickets to speed up the negotiation, and the support of different strategies to carry on a negotiation.

7.2 Cryptographic Approaches to Automated Trust Negotiation

Recently, several cryptographic protocols have been proposed to address the limitations in ATN. For example, oblivious signature based envelopes [45], hidden credentials [22, 68], oblivious commitment based envelopes in Chapter 3, and secret handshakes [71, 73] can be used to handle policy cycle problems. Access control using pairing-based cryptography [86], anonymous identification [87], certified input private policy evaluation [88], hidden policies with hidden credentials in Chapter 5 (and also in [67, 89]), and policy-based cryptography [90] 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. While all the above protocols are useful tools and building blocks for ATN, they are not general enough to solve arbitrary trust negotiation problems in a systematic way.

7.3 Anonymous Credential Systems

Anonymous credential systems (also called pseudonym systems) [29–33] enable business transactions to be conducted in an anonymous yet authenticated manner. Similarly to OACerts, in anonymous credential systems, a user can choose which information in a credential, and which aspects of that information, to disclose or prove to another party. For example, suppose a credential asserts that Alice's age is 24. Then Alice can prove to Bob that her age is over 18 without revealing the exact value of her age. This property is desirable in trust negotiation because it minimizes the information revealed during an interaction and enhances privacy protection.

7.4 Secure Function Evaluation

Secure Function Evaluation (SFE) [23,26,41] is a powerful and general cryptographic primitive. In SFE, Alice and Bob each have private data (say, x_A for Alice and x_B for Bob), and they want to compute $f(x_A, x_B)$ where the function f is known to both Alice and Bob, and $f(x_A, x_B)$ is efficiently computable by someone who had both x_A and x_B. However, neither Alice nor Bob is willing to disclose his/her private data to the other or to a third party. Informally speaking, a protocol that involves only Alice and Bob, is said to be secure if, at its end, Alice and Bob have learned only $f(x_A, x_B)$. The history of the multi-party computation problem is extensive since it was introduced by [91] and extended by [41] and others. Broadly speaking, it has been established that there exists a secure protocol to evaluate any well-defined function, no matter how complex. However, [92] states that although the general secure multi-party computation problem is solvable in theory, using the solutions derived by these general results can be impractical. In other words, efficiency dictates the development of special solutions for special cases. Therefore, for efficiency reasons we might need to either transform the computation into a different form or provide a customized solution.

Selective Private Function Evaluation was introduced by Canetti *et al.* [93] whose goal is for Bob to compute a private function $f(x_{i_1}, \ldots, x_{i_m})$ over a subset of Alice's database $x = x_1, \ldots, x_n$ without revealing Bob's function. In their, the authors focused on the case where f and m are public but the m locations in the database are private to Bob.

Abadi and Feigenbaum [94] introduced the notion of Secure Circuit Evaluation. In Secure Circuit Evaluation, Alice has a private input x and Bob has a private circuit C. In the end Alice learns the value $C(x)$ but nothing else about C. Sander et al. [95] improved the previous results and gave an efficient one-round protocol for secure evaluation of circuits that have polynomial size and depth $O(\log n)$.

8 SUMMARY

In ATN, two parties exchange digitally signed credentials that contain attribute information to establish trust and make access control decisions. However, in existing approaches to ATN, there are several limitations due to the privacy constraints. In this thesis, we introduced a number of techniques that address these limitations. In particular,

- We proposed OACerts, an attribute certificate scheme especially designed for ATN. We presented an efficient and provably secure solution to policy-hiding access control using OACerts, which enables Bob to decide whether Alice's certified attribute values satisfy Bob's policy, without Bob learning any other information about Alice's attribute values or Alice learning Bob's policy.

- We gave an efficient protocol for Alice and Bob to negotiate trust, such that Alice does not learn Bob's credentials and policies, and Bob does not learn Alice's credentials and policies. The only information they learn is whether the trust between them can be established, or in other words, whether Alice is eligible for Bob's service or resource. Our work is a substantial extension of the state-of-the-art in privacy-preserving trust negotiations.

- 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.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.

[2] Ronald L. Rivest and Bulter Lampson. SDSI — A simple distributed security infrastructure, October 1996. *http://theory.lcs.mit.edu/∼rivest/sdsi11.html*.

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

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

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

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

[7] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88–102. IEEE Press, January 2000.

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

[9] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 92–103. IEEE Computer Society Press, June 2002.

[10] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.

[11] Ting Yu and Marianne 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.

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

[13] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.

[14] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[15] Tatsuaki Okamoto, Shigenori Uchiyama, and Eiichiro Fujisaki. Epoc: Efficient probabilistic public-key encryption. In *IEEE P1363: Protocols from other families of public-key algorithms*, November 1998.

[16] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[17] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136. Springer, 2001.

[18] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *Proceedings of the 8th Australasian Conference on Information Security and Privacy*, volume 2727 of *LNCS*, pages 350–364. Springer, 2003.

[19] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology: CRYPTO '84*, volume 196 of *LNCS*, pages 47–53. Springer, 1984.

[20] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology: CRYPTO '01*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.

[21] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *8th IMA International Conference on Cryptography and Coding*, volume 2260, pages 360–363. Springer, December 2001.

[22] Jason E. Holt, Robert W. Bradshaw, Kent E. Seamons, and Hilarie Orman. Hidden credentials. In *Proceedings of the 2nd ACM Workshop on Privacy in the Electronic Society*, pages 1–8, October 2003.

[23] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.

[24] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[25] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[26] Oded Goldreich. *The Foundations of Cryptography — Volume 2*. Cambridge University Press, May 2004.

[27] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[28] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 public key infrastructure certificate and CRL profile. IETF RFC 2459, January 1999.

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

[30] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, August 2000.

[31] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Proceedings of the 6th Workshop on Selected Areas in Cryptography*, volume 1758 of *LNCS*, pages 184–199. Springer, 1999.

[32] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology: EUROCRYPT '01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[33] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 21–30. ACM, nov 2002.

[34] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology: CRYPTO '97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[35] Ronald Cramer and Ivan Damgård. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *LNCS*, pages 424–441. Springer, 1998.

[36] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology: ASIACRYPT '02*, volume 2501 of *LNCS*, pages 125–142. Springer, December 2002.

[37] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology: EUROCRYPT '96*, volume 1070 of *LNCS*, pages 72–83. Springer, 1996.

[38] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *Public Key Cryptography: PKC'98*, volume 1431 of *LNCS*, pages 60–71. Springer, February 1998.

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

[40] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology: EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444, May 2000.

[41] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the 19th ACM Conference on Theory of Computing*, pages 218–229, May 1987.

[42] Sharon Boeyen, Tim Howes, and Patrick Richard. Internet X.509 public key infrastructure LDAPc2 schema. IETF RFC 2587, June 1999.

[43] Stephen Farrell and Russell Housley. An internet attribute certificate profile for authorization. IETF RFC 3281, April 2002.

[44] Eric Rescorla. *SSL, TLS: Designing, and Building Secure Systems*. Addison-Wesley, 2001.

[45] Ninghui Li, Wenliang Du, and Dan Boneh. Oblivious signature-based envelope. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pages 182–189. ACM Press, July 2003.

[46] Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, November/December 2002.

[47] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, feb 1989.

[48] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[49] Ninghui Li, Wenliang Du, and Dan Boneh. Oblivious signature-based envelope. *Distributed Computing*, 17(4):293–302, 2005.

[50] JCSI. Java cryptographic secure implementation. Wedgetail Communications, 2004.

[51] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay – Secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, pages 287–302. USENIX, 2004.

[52] Murat Kantarcioglu and Chris Clifton. Assuring privacy when big brother is watching. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 88–93. ACM Press, 2003.

[53] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[54] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 639–648. ACM Press, 1996.

[55] Nicholas J. Pippenger. Generalized connectors. Technical Report RC-6532, IBM Res. Rep., 1977.

[56] Clark D. Thompson. Generalized connection networks for parallel processor intercommunication. *IEEE Transactions on Computers*, 27(12):1119–1125, December 1978.

[57] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a nonblocking network. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 149–158. ACM Press, 1990.

[58] Ellen Witte Zegura. Evaluating blocking probability in generalized connectors. *IEEE/ACM Transactions on Networking*, 3(4):387–398, 1995.

[59] Yuanyuan Yang and Gerald M. Masson. The necessary conditions for clos-type nonblocking multicast networks. *IEEE Transactions on Computers*, 48(11):1214–1227, 1999.

[60] Ju P. Ofman. A universal automaton. *Transactions of the Moscow Math Society*, 14:200–215, 1965.

[61] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology: CRYPTO '89*, volume 435 of *LNCS*, pages 547–557. Springer, 1989.

[62] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of SIAM Symposium on Discrete Algorithms*, pages 448–457, January 2001.

[63] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology: CRYPTO '84*, volume 196 of *LNCS*, pages 10–18. Springer, 1985.

[64] Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In *Proceedings of the 1st International Workshop on Practice and Theory in Public Key Cryptography*, pages 117–134. Springer, 1998.

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

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

[67] Keith B. Frikken, Mikhail J. Atallah, and Jiangtao Li. Hidden access control policies with hidden credentials. In *Proceedings of the 3rd ACM Workshop on Privacy in the Electronic Society*, October 2004.

[68] Robert Bradshaw, Jason Holt, and Kent Seamons. Concealing complex policies with hidden credentials. In *Proceedings of 11th ACM Conference on Computer and Communications Security*, pages 146–157, October 2004.

[69] Keith B. Frikken. *Secure and Private Online Collaboration*. PhD thesis, Purdue University, West Lafayette, Indiana, 2005.

[70] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology: EUROCRYPT '04*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004.

[71] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *Proceedings of the IEEE Symposium and Security and Privacy*, pages 180–196, May 2003.

[72] Ninghui Li and John C. Mitchell. 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, pages 58–73. Springer, January 2003.

[73] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret handshakes from CA-oblivious encryption. In *Advances in Cryptology: ASIACRYPT '04*, pages 293–307, December 2004.

[74] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 44–54, May 1997.

[75] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[76] Adam Hess, Jared Jacobson, Hyrum Mills, Ryan Wamsley, Kent E. Seamons, and Bryan Smith. Advanced client/server authentication in TLS. In *Network and Distributed System Security Symposium*, pages 203–214, February 2002.

[77] Kent E. Seamons, Marianne Winslett, Ting Yu, Lina Yu, and Ryan Jarvis. Protecting privacy during on-line trust negotiation. In *2nd Workshop on Privacy Enhancing Technologies*. Springer-Verlag, April 2002.

[78] Ting Yu and Marianne 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, October 2003.

[79] William H. Winsborough and Ninghui 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, November 2002.

[80] Keith Irwin and Ting Yu. Preventing attribute information leakage in automated trust negotiation. In *Proceedings of the 12th ACM conference on Computer and Communications Security*, pages 36–45, 2005.

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

[82] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-χ: An XML framework for trust negotiations. In *11 International Conference on Communications and Multimedia Security*, pages 146–157, October 2003.

[83] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. X-TNL: An XML-based language for trust negotiations. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 81–84, June 2003.

[84] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Privacy-preserving trust negotiations. In *4th International Workshop on Privacy Enhancing Technologies*, pages 283–301, May 2004.

[85] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-χ: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.

[86] Nigel Smart. Access control using pairing based cryptography. In *Proceedings of the Cryptographers' Track at the RSA Conference 2003*, pages 111–121. Springer-Verlag LNCS 2612, April 2003.

[87] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. Anonymous identification in ad hoc groups. In *Advances in Cryptology: EUROCRYPT 2004*, pages 609–626, May 2004.

[88] Jiangtao Li and Ninghui Li. Policy-hiding access control in open environment. In *Proceedings of the 24nd ACM Symposium on Principles of Distributed Computing*, pages 29–38. ACM Press, July 2005.

[89] Keith B. Frikken, Jiangtao Li, and Mikhail J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *Proceedings of 13th Network and Distributed System Security Symposium*, pages 157–172, February 2006.

[90] Walid Bagga and Refik Molva. Policy-based cryptography and applications. In *Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, February 2005.

[91] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society Press, 1982.

[92] Shafi Goldwasser. Multi-party computations: Past and present. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pages 1–6, 1997.

[93] Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of the 20th ACM symposium on Principles of Distributed Computing*, pages 293–304. ACM Press, 2001.

[94] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation: A protocol based on hiding information from an oracle. *Journal of Cryptology*, 2(1):1–12, 1990.

[95] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, page 554. IEEE Computer Society, 1999.

VITA

VITA

Jiangtao Li was born in Huaian, China. At the age of fifteen, he entered the Special Class for the Gifted Young at the University of Science and Technology of China (USTC), where he obtained his bachelor's degree in computer science in July 1999. After he graduated from USTC, he spent one year at the Department of Statistics of Purdue University as a Ph.D. student.

In May 2000, he enrolled in the Department of Computer Science of Purdue University where he obtain his master's degree in May 2002. In the summer of 2003, he performed three months of his Ph.D. research at Microsoft Corporation. He received the degree of Doctor of Philosophy in May 2006 under the direction of Professor Mikhail Atallah and Professor Ninghui Li. His research interests are in computer security, applied cryptography, electronic commerce, bioinformatics, with a focus on privacy and data protection.