CERIAS Tech Report 2005-31

POLICY-HIDING ACCESS CONTROL IN OPEN ENVIRONMENT

by Jiangtao Li and Ninghui Li

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

Policy-Hiding Access Control in Open Environment *

Jiangtao Li and Ninghui Li
CERIAS and Department of Computer Science, Purdue University
250 N. University Street, West Lafayette, IN 47907, USA
{jtli, ninghui}@cs.purdue.edu
Phone: (765) 496-6767, (765) 496-6756
FAX: (765) 494-0739

Abstract

In Trust Management and attribute-based access control systems, access control decisions are based on the attributes (rather than the identity) of the requester: Access is granted if Alice's attributes in her certificates satisfy Bob's access policy. In this paper, we 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. To enable policy-hiding access control, we introduce the notion of Certified Input Private Policy Evaluation (CIPPE). Our construction for CIPPE uses Yao's scrambled circuit protocol and two new techniques introduced in this paper. One novel technique is constructing circuits with uniform topology that can compute arbitrary functions in a family. The other technique is Committed-Integer-based Oblivious Transfer.

Keywords

Privacy, Access Control, Cryptographic Commitment, Cryptographic Protocol, Secure Function Evaluation, Automated Trust Negotiation, Digital Certificate, Oblivious Transfer

1 Introduction

In Trust Management and attribute-based access control systems [5, 48, 23, 17, 38, 37], access control decisions are based on attributes of clients, which are often documented by public key certificates. Each certificate 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 certificates are digitally signed, they can serve to introduce strangers to one another without online contact with the Certificate Authorities (CA's). 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 1 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

*This work is supported by NSF ITR grant CCR-0325951 and by sponsors of CERIAS.

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 one's birth-date, current salary, and the length of the current employment; however, Bob considers the details of his policy to be commercial 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 the exact birth-date of Alice. 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 security 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 (which usually renders an economically-motivated policy less effective). This is particularly important for policies that are not incentive-compatible in economic terms. Finally, it is important to point out that a process that protects Alice's certificates 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.

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) [56, 31, 30] (e.g., Yao's scramble circuit protocol [56]) 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 *Oblivious Attribute Certificates* (OACerts) proposed in [35]. In an OACert, attribute values are not stored in the clear; instead, a cryptographic commitment for each of these values is stored in the certificate. A certificate authority (CA) generates the commitments of Alice's attribute values and stores them in the certificates. Alice is able to disclose her certificates to Bob without revealing her attribute values. OACerts can be integrated into current standards for public-key certificates such as X.509 Public Key Infrastructure Certificates [6, 34] and X.509 Attribute Certificates [25]. A prototype of OACerts as been implemented by storing the commitments in X.590v3 extension fields [35]. Using OACerts, the policy-hiding access control problem becomes that Alice inputs her committed attributes which are documented in her OACerts and Bob inputs his policy, they want to learn whether Alice's committed attributes satisfy Bob's policy without revealing the other party's private input.

In this paper, we introduce the notion of *Certified Input Private Policy Evaluation (CIPPE)*, which enables policy-hiding access control using OACerts. Formal definition of CIPPE is given in Section 3. 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 $f(x_1, \cdots, x_n)$. Bob should not learn anything about $x_1, \ldots, x_n$; and Alice should not learn more than the fact that $f \in \mathcal{F}$.

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

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 [56, 39]. 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. The CIPPE protocol is efficient; and we believe it can be deployed in practice (see [39] for an implement of the scramble circuit protocol by Malkhi et al.).

The rest of this paper is organized as follows. We first describe how CIPPE can be used to enable policy-hiding access control in Section 2. Then we give a formal definition of CIPPE in Section 3. In Section 4, we review two cryptographic building blocks that we use, namely, the Pedersen commitment scheme and the scrambled circuit protocol for 2-SFE. 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 7 we give an efficient construction for CIPPE. We discuss the related work in Section 8 and conclude our paper in Section 9.

2 Using 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 policy-hiding access control in Example 1 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.

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 three attributes: birth-date, current salary, and 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 [34]. Recall that OACerts can be used as a regular digital certificate (e.g., X.509 certificate) 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.

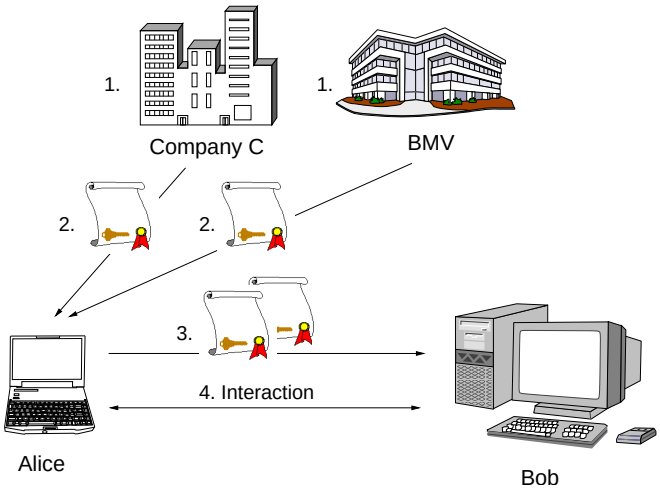

Figure 1: An example of policy-hiding access control procedures between Alice and Bob.

Figure 1 depicts how CIPPE can be used in the trust negotiation process. 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 at all.

3 Definition of Certified Input Private Policy Evaluation (CIPPE)

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

Definition 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 such that $1 \leq i \leq n$, 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 2, CA does not directly communicate with S. Instead, CA verifies C's attribute values before computing the commitments and stores $c_1, \ldots, c_n$ into C's OACert certificate. The certificate is then sent by C to S, enabling S to have the commitment values as if they are sent from CA. Here we abstract these steps away to have CA sending c_i to S. We stress that CA does *not* 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 0 or 1, i.e., $f : ([0..2^\ell - 1])^n \to \{0, 1\}$. 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 $f(x_1, \ldots, x_n)$.

To avoid unnecessarily cluttering the exposition, in Definition 1 we assume that there is only one CA in the CIPPE scheme, and that $x_1, \ldots, x_n$ are equal-length and are committed under the same commitment parameters. The definition of the CIPPE scheme 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 We consider security against three kinds of adversaries [13, 30]. An *adversary* is a probabilistic interactive Turing Machine [32]. 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 [13] is an adversary who may deviate arbitrarily from the protocol, as long as its behavior appears honest to parties executing the protocol. A *malicious* adversary is an adversary who may behave arbitrarily. To show a CIPPE protocol is secure, it should be proved that neither C nor S could gain more information than in the ideal model, in which there is a trusted third party who receives $x_1, \ldots, x_n$ from C and f from S, and sends $f(x_1, \ldots, x_n)$ to both C and S. See Appendix A for the formal definitions.

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.

4 Cryptographic Assumptions and Tools

In this section we first give the cryptographic assumptions and then briefly review two cryptographic tools that we use for our CIPPE construction: the Pedersen commitment scheme [45] and the scrambled circuit protocol for 2-SFE [56].

Basic Cryptographic Assumptions The security of our CIPPE protocol is based on the following two standard assumptions in cryptography. A function μ is called *negligible* in the security parameter σ if, for every polynomial p, $|\mu(\sigma)|$ is smaller than $1/|p(\sigma)|$ for large enough σ; otherwise, it is *non-negligible*.

1. *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.
2. *Decision 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.

The Pedersen Commitment Scheme [45]

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. T picks g to be a generator of G_q, the unique order-q subgroup of $\mathbb{Z}_p^*$. We use $s \in_R \mathbb{Z}_q$ to denote that s is uniformly randomly chosen from $\mathbb{Z}_q$. T picks $s \in_R \mathbb{Z}_q$ and computes $h = (g^s \bmod p)$. T keeps the value s secret and makes the values p, q, g, h public.

Commit The domain of the committed values is $\mathbb{Z}_q$. For a party A to commit an integer $x \in \mathbb{Z}_q$, A chooses $r \in_R \mathbb{Z}_q$ and computes the commitment $c = (g^x h^r \bmod p)$.

Open To open a commitment c, A reveals x and r, and a verifier verifies whether $c = (g^x h^r \bmod p)$.

We use a trusted third party T to generate the parameters of the Pedersen commitment scheme because the setup algorithm is run by a trusted CA in the CIPPE setting. 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 x 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 committer to open a value x' other than x in the open phase of the commitment scheme.

The scrambled circuit protocol for 2-SFE The scrambled circuit protocol was developed by Yao [56] (See Appendix B for more detailed description of the protocol). This protocol runs between two players: a *generator* and an *evaluator*. To compute $f(x, y)$, the generator first constructs a circuit for computing f. The generator then constructs a scrambled version of the circuit and sends the scrambled circuit to the evaluator for evaluation. In a scrambled circuit, each wire is associated with two random numbers, one corresponds to 0 and the other to 1. Before the evaluation, the evaluator uses oblivious transfer to obtain the random values corresponding to each bit of the evaluator's private input x. During the evaluation, the evaluator learns exactly one random value for each internal wire, yet she doesn't know whether it corresponds to 0 or 1. Finally the evaluator sends the outcome of the evaluation to the generator, who recovers the final result.

5 Building Circuits with Uniform Topological Structure

When a circuit is scrambled, 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 the 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 $1 \le i_1, i_2, \ldots, i_m \le n$, each op_i is one of the following predicates $\{=, \neq, >, <, \ge, \le\}$, and p is a disjunctive (or conjunctive) normal form in which the number of disjuncts (or conjuncts) is no more than λ.

Loosely speaking, 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 1 in Section 1, Bob (the bank) can set $n = 3, m = 8, \lambda = 4$, and the policy function is of the form:

$$f(x_1, x_2, x_3) = (x_1 \ge 30 \wedge x_2 \ge 43000 \wedge x_3 > 6) \vee (x_1 \ge 25 \wedge x_2 \ge 45000 \wedge x_3 > 12),$$

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 which values they are compared with, and how many times each attribute is compared, etc.

If Bob builds a circuit for $f(x_1, x_2, x_3)$ in the 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 is that each attribute may be compared multiple times, and we want to hide the number of times it is compared. 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([46, 50, 2, 59, 55], to list a few). We believe that some of these networks may be useful for constructing circuits for families of functions beyond the ones considered in this paper.

Basic circuit components we introduce three basic circuit components that will be used in our construction:

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 $x \ge y$ and $x \le y$ 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.
   - 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 $\bigvee_{i=0}^{\ell-1} (x_i \neq y_i)$

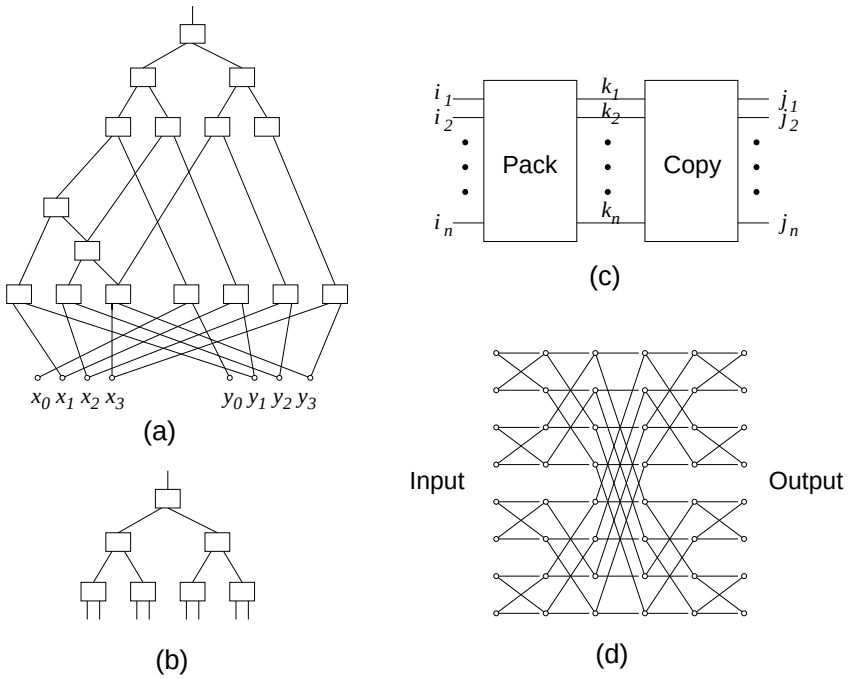

Figure 2: Basic circuit components: (a) the structure of 4-bit comparison circuits, (b) the structure of 8-input logical operation circuits, (c) the high-level schema for a generalizer circuit, (d) an (8,8)-generalizer,

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 2(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 2(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 a n inputs and n outputs switching network, it passes each input i to zero or more outputs. The existence of (n, n)-generalizer with $O(n)$ gates has demonstrated nonconstructively by Pipenger [46]. Ofman [43] gives a construction of a generalizer using the schema shown in Figure 2(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 Ofman proposed requires $3n \log n$ gates. Thompson [50] 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 2(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 (around $2\ell m \log n$ 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 (around $5\ell m$ 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 is 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 (around λm gates).

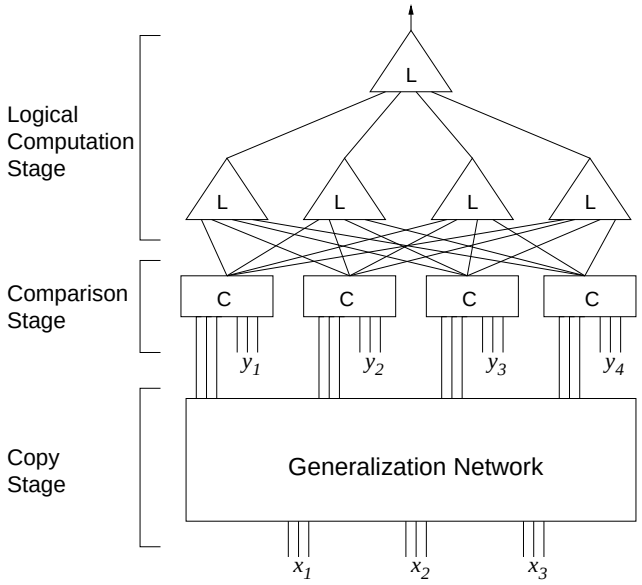

Figure 3: An example circuit structure for the family $\mathcal{F}$ of functions with parameters $\ell = 3$, $n = 3$, $m = 4$, and $\lambda = 4$. There are 4 comparison circuits in the comparison stage, and 5 logical operation circuits in the logical computation stage.

Figure 3 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).

6 Committed-Integer based Oblivious Transfer (CIOT) Protocol

To build a CIPPE protocol using the scrambled circuit protocol (see Section 4), 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 corresponds 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 [4, 41] for each commitment.

Protocol 1 (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 $x_{\ell-1} \ldots x_1 x_0$ 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 $(k_0^0, k_0^1), \ldots, (k_{\ell-1}^0, k_{\ell-1}^1)$. Both the sender and receiver have c.

Output The receiver learns $k_0^{x_0}, \ldots, k_{\ell-1}^{x_{\ell-1}}$. 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 $\prod_{i=0}^{\ell-1} (c_i)^{2^i} = \prod_{i=0}^{\ell-1} (g^{a_i} h^{r_i})^{2^i} = g^{\sum_{i=0}^{\ell-1} a_i 2^i} h^{\sum_{i=0}^{\ell-1} r_i 2^i} = g^a h^r = c$
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 [22] (modified to have messages from a subgroup [51]), 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 uniformly randomly 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.

Note that both the sender and receiver need $O(\ell)$ modular exponentiation. More precisely, the sender needs 2ℓ modular exponentiation, and receiver needs 4ℓ modular exponentiation. The sender does not learn anything from the CIOT protocol. Under the DDH assumption and the DL assumption on G_q, the receiver learns at most one value per (k_i^0, k_i^1) pair; and if the receiver learns ℓ keys, these values must be $k_0^{x_0}, \ldots, k_{\ell-1}^{x_{\ell-1}}$. Observe that it is possible for an adversarial receiver to learn keys other than $k_0^{x_0}, \ldots, k_{\ell-1}^{x_{\ell-1}}$; however, in such case, she cannot get all ℓ keys. This is sufficient for the security of our CIPPE protocol, because if the client cannot get a key for each input wire, then she cannot evaluate the scrambled circuit to get an output. Due to space limitation, we put security properties and formal proofs of CIOT protocol in Appendix C.

7 Our CIPPE Protocol

We now give the CIPPE protocol which follows Definition 1, and specify what each participant does in each step.

Protocol 2 (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, $1 \le i \le n$, CA picks $r_i \in_R \mathbb{Z}_q$ and computes the commitment $c_i = (g^{x_i} h^{r_i} \mod p)$. 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 5. 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 specified in Section 5, then scrambles the circuit in the same manner as the generator scrambles the circuit in the scrambled circuit protocol. 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 $\langle \eta_0 = E_{k_t^0}[\,0^\sigma\,], \eta_1 = E_{k_t^1}[\,1^\sigma\,] \rangle$ to C.
3. *Coding the input:* For each x_i where $1 \le i \le n$, 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 aborts; this happens only when S intentionally misbehave. 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 $k = k_t^1$.

Note that the idea of committing the output in step 2 comes from [42, 39] 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. Because 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. Our CIPPE protocol is secure both in the honest-but-curious model and in the weak-honest model. In the malicious model, the protocol is secure against the client. A malicious server may learn one bit more information than she is allowed, by constructing a malfunction circuit. Considering the damage caused by being detected to be dishonest, this small extra gain does not seem to warrant such malicious behavior in the application scenarios we consider. Due to space limitation, we put security properties and formal proofs of CIPPE protocol in details in Appendix D.

8 Related Work

Automated Trust Negotiation Our work is closely related to a growing body of work on Automated Trust Negotiation (ATN) [54, 58, 57, 53] whose goal is to enable clients and servers to establish trust in each other through cautious, iterative, bilateral disclosure of sensitive certificates and policies. Recent works on using cryptographic protocols for ATN include Hidden Credentials [33, 7, 26], Secret Handshakes [3], and Oblivious Signature Based Envelope [36]. While these schemes are useful for scenarios where policies are based on attributes such as secret clearance or memberships in some secret underground movements, they are not suitable for the kind of e-commerce scenarios such as Example 1. Using any of these schemes, the server could send an encrypted message to a client such that the client can decrypt if and only if the client has certificates whose contents are the same as those identified by the server's policy; at the same time, the server does not know whether the client has those certificates or not. These schemes can implement policy-hiding access control when the servers' policies have very specific forms. In Example 1, if Bob's loan approval policy is either Alice's birthdate is April 1st, 1974 or Alice's salary is exactly \$60,000, then policy-hiding access control can be achieved using these existing schemes. However, for the kind of policies in Example 1, where many possible attribute values would satisfy a policy, these schemes do not work well.

Secure Function Evaluation Secure Function Evaluation (SFE) [56, 31, 30] is a powerful and general cryptographic primitive. It allows two or more parties to jointly compute some function while hiding their inputs to each other. CIPPE may be cast as a special case of 2-SFE problem: (1) take $x_1, \ldots, x_n$ as Alice's private input and make the commitment verification part of the public function, (2) treat the description of Bob's private function f as part of Bob's private input, and (3) make the public function to be evaluated a universal circuit that takes f's description, $x_1, x_2, \cdots, x_n$, and $c_1, c_2, \cdots, c_n$, and computes $f(x_1, \cdots, x_n)$ (this universal circuit is similar in concept to a universal Turing Machine). However, applying the general solution in this case is very inefficient, as the circuit for verifying commitments and the universal circuit are very large.

The idea of committing local inputs before the function evaluation has appeared in the SFE literature (e.g., [16, 31, 29, 9]) to ensure the correctness of the computation. This concept is substantially different from our model, where Alice's input is certified by a trusted third party in CIPPE. Cachin and Camenisch [9] introduced the notion of fair secure computation where a partially trusted third party T participates to ensure the fairness of the computation. Their work [9] is different from CIPPE in that (1) T's job is to achieve fairness instead of certifying Alice's input (in fact, Alice's input is even private to T), and (2) the function to compute is public.

Selective Private Function Evaluation (SPFE) was introduced by Canetti et al. [14] 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 [1] 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. [49] improved the previous results and gave an efficient one-round protocol for secure evaluation of circuits that have polynomial size and depth $O(\log n)$. In these protocols, Alice can choose which input value to use in the circuit evaluation. It is not clear how Secure Circuit Evaluation protocols can be applied to CIPPE because the client's input in CIPPE is committed and certified by a trust CA.

OACerts and Anonymous Credentials Our work builds directly on OACerts developed in [**?**]. The ideas of storing commitments of attribute values in certificates appeared in the literature on anonymous credentials [15, 8, 11, 10]. Thus it is possible to replace OACerts and use anonymous credentials in policy-hiding access control and CIPPE. Note that using zero-knowledge proof protocols [18, 40, 27, 8, 11] together with OACerts or anonymous credentials, Alice can prove that her attribute values in her certificates have certain properties without revealing any other information about her attributes. However, in order for Alice to use such techniques to prove that she satisfies Bob's policy, she needs to know the policy. Therefore, zero-knowledge proof protocols are not suitable for policy-hiding access control.

Oblivious Transfer Crépeau [19] introduced the notion of Committed Oblivious Transfer (COT). In COT, Bob commits two bits: a_0 and a_1, and Alice commits a bit b. In the end, Alice learns a_b without learning anything else, while Bob learns nothing. Garay et al. [28] gave an efficient construction of COT in the universal composability framework. The CIOT protocol we propose in this paper differs from the COT protocols in that the receiver's input in CIOT is a committed integer instead of a bit. Finally, the details of our CIOT protocol are reminiscent of the techniques used in the oblivious transfer protocols [41, 52], zero-knowledge proofs of that a committed number belongs to an interval [40, 21], and anonymous fingerprinting [44].

9 Conclusion and Future Work

We have presented an efficient and provably secure solution to policy-hiding access control, 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. Our approach uses OACerts and CIPPE. Our construction for CIPPE uses Yao's scrambled circuit protocol and two novel techniques, one is constructing topologically uniform circuits that can compute arbitrary functions in a function family, the other is the Committed-Integer-based Oblivious Transfer (CIOT) protocol. Future work includes constructing efficient topologically uniform circuits for function families other than the one we studied in Section 5.

References

[1] 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.

[2] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a nonblocking network. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 149–158. ACM Press, 1990.

[3] 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.

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

[5] 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.

[6] Sharon Boeyen, Tim Howes, and Patrick Richard. Internet X.509 Public Key Infrastructure LDAPc2 Schema. IETF RFC 2587, June 1999.

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

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

[9] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Advances in Cryptology: CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2000.

[10] 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, CCS '02*, pages 21–30. ACM, nov 2002.

[11] 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 *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.

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

[13] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 639–648. ACM Press, 1996.

[14] 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 twentieth annual ACM symposium on Principles of distributed computing*, pages 293–304. ACM Press, 2001.

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

[16] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology: CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119. Springer, 1988.

[17] 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.

[18] 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 *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.

[19] Claude Crépeau. Verifiable disclosure of secrets and applications (abstract). In *Advances in Cryptology: EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 150–154. Springer, 1990.

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

[21] 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.

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

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

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

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

[26] 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.

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

[28] Juan Garay, Philip MacKenzie, and Ke Yang. Efficient and universally composable committed oblivious transfer and applications. In *Theory of Cryptography, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2004.

[29] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM Press, 1998.

[30] Oded Goldreich. Secure multi-party computation, October 2002.

[31] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, May 1987.

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

[33] 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*, October 2003.

[34] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF RFC 2459, January 1999.

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

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

[37] 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.

[38] 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.

[39] 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.

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

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

[42] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM Press, 1999.

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

[44] Joon S. Park and Ravi Sandhu. Smart certificates: Extending X.509 for secure attribute services on the web. In *Proceedings of the 22nd National Information Systems Security Conference*, October 1999.

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

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

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

[48] Ronald L. Rivest and Bulter Lampson. SDSI — a simple distributed security infrastructure, October 1996. Available at $\mathrm{http://theory.lcs.mit.edu/\sim rivest/sdsi11.html}$.

[49] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for nc1. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, page 554. IEEE Computer Society, 1999.

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

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

[52] Wen-Guey Tzeng. Efficient 1-out-n oblivious transfer schemes. In *PKC '02: Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, number 2274 in Lecture Notes in Computer Science, pages 159–171. Springer, 2002.

[53] 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.

[54] 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.

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

[56] 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.

[57] 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.

[58] 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 (TISSEC)*, 6(1):1–42, February 2003.

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

A Formal Security Definition of CIPPE

The security definitions we use follow [30, 12, 13]. The security of 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). Recall in the CIPPE scheme, C inputs x, S inputs f, and finally both C and S output $f(x)$. The ideal model differs for honest-but-curious and malicious adversaries (see [30] for a formal definition). 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, and (3) may abort the protocol prematurely. When we consider weak-honest, we cannot prevent an adversary from substituting her local input.

1. For the ideal model with honest-but-curious adversaries, the client sends x to the TTP, the server sends f to the TTP, and finally both the client and the server receive $f(x)$. An honest party outputs her output from the TTP, whereas a honest-but-curious party outputs an arbitrary function from her initial input and the output she obtained from the TTP.

2. 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.

3. The ideal model for malicious adversaries is similar to the ideal model for honest-but-curious adversaries, but differs in that 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 2 (The Ideal Model) Let $f : \{0,1\}^* \to \{0,1\}$ be a functionality. Let (C, S) be a pair of probabilistic polynomial-time machines representing the client and server in the ideal model. Such a pair is *admissible*, if at least one of (C, S) is honest. The joint execution of f under (C, S) in the ideal model, denoted $\mathsf{IDEAL}_{C,S}(x, f)$, is defined as the output pair of C and S from the above ideal execution. For instance, if C is malicious and terminates the computation prematurely, the $\mathsf{IDEAL}_{C,S}(x, f) = (C(x, f(x')), \perp)$; if C never aborts, $\mathsf{IDEAL}_{C,S}(x, f) = (C(x, f(x')), f(x'))$ where x' is the input C gives to the TTP.

Definition 3 (The Real Model) Let f be as in Definition 2. Let Π be a CIPPE protocol for computing f. Furthermore, let (C, S) be a pair of probabilistic polynomial-time machines representing the client and server in the real model. Such a pair is *admissible*, if at least one of (C, S) is honest, i.e., follows the protocol Π. Then, the joint execution of Π under (C, S) in the real model, denoted $\mathsf{REAL}_{\Pi,C,S}(x, f)$, is defined as the output pair of C and S from the protocol interaction.

Definition 4 (Security) Let f and Π be as in Definition 3. 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 $\mathsf{REAL}_{\Pi,C^*,S^*}(x, f)$ is computationally indistinguishable from $\mathsf{IDEAL}_{C,S}(x, f)$.

B The scrambled circuit protocol for 2-SFE

The scrambled circuit protocol was developed by Yao [56]. This protocol runs between two players: a *generator* and an *evaluator*. In the scrambled circuit 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 recover the final result. Our CIPPE protocol builds upon the scrambled circuit protocol, where the generator is the server (Bob) and the evaluator is the client (Alice).

Protocol 3 (Scrambled Circuit Protocol) Let x be the evaluator's input, and y be the generator's input. Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be a function known to both parties. In the end, both parties learn $f(x, y)$. The scrambled circuit 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 $\{0,1\}^2 \to \{0,1\}$ 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 $1 \le i \le t$.
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 $1 \le a, b, c \le t$.
   (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 garbages 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 [47, 24, 30] 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. She will succeed only 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 scrambled circuit protocol is secure in honest-but-curious model [56, 30]. Let us briefly discuss the security of the scrambled circuit protocol. As for the security of each gate, it is necessary to construct the table using a *non-malleable* encryption scheme [20] (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.

C Security Properties and Proofs for CIOT protocol

Theorem 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 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 [51]. 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 $h^r = g h^{r'}$; she can effectively compute $\log_g(h) = (r - r')^{-1} \bmod q$, which contradicts the DL assumption. ∎

Theorem 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 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. ∎

D Security Properties and Proofs for CIPPE protocol

Theorem 4 *The CIPPE protocol is secure in the honest-but-curious model.*

Proof. In the honest-but-curious model, the security definition in the ideal model (Definition 4) is equivalent to the definition based on simulation. That is, it is sufficient to show that, C (and S), given her own input and $f(x_1, \ldots, x_n)$, can simulate the execution of the protocol.

If, for any $x_1, \ldots, x_n \in [0, 2^\ell - 1]$ and $f \in \mathcal{F}$, the distribution of C's view from the real model cannot be distinguished from a simulation of this view that only uses $x_1, \ldots, x_n, r_1, \ldots, r_n, c_1, \ldots, c_n$ and $f(x_1, \ldots, x_n)$, then clearly C cannot learn anything about S's private function f. Similarly, if distribution of S's view cannot be distinguished from a simulation of this view that only uses $f, c_1, \ldots, c_n$ and $f(x_1, \ldots, x_n)$, then S cannot learn anything about C's input $x_1, \ldots, x_n$.

The simulator for S is constructed as follows. Assume the simulator has the knowledge of $f, c_1, \ldots, c_n$ and $f(x_1, \ldots, x_n)$.

1. In step 1 of the client-server interaction phase, given $f \in \mathcal{F}$, the simulator constructs a circuit for f and scrambled it.

2. In step 3, S runs the CIOT protocol n times with C in the real model. For each of C's input x_i, S receives ℓ commitments, denoted as $c_{i,0}, c_{i,1}, \ldots, c_{i,\ell-1}$, where $c_{i,0} = c_i \prod_{j=1}^{\ell-1}(c_{i,j})^{-2^j}$. The simulator chooses $\ell - 1$ random values from G_q, denoted as $c'_{i,1}, \ldots, c'_{i,\ell-1}$, and computes $c'_{i,0} = c_i \prod_{j=1}^{\ell-1}(c'_{i,j})^{-2^j}$.

3. In step 5, S receives from C the key corresponding the result $k \in \{k_t^0, k_t^1\}$. If $f(x_1, \ldots, x_n) = 0$, the simulator generates k_t^0; otherwise, $f(x_1, \ldots, x_n) = 1$, the simulator generates k_t^1.

We briefly show that the real and simulated views for S are perfectly indistinguishable. In real views, S gets $c_{i,0}, c_{i,1}, \ldots, c_{i,\ell-1}$ for each x_i, whereas in simulated views, S gets $c'_{i,0}, c'_{i,1}, \ldots, c'_{i,\ell-1}$. By Theorem 1, the distributions of $c_{i,1}, \ldots, c_{i,\ell-1}$ are exactly the same as the distribution of any commitment under the Pedersen commitment scheme, thus the distribution of $(c_{i,0}, c_{i,1}, \ldots, c_{i,\ell-1})$ and $(c'_{i,0}, c'_{i,1}, \ldots, c'_{i,\ell-1})$ are perfectly indistinguishable.

Assume the simulator for C has the knowledge of $x_1, \ldots, x_n, r_1, \ldots, r_n, c_1, \ldots, c_n$ and $f(x_1, \ldots, x_n)$. Let us examine what C receives in the real model. C receives a scrambled circuit, one key per input wire (Theorem 2 guarantees that C could get only one key per input wire), and $\langle \eta_0 = E_{k_t^0}[0^\sigma], \eta_1 = E_{k_t^1}[1^\sigma] \rangle$. During the circuit evaluation phase, C learns one key per wire, and eventually learns k, the key of the output wire. The simulator for C is constructed as follows.

1. As the circuits for computing any functions $\mathcal{F}$ has same topological structure by construction, we assume the topology of the scrambled circuit is public.

2. The simulator generates t random keys $k'_1, k'_2, \ldots, k'_t$ and assigns k'_i for wire w_i where $1 \leq i \leq t$.

3. For each gate g_i in the circuit, let w_a and w_b be the corresponding input wires and w_c be the corresponding output wire. The simulator chooses randomly three message m'_1, m'_2, and m'_3 from the ciphertext space. The simulator computes $m'_4 = E_{k'_b}\left[E_{k'_a}[k'_c || 0^\sigma]\right]$. The simulator randomly permutes $\{m'_1, m'_2, m'_3, m'_4\}$.

4. If $f(x_1, \ldots, x_n) = 0$, the simulator computes $\eta'_0 = E_{k'_t}[0^\sigma]$, and chooses a random message η'_1 from the ciphertext space. If $f(x_1, \ldots, x_n) = 1$, the simulator computes $\eta'_1 = E_{k'_t}[1^\sigma]$, and chooses a random message η'_0 from the ciphertext space.

We briefly sketch that the real and simulated views for C are computationally indistinguishable. Assume the encryption scheme is secure, C can learn only one key per wire. The keys C learns for each wire are computationally distinguishable from $k'_1, k'_2, \ldots, k'_t$. For each gate g_i, $\{m_{0,0}, m_{0,1}, m_{1,0}, m_{1,1}\}$ in real model is

computationally indistinguishable from the permuted set $\{m'_1, m'_2, m'_3, m'_4\}$ in the simulated views. $\langle \eta_0, \eta_1 \rangle$ in the real model is computationally indistinguishable from $\langle \eta'_0, \eta'_1 \rangle$.

Therefore, our CIPPE protocol is secure against honest-but-curious adversaries. ∎

Theorem 5 *The CIPPE protocol is secure against malicious clients.*

Proof Sketch. Due to the space limitation, we only sketch this proof as well as the next two. The formal proofs will be given in the full version of this paper.

The correctness of the protocol against malicious clients relies on the security of the CIOT protocol. Theorem 3 guarantees that C gets only the keys corresponding to her committed input. Assume that the encryption scheme used in the scrambled circuit is secure, C learns at most one key per wire from the scrambled circuit. As S is honest and builds the circuit correctly, C can only get the key of the output wire corresponding to $f(x_1, \ldots, x_n)$.

As for server's privacy, the reasoning is similar to the correctness of the protocol. Because S constructs the circuit in a way that all circuits have the same topological structure, C cannot distinguish f from other $f' \in \mathcal{F}$ from circuit topology. Again, assume that the encryption scheme used in the scrambled circuit is secure and CIOT is secure, C learns at most one key per wire from the scrambled circuit, C cannot learn which gate computes which function. Thus, even malicious client cannot further information about S's private function. ∎

Theorem 6 *The CIPPE protocol is secure against weak-honest servers.*

Proof Sketch. To prove the security against weak-honest servers, we show that for every probabilistic polynomial-time machine S^* in the real model, there is a probabilistic polynomial-time machine S in the ideal model, such that the views of parties (C, S) in the ideal model is computationally indistinguishable from the views of (C, S^*) in the real model.

Before we prove the security, we briefly analyze the possible server behavior in the real model. For each instance of the CIOT protocol, S receives ℓ commitments, then it sends 2ℓ ElGamal encryptions to C. Weak-honest server behavior of in the CIOT protocol cannot gain S further information. For the scrambled circuit, a weak-honest S can do two things that deviate from the protocol: (1) constructing a circuit that computes f' other than f, and (2) scrambling the circuit without following the protocol description. Because S committed two keys of the output wire (k_t^0 and k_t^1) to C in step 2, and S is weak-honest; S can always get one key $k \in \{k_t^0, k_t^1\}$ from C in step 5. Key k is all S learns from C besides information S learns from the CIOT protocol.

Let $\mathcal{F}$ be the family of functions that each $f \in \mathcal{F} : ([0..2^\ell - 1])^n \to \{0, 1\}$. Then S learns just one bit information about C's input, i.e., S learns $f'(x_1, \ldots, x_n)$ for some function $f' \in \mathcal{F}$. Therefore, for any $x_1, \ldots, x_n \in [0, 2^\ell - 1]$ and $f \in \mathcal{F}$, the views of parties (C, S^*) in the real model can be simulated by the view of parties (C, S) in ideal model using $x_1, \ldots, x_n$ and possibly $f' \in \mathcal{F}$. ∎

Theorem 7 *Let $g : ([0..2^\ell - 1])^n \to \{0, 1, 2\}$ be a functionality, a malicious server can learn $g(x_1, \ldots, x_n)$ for some g, but no more than that.*

Proof Sketch. The analysis is similar to the proof of Theorem 7. For the scrambled circuit, a malicious S can (1) construct a circuit that computes f' other than f, or (2) scramble the circuit arbitrarily without following the protocol description. Because S committed two keys of the output wire (k_t^0 and k_t^1) to C in step 2, there is three possibilities S can conceive from the CIPPE protocol. First, S learns k_t^0. Second, S learns k_t^1. Third, S constructed a "bad" scrambled circuit intentionally, and C fails in evaluating the circuit or fails in decrypting η_0 and η_1. In any case, S learns no more than 2 bit information about C's input.

More precisely, let $\mathcal{G}$ be the family of functions that each $g \in \mathcal{G} : ([0..2^\ell - 1])^n \to \{0, 1, 2\}$. Then S learns $g(x_1, \ldots, x_n)$ for some function $g \in \mathcal{G}$ instead of $f(x_1, \ldots, x_n)$ where $f \in \mathcal{F}$. Therefore, for any $x_1, \ldots, x_n \in [0, 2^\ell - 1]$ and $f \in \mathcal{F}$, the views of parties (C, S^*) in the real model can be simulated by the view of parties (C, S) in ideal model using $x_1, \ldots, x_n$ and possibly $g \in \mathcal{G}$. ∎