On Protection in Federated Social Computing Systems

Ebrahim Tarameshloo Philip W. L. Fong Payman Mohassel
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{etarames, pwlfong, pmohasse}@ucalgary.ca

ABSTRACT

Nowadays, a user may belong to multiple social computing systems (SCSs) in order to benefit from a variety of services that each SCS may provide. To facilitate the sharing of contents across the system boundary, some SCSs provide a mechanism by which a user may "connect" his accounts on two SCSs. The effect is that contents from one SCS can now be shared to another SCS. Although such a connection feature delivers clear usability advantages for users, it also generates a host of privacy challenges. A notable challenge is that the access control policy of the SCS from which the content originates may not be honoured by the SCS to which the content migrates, because the latter fails to faithfully replicate the protection model of the former.

In this paper we formulate a protection model for a federation of SCSs that support content sharing via account connection. A core feature of the model is that sharable contents are protected by access control policies that transcend system boundary — they are enforced even after contents are migrated from one SCS to another. To ensure faithful interpretation of access control policies, their evaluation involves querying the protection states of various SCSs, using Secure Multiparty Computation (SMC). An important contribution of this work is that we carefully formulate the conditions under which policy evaluation using SMC does not lead to the leakage of information about the protection states of the SCSs. We also study the computational problem of statically checking if an access control policy can be evaluated without information leakage. Lastly, we identify useful policy idioms.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access Controls

General Terms

Security

Keywords

Protection Model, Federated Social Computing Systems, Account Connection, Secure Content Sharing, Secure Multiparty Computation, Safe Function Evaluation, Composite Policy, Policy Language

1. INTRODUCTION

Although a social computing system (SCS) provides a wide variety of services for its members, users may prefer to be members of different SCSs to benefit from unique services of these SCSs, or socialize differently on different SCSs. For instance, a user could be a member of Facebook, Google+ or Path to be connected with her friends or family, and share pictures and postings in order to socialize. In Yelp or Foursquare, members locate local deals and restaurants, and share their experiences with friends. Banjo or Sonar users generally try to explore new people and events nearby, and share their feeds and photos with them. To benefit from each of these services, a user usually ends up with accounts on multiple SCSs [19, 26].

To facilitate the sharing of contents across the system boundary, some SCSs provide a mechanism by which a user may "connect" her accounts on two SCSs [20, 30]. The effect is that contents from one SCS can now be shared to another SCS. For instance, by connecting Banjo to Foursquare, a user can read her friends' Foursquare feeds in Banjo, and can post her current location to Foursquare from within Banjo.

Although such a connection feature delivers clear usability advantages for users, it also generates a host of privacy challenges. A notable challenge is that the access control policy of the SCS from which the content originates may not be honoured by the SCS to which the content migrates, because the latter fails to faithfully replicate the access control policy and/or the protection model of the former. As a case in point, a member may declare her location in Foursquare, and share it on her Facebook account through Foursquare's connection service. Facebook treats her declared location information as its own resource so the user's Facebook policy (instead of her Foursquare policy) will be applied on the shared content. On closer examination, three protection challenges make this research problem nontrivial.

1. ***Policy Fidelity***. The access control policy of a shared content prior to migration is not communicated to the destination site. As a result, the policy to be enforced on the destination site need not be consistent with the user's privacy expectation as expressed in the origin policy. This ambiguity in terms of what policy to be

used for protecting shared contents is a first protection challenge.

2. ***Mechanism Fidelity***. Even if the origin policy (e.g., "friends" in Facebook) is known to the destination site (e.g., Foursquare), the latter may not be able to enforce it. A reason is that the destination site has to emulate the authorization mechanism of the origin site in order to enforce the origin policy. One can do so by reverse engineering, but the protection model of a typical SCS (e.g., Facebook) is a moving target (consider how often Facebook "upgrades" its protection model), and thus tracking the protection model of the origin site with fidelity is a tremendous software engineering challenge.

3. ***State Fidelity***. Another reason for the destination site not to be able to enforce the origin policy is that the access control policies of an SCS usually depends on user information not available to the destination site. In a traditional information system, the protection state and the application state of the system do not intersect. A characteristic of SCSs is that user-contributed information is used for authorization. For example, friendship information or location claims (i.e., "check in") are used for authorization purposes. Consequently, to evaluate the "friends" policy of Facebook, one needs to know the social network of Facebook. If the destination site (e.g., Foursquare) attempts to "approximate" this proprietary knowledge (e.g., by consulting the Foursquare friend list of the user), then emulation fidelity is compromised.

To the best of our knowledge, the need of a privacy policy for shared data across social network systems were first identified by Ko *et al.* [20]. We are aware of two lines of previous work that have considered protection of shared data across distributed systems [19, 26, 28]. In their pioneering work [19, 26], Shehad *et al.* proposed the formulation of cross-site policies for shared contents, in order to address the challenge of Policy Fidelity. They advocated also the adoption of a trusted third party, *x-mngr*, for managing cross-site policies and arbitrating accesses. Their work, however, considers policies in the form of access control lists. Consequently, the considered policies do not reflect the kind of policies that are actually used in the context of social computing (e.g., colocation and friendship). The rationale is understandable: adoption of social computing policies would result in the challenge of State Fidelity that to date has no obvious solution.

A second line of related work is that of Squicciarini *et al.* [28]. This ambitious work deals with the general problem of protecting shared data in distributed systems, and thus its scope is not restricted to that of content sharing in social computing. They introduced the idea of a self-controlling object (SCO), which encapsulates the shared data object, the access control policy, mobile code that specifies the semantics of the policy (i.e., the procedure for evaluating the policy), as well as ways to synchronizing the policy to its latest version. This framework clearly addresses the challenges of Policy and Mechanism Fidelity. Yet, again, the type of policies considered in this work is not the kind typical in social computing. The protection states of the SCSs simply cannot be encapsulated in such a package. Apply their framework to protect shared contents in SCSs does not address the challenge of State Fidelity.

This paper proposes a protection model for shared contents in a confederation of SCSs, with the goal of supporting access control policies commonly found in social computing (e.g., friends, co-location, etc), while tackling the challenge of State Fidelity head on (as well as Policy and Mechanism Fidelity). Our contributions are the following:

1. A novel protection model for controlling access to shared contents in SCSs is proposed in §2. To ensure Policy Fidelity, we allow users to formulate explicit shared access policies for protecting shared accesses. Two key features of the policy language are: (a) Atomic policies come from the policy vocabulary of the SCSs in the confederation, thereby supporting social computing policies; (b) The atomic policies are interpreted according the authorization mechanisms of explicitly named SCSs. Atomic policies can be combined into composite policies. To ensure Mechanism Fidelity, such a composite policy is evaluated using distributed computing: each atomic policy is evaluated by a query to the SCS who knows how to interpret that query. Such SCS also has access to the protection state needed for evaluating the atomic policy, so State Fidelity is guaranteed.

2. Distributed evaluation of shared access policies may disclose part of the protection states (and thus user information) of SCSs to one another. Prevention of such information leak is a novel privacy goal that has not been studied in the context of SCS content sharing before. In §3, we describe how to employ a variant of Secure Multi-party Computation (SMC), known as Private Function Evaluation (PFE), for achieving such a privacy goal.

3. Motivated by the need of default policies to ease policy administration, we notice that the PFE solution may not be applicable in the presence of default policies. We therefore articulate another means to achieve the privacy goal using regular SMC. In §4, we formally characterize policies that do not leak information about protection states of the SCSs when it is evaluated using SCM, and do so via the classical notion of nondeducibility [29]. We also characterize the computational complexity of deciding if a policy leaks information (i.e., Π_2^p-complete). Lastly, we discuss policy idioms that are useful in practice.

2. FEDERATED SCSs

We present a protection model for shared resources among a confederation of SCSs, in which member SCSs collaborate to offer protection. We begin with detailing the assumptions we make of the member SCSs as they participate in the confederation (§2.1). We then outline the features offered by this protection model (§2.2). After that, the protection model itself will be described in details (§2.3, §2.4, §2.5).

2.1 Assumptions

Suppose a user has the user name u_1 on SCS i_1, and user name u_2 on SCS i_2. When she "connects" her account on i_2 to her account on i_1, she is essentially claiming that the identity u_1 on i_1 is equivalent to the identity u_2 on i_2. Typically, she does so by presenting to i_1 the user name u_2 as well as the corresponding password. Then, SCS i_1 attempts to use the credential to log on to SCS i_2, success of which will be taken as a proof that the claim of identity mapping is trustworthy. In short, account connection is a manual identity mapping process. We make the following assumption.

Assumption 1 (User Identity). *The manual identity mapping process is (a) consistent and (b) applied whenever needed.*

Technically, part (a) of the assumption requires that the establishment of the "is equivalent to" relation among identities of different SCSs result in an equivalence relation (i.e., reflexive, symmetric and transitive), such that no two identities within the same equivalence class come from the same SCS. This is not necessarily true with current technology. For example a user may claim to Google+ that his Facebook identity is "James White," while at the same time claiming to Foursquare that his Facebook identity is "Jim White." We assume that, collaboration within the confederation will regulate such inconsistencies (e.g., by identifying cases of account hijacking and duplicate identities). Part (b) of the assumption requires that, as our proposed model is adopted by a confederation of SCSs, the latter will faithfully impose manual identity mapping whenever our model demands it. In existing implementations, manual identity mapping is only imposed under two scenarios. First, when a user attempts to "push" information from SCS i_1 to SCS i_2, she does so by presenting to i_1 her credential in i_2. Second, when a user attempts to "pull" information from SCS i_1 to SCS i_2, she will do so by presenting to i_2 her credential in i_1. In short, manual identity mapping is performed when resources are shared. In our protection model, we also require manual identity mapping when a user accesses a resource that has been shared to an SCS. That is, manual identity mapping predicates access to shared resources. Part (b) assumes this will be enforced by member SCSs of the confederation.

A major corollary of Assumption 1 is that users across all SCSs are uniquely identifiable. From now on, we will bypass the differentiation of user identities in different SCSs. This does not mean that the confederation needs to have a universal identifier for each user (e.g., a single sign-on mechanism). Each SCS will continue to feature its own identity management system, and identity mapping is still performed manually, on demand, via account connection. All that we assume is that the equivalence classes of user identities can uniquely identify users within the confederation.

Next, we assume that when a resource created in SCS i_1 by a user u is shared to SCS i_2, the destination SCS will retain all information regarding the origin of the resource, its identity in the originating SCS, as well as which user created that resource.

Assumption 2 (Resource Identity). *Every shared resource retains its identity, its originating SCS as well as ownership. Other auxiliary information regarding a shared resource that is needed for protection will also be retained after sharing*

Consequently, we can uniquely identify a shared resource across all SCSs in the confederation.

The third assumption is concerned with the Policy Decision Points (PDPs) of the federated SCSs.

Assumption 3 (Authorization Service). *The member SCSs of the confederation open up their PDPs (or part of them) as services that other member SCSs can query.*

While current implementations do not yet support this assumption, we believe that the assumption is in line with current trend, in which the authentication mechanisms of popular SCSs are turned into services that other web applications may reuse (e.g., logging in via Facebook or Google+). This is an architectural price to be paid for participating in the confederation. Privacy enhancing technologies are devised in §3 and §4 to ensure that, by turning its PDP into a queriable service, an SCS will not compromise privacy.

Lastly, we assume that the members of the confederation are cooperative in implementing the protection model.

Assumption 4 (Curious Member). *The member SCSs of the confederation are curious but not malicious.*

We are aware that SCSs are curious in that they may gather information about other SCSs as they participate in the protection model. As we shall make explicit in §3, our privacy goal is to ensure that, by opening the authorization procedure as a queriable service, an SCS does not disclose information about its protection state to other SCSs.

2.2 Feature Overview

Our proposed protection model for federated SCSs offers four key features: (1) protection of shared resources, (2) shared access policies, (3) distributed evaluation of situated queries, and (4) policy composition.

2.2.1 Protection of Shared Resources

Suppose u and v are both users of an SCS i_1 (e.g., Facebook). Suppose u is the owner of a resource r (e.g., a photo) that she created on i_1. When v attempts to access r within i_1, this is called a ***native access***. Such an access is mediated by the protection model of i_1, and thus it is *not* the focus of this work.

Now suppose u shares r to a different SCS i_2 (e.g., Banjo), on which she also has an account (Assumption 1). Suppose v also has an account on i_2. When v attempts to access r from within i_2, this is called a ***shared access***. Such an access is the focus of this work. In current implementations, there is no guarantee that a shared access is properly mediated. Even if the developer of i_2 attempts to emulate the protection model of i_1, limitations of reverse engineering may affect the fidelity of emulation. The goal of this work is to articulate a rational protection model for shared accesses.

2.2.2 Shared Access Policies

In the proposed model, u, who is the owner of the shared resource r, may specify a ***shared access policy*** for controlling shared accesses to r (but not native accesses). This shared access policy is assumed to be known within the confederation of SCSs (Assumption 2), so that wherever shared access is requested, the SCS to which r is shared will honor this policy (Assumption 4). This feature addresses the need for Policy Fidelity.

2.2.3 Distributed Evaluation of Situated Queries

The shared access policy may take the form of ***situated queries***. For example, u may demand that r be accessible only to her friends on Facebook, no matter where r has been shared. The shared access policy for r will then be "friends@*Facebook*," meaning that u and the requestor shall satisfy the query "friends" at the SCS *Facebook*.

In our running example, when v requests a shared access to r on i_2, the latter will not attempt to emulate friendship testing. Instead, it will query the authorization service of *Facebook* to ascertain if u and v are friends of one another (Assumption 3). Distributed evaluation ensures Mechanism and State Fidelity.

2.2.4 Policy Composition

To make the protection model more flexible, users may formulate composite policies made up of boolean combinations of situated queries. For example, consider the composite policy below:

$$\mathsf{friends}@\mathit{Facebook} \vee \mathsf{friends}@\mathit{Google+}$$

The policy grants shared access to either friends in Facebook or friends in Google+. Note that a requestor may be known to be a friend of the owner on Google+ but not on Facebook. Essentially, the query term friends has different semantics on different SCSs.

2.3 Schema, Configuration and State

Our formal model of federated SCSs is composed of three layers — schema, configuration and state. The schema of a confederation (§2.3.1) specifies the basic entities that exist in the confederation. Components of the schema remain constant unless the membership of the confederation changes (e.g., a new SCS joining the confederation) or when the protection model of a member SCS evolves. A privacy configuration (§2.3.2) of the confederation specifies the current privacy settings of the users. Components of a configuration may change as a result of administrative actions, such as the introduction of new users, creation of new resources, or the change of privacy settings. A protection state (§2.3.3) of the confederation tracks the current protection states of member SCSs within the confederation, as well as the whereabout of shared resources. Change of protection state occurs during the normal operation of the confederation. Configuration and state transitions are not modelled in this work.

2.3.1 Confederation Schema

The ***schema*** of a confederation is a 6-tuple $\mathcal{F} = \langle \mathit{Id}, \{\Sigma_i\}_{i\in \mathit{Id}}, \mathcal{U}, \mathcal{R}, Q, \mathcal{I}\rangle$, where:

- Id is a finite set of ***system identifiers***. Each identifier uniquely identifies an SCS in the confederation. A typical member of Id is denoted by i.
- $\{\Sigma_i\}_{i\in \mathit{Id}}$ is a family of countable sets, indexed by system identifiers. Each Σ_i is the set of all possible ***protection states*** for the SCS i. The sets are assumed to be pairwise disjoint (i.e., they don't intersect with one another). We write Σ for $\bigcup_{i\in \mathit{Id}} \Sigma_i$. A typical member of Σ is denoted by σ.
- $\mathcal{U}$ is a countable set of ***user identifiers***. By Assumption 1, we do not differentiate the different user names of the same user in different SCSs. We are not suggesting that actual implementation of this model needs to have a centralized identity management infrastructure. The global user identifiers are but a mathematical abstraction of users within the model.
- $\mathcal{R}$ is a countable set of ***resource identifiers***. Again, this is but a mathematical abstraction of resources within the confederation (Assumption 2).
- Q is the universe of ***atomic queries*** (or simply ***queries***). For instance, friends, friends-of-friends, and acquaintances are queries known to Facebook, and friends, co-located, in, nearby are queries known to Foursquare. Because the same query may be known across multiple systems (e.g., friends), we postulate that there is a common universe of queries. A typical member of Q is denoted by q.
- The ***interpretation function*** $\mathcal{I} : \Sigma \times \mathcal{U} \times \mathcal{U} \times Q \rightarrow \mathit{Bool}$ defines the semantics of atomic queries. Suppose $\sigma \in \Sigma_i$. That $\mathcal{I}(\sigma, u, v, q) = 1$ signifies that query q is satisfied by owner u and requestor v in protection state σ of system i. Otherwise, q is not satisfied by u and v when i is in state σ, or q is not supported by system i, or one of u and v does not have a corresponding identity in system i, or q is not applicable when system i is in state σ.

2.3.2 Privacy Configuration

A confederation can be configured with different privacy settings over its life cycle. A ***privacy configuration*** (or simply a ***configuration***) is an abstraction of such settings. Intuitively, a configuration specifies (a) the access control policies of user resources and (b) static properties of a resource such as the SCS in which the resource is created (Assumption 2). Formally, given a schema $\mathcal{F} = \langle \mathit{Id}, \{\Sigma_i\}_{i\in \mathit{Id}}, \mathcal{U}, \mathcal{R}, Q, \mathcal{I}\rangle$, a configuration is a tuple $\mathcal{C} = \langle U, R, \mathit{org}, \mathit{own}, \mathit{pol}\rangle$, in which:

- $U \subseteq \mathcal{U}$ is a finite set of ***active users***.
- $R \subseteq \mathcal{R}$ is a finite set of ***active resources*** that is currently being protected by the systems.
- Every resource is assumed to have been created in exactly one system. The function $\mathit{org} : R \rightarrow \mathit{Id}$ identifies the ***origin*** of each resource.
- The function $\mathit{own} : R \rightarrow U$ identifies the ***owner*** of every resource in the confederation.
- The function $\mathit{pol} : R \rightarrow \mathcal{PO}(Q, \mathit{Id})$ assigns a shared access policy to each resource. Note that this policy is not used for protecting a resource against native accesses. This policy controls shared accesses after a resource is shared to another SCS. Lastly, $\mathcal{PO}(Q, \mathit{Id})$ is the set of all policies. The syntax and semantics of policies are described in §2.4.

2.3.3 Protection State

The ***protection state*** (or simply ***state***) of a confederation is a pair $\mathcal{S} = \langle \mathit{sta}, \mathit{cur}\rangle$, such that:

- The function $\mathit{sta} : \mathit{Id} \rightarrow \Sigma$ assigns a protection state to each system identifier such that $\mathit{sta}(i) \in \Sigma_i$.
- The binary relation $\mathit{cur} \subseteq R \times \mathit{Id}$ records the SCSs to which each resource has been shared. A resource may have been shared to multiple systems at one time. An additional requirement is that $(r, \mathit{org}(r)) \notin \mathit{cur}$. That is, a resource is never "shared" to its origin.

2.4 Policy Language

We present a policy language (i.e., the set $\mathcal{PO}(Q, Id)$ in §2.3.2) for expressing shared access policies in federated SCSs. The most distinctive feature of the policy language is that one can specify policies as atomic queries to be interpreted at specific SCSs, thereby ensuring that their SCS-specific semantics are honoured. Such atomic queries can then be composed into composite policies via boolean connectives. To simplify discussion (especially in §4), we use propositional logic for policy composition. We could have adopted, for example, the Belnap Logic [5] as our policy composition framework, but such an extension is trivial.

2.4.1 Syntax

A policy is a propositional formula with the following abstract syntax:

$$\phi, \psi ::= \top \mid q@t \mid \phi \vee \phi \mid \neg\phi$$
$$t ::= i \mid \mathsf{org} \mid \mathsf{cur}$$

where $q \in Q$ and $i \in Id$. A ***situated query*** $q@t$ is an atomic query q that will be evaluated at the SCS represented by the ***location tag*** t (Assumption 3). A location tag t can be either ***absolute*** (i.e., a system identifier i) or ***relative*** (i.e., org or cur). The relative location tag org means that the tagged query is to be evaluated at the originating SCS of the resource that the policy is protecting. The relative location tag cur means that the tagged query is to be evaluated at the SCS in which the resource is being accessed. We write $\mathcal{PO}(Q, Id)$ for the set of all policies.

Standard derived forms can be defined as follows:

$$\bot = \neg\top \qquad \phi \wedge \psi = \neg(\neg\phi \vee \neg\psi)$$

2.4.2 Semantics

The semantics of the policy language is defined with respect to a schema $\mathcal{F} = \langle Id, \{\Sigma_i\}_{i \in Id}, \mathcal{U}, \mathcal{R}, Q, \mathcal{I}\rangle$, a configuration $\mathcal{C} = \langle U, R, org, own, pol\rangle$, and a $\mathcal{S} = \langle sta, cur\rangle$. In particular, the semantics is defined via a preprocessing function and a satisfaction relation.

First, the preprocessing function translates all occurrences of relative location tags to absolute ones. Suppose the system identifiers i_o and i_c are the SCSs to which org and cur respectively represent, and ϕ is a policy formula. Then $\mathsf{preproc}_{i_o,i_c}(\phi)$ is obtained from ϕ by replacing occurrences of org and cur by i_o and i_c respectively. The preprocessed formula contains only absolute location tags.

Second, the following satisfaction relation specifies the meaning of a formula ϕ with only absolute location tags:

$$\mathcal{I}, sta, u, v \models \phi$$

in which $u, v \in U$ are the owner and requestor respectively. The satisfaction relation is defined inductively as follows:

- $\mathcal{I}, sta, u, v \models \top$ always hold.
- $\mathcal{I}, sta, u, v \models q@i$ iff $\mathcal{I}(sta(i), u, v, q) = 1$.
- $\mathcal{I}, sta, u, v \models \neg\phi$ iff $\mathcal{I}, sta, u, v \not\models \phi$.
- $\mathcal{I}, sta, u, v \models \phi \vee \psi$ iff either $\mathcal{I}, sta, u, v \models \phi$ or $\mathcal{I}, sta, u, v \models \psi$.

2.5 Access Requests and their Authorization

A user v working in an SCS with system identifier i may request to access a resource r that has been shared to i. Such a request is characterized by the triple (v, r, i), and is authorized when both of the following two tests succeed:

$$(r, i) \in cur \tag{1}$$
$$\mathcal{I}, sta, u, v \models \phi \tag{2}$$

where $u = own(r)$, $\phi = \mathsf{preproc}_{i_o,i}(\psi)$, $i_o = org(r)$, and $\psi = pol(r)$. Test (1) is simply a sanity check, while test (2) evaluates the preprocessed policy against the owner of the resource.

3. PRIVACY VIA PFE

According to Assumption 3, each SCS opens up part of its authorization mechanism as a queriable service in order to support the distributed evaluation of shared access policies. This results in the disclosure of their protection states and thus user information. As a case in point, since interpersonal relationships are part of the Facebook protection state and user location claims are part of the Foursquare protection state, evaluating the situated queries $\mathsf{friends}@\textit{Facebook}$ and $\mathsf{nearby}@\textit{Foursquare}$ leads to disclosure of user information. If no further measures are imposed, then this could breach user privacy or compromise the SCSs' competitive advantages (i.e., competitors harvesting your business data).

Preserving the privacy of the SCSs' protection states during the evaluation of shared access policies is therefore an important privacy goal. To this end, we adopt ***secure multi-party computation (SMC)*** to carry out policy evaluation in a secure manner. In particular, this section focuses on the application of a variant of SMC known as ***private function evaluation (PFE)***.

In this section, we first review basic backgrounds on SMC and PFE (§3.1). We then survey a number of architectures in which the distributed evaluation of shared access policies is conducted in a secure manner with the help of PFE (§3.2).

3.1 Secure Multiparty Computation

Secure Multiparty Computation (SMC) allows a group of parties $P_1, \ldots, P_n$ each with their own private inputs $x_1, \ldots, x_n$ to collectively compute a function f of their inputs without revealing any additional information. SMC is a fundamental problem in cryptography and distributed computing and has been studied for over thirty years with seminal results on its feasibility in various settings [31, 15, 7, 3]. Recent work on *practical* SMC has even led to the design and implementation of several SFE/MPC frameworks [22], VIFF [10], Sharemind [4], Tasty [16], and many more.

The two most common guarantees provided by SMC protocols are *privacy* and *correctness*. Loosely speaking, privacy means that no "admissible" collusion of dishonest parties are able to learn any information about the honest parties' inputs besides what is derived from their own input and output. Correctness means that a similar collusion of dishonest parties, cannot trick the honest parties to learn and/or reveal an incorrect function f' of their input. Additional guarantees not discussed here but provided by some SMC protocols include output delivery, input independence, and fairness. Dishonest parties are allowed to behave in two different ways, i.e., semi-honest or malicious. Semi-honest parties follow the steps of the protocol but try to learn more information by looking at the transcripts. Malicious parties, on the other hand, are not restricted in any way and can behave arbitrarily. In this paper we only need SMC with security against semi-honest adversaries (Assumption 4).

The above security guarantees for an SMC protocol π_f computing a function f are usually formalized by comparing the actual execution of π_f in the real world (the real execution), with an ideal execution wherein each party sends his input to a trusted third party (TTP) through a secure channel, and receives an honestly computed output in return. One then declares π_f "secure" if for any adversary corrupting a subset of parties in the real execution, there exists a simulator corrupting the same parties in the ideal execution where the final output of the real-execution adversary and the ideal-execution simulator are indistinguishable. Given that the extent of the simulator's cheating is very limited in the ideal-execution, security of π_f implies that the real-world adversary cannot do much better either. We refer the reader to [14], for a more formal presentation of the above intuition. An important benefit of proving protocols secure in this fashion is that once we prove π_f secure, we can treat it as an ideal functionality, and when describing larger systems that use π_f as a building block, replace π_f with this ideal functionality that simply receives each parties' input and returns to him the portion of the output that belongs to him.

3.1.1 Private Function Evaluation

In the above discussion, we assumed that the function f is publicly known to all the participants in the SMC. But this is not always the case in practice. The function may be proprietary, or may otherwise contain private and sensitive information. Private Function Evaluation (PFE) is a variant of SMC that tries to address this problem. In a PFE protocol, unlike SMC, we assume that f is the private input of one of the parties (say P_1), while each party P_i also holds a private input x_i of its own. The protocol then computes P_1's private function on parties input in a secure manner. The security definitions for PFE are essentially identical to SMC, as PFE can be seen as an SMC protocol where the description of f is part of P_1's private input. The description of the ideal functionality can also be adjust to take as P_1's input not only x_1 but also the function f.

Several implementations of PFE protocols have been considered in the literature [21, 18, 25]. We also note while it is usually assumed that f is known by a single party in PFE, many of the existing constructions can be naturally extended to consider the case where f is secretly shared between a subset or all the parties but no individual party knows the function itself.

3.1.2 Safe Functions for SMC

In SMC protocols, traditionally, we assume that parties have agreed on a function f and are willing to reveal to others anything that can be derived from its output. In other words, SMC does not try to determine which functions are "safe" and delegates this to the participants of the SMC. To see why some functions are not "safe," and knowledge of its output allows one to infer its inputs (even if SMC protocols are used for function evaluation), consider the following boolean function: $f(x, y) = x \wedge y$. If one learns that the output of f is 0, then one can infer that x and y are both 0.

3.2 Three PFE-based Architectures

When a shared access to resource r is requested in an SCS i, i will need to evaluate policy $\phi = \mathit{pol}(r)$ to reach an authorization decision. To do so, it requires the truth values of the situated queries that appear in ϕ. In the interest of privacy, the SCSs that are referenced in the location tags of the situated queries cannot simply send those truth values to i. Instead, the evaluation of ϕ will be conducted using SMC, and the output is made available to i.

A potential hole in this scheme is that ϕ may not be a safe function (§3.1.2). By reading the output of ϕ, a curious i may still infer information about the truth values of some of the situated queries that appear in ϕ.

In this work we employ two approaches to address this issue. A first approach is to hide $\mathit{pol}(r)$ from the SCSs, so that they do not know what function the SMC protocol is computing. In that way, simply learning the output of an unknown function does not allow i to infer the truth values of the situated queries. This requires the deployment of PFE protocols (§3.1.1).

The advantage of this approach is that there is almost no restriction on what ϕ can be. The core challenge is to ensure that the shared access policies are hidden from the SCSs involved. The assumption behind this approach is that SCSs are semi-honest (Assumption 4).

We will examine a number of architectures that are the natural starting points for realizing this approach. Our goal is not to advocate PFE as the optimal solution, but to carefully examine the design forces at play in the problem, so as to motivate the second approach (formulation of safe functions) that we advocate in §4.

In the following, we will first describe the three architectures without passing value judgement (§3.2.1, §3.2.2, §3.2.3), and then assess them in §3.2.4. We conclude this section by pointing out the fundamental limitation of achieving our privacy goal via PFE (§3.2.5), thereby paving the way to the next section.

3.2.1 Origin Tracks Policy (ORIGIN)

In the ORIGIN architecture, each SCS keeps track of the shared access policies for resources that originate from that SCS: i.e., SCS i stores $\mathit{pol}(r)$ whenever $\mathit{org}(r) = i$.

When a user v requests to access a shared resource r on SCS i_c, site i_c will contact the originating SCS $i_o = \mathit{org}(r)$ to initiate the computation of authorization decision. The originating SCS i_o will then look up the shared access policy $\phi = \mathit{pol}(r)$, and compute $\psi = \mathsf{preproc}_{i_o,i_c}(\phi)$ (i.e., replacing relative location tags by absolute ones). Based on the policy formula ψ, i_o identifies the parties that should participate in PFE, and then initiate π_ψ. The participants include i_o (who contributes the formula ψ), the SCSs who appear in the situated queries in ψ (if the situated query $q@i$ appears in ψ, then SCS i will contribute the input that corresponds to the truth value of $q@i$), and i_c (who reads the output of ϕ). The output of the PFE protocol will be visible only to i_c, the SCS from which the request was first raised.

3.2.2 User Tracks Policy (USER)

In the USER architecture, each user stores the shared access policies of the resources that she owns, on a user-owned storage. For example, a user can save her shared access policies on a network attached storage, or in her cloud storage account. Whenever an access to a shared resource is requested, the storage site will have to participate in the PFE protocol as the party to contribute the function as an input to the protocol. The rest is similar to the Origin architecture. The architectural price is that the storage service that tracks the user's policies must now need to understand the confederation architecture as well as the PFE protocol.

3.2.3 *Third Party Tracks All Policies (*TP*)*

In the TP architecture, rather than having a vast number of storage services involved, as in the case of the User architecture, we postulate the existence of a centralized policy storage service, who is an honest third party denoted by $\hat{T}$. We assume that the SCSs may collude with each other except with $\hat{T}$.

The centralized policy storage service will be contacted when a shared access is to be authorized, and it will be the party to contribute policy formulas as inputs to the PFE protocol. The policies are hidden from all SCSs except for $\hat{T}$. The architectural price is much lower in this case: only one additional party other than the SCSs need to understand the confederation architecture and the PFE protocol.

3.2.4 *Assessment*

We analyze the pros and cons of the three architectures.

Privacy.

Can the three architectures achieve the core privacy goal by keeping the policies hidden from the SCSs in the confederation?

Consider the Origin architecture. Suppose the policy contains the relative location tag org, or an absolute location tag that is identical to $org(r)$, then the originating SCS will be contributing both the function to be evaluated as well as one of the inputs to the function when it participates in the PFE protocol. As a result, the originating SCS may end up learning something about the protection states of the other SCSs who are named in the situated queries of the policy. One way to prevent this issue is to ban the use of the relative location tag org, or the mentioning of the originating SCS in situated queries. Such a restriction reduces the flexibility of the architecture.

The User architecture can effectively hide the policies from the SCSs in the confederation. Yet, practical considerations suggest that the SCSs are likely to be the cloud storage service providers adopted by the users. For example, Google+ belongs to the confederation, and a user stores his shared access policies in Google Drive. In such a case, it is difficult to convince the other SCSs that Google Drive (being aware of the confederation architecture) will not leak the policy to Google+, thereby compromising the privacy of the other confederation members.

The TP architecture meets the privacy goal by hiding the policies from the SCSs, so long as the assumptions in §3.2.3 are met.

Knowledge of Query Vocabulary.

In order to participate in the PFE protocol, the party who stores the policy and contributes the function to be evaluated must understand the query vocabulary of all the SCSs in the confederation. In the Origin architecture, this means that *every* SCS in the confederation must understand the full query vocabulary of all the other confederation members. This is a rather high price to pay. The same objection can be said of the User architecture, only that the situation is even worse: the storage services outside of the confederation are involved. In comparison, only one party ($\hat{T}$) needs to have such knowledge in the TP architecture.

Fault Tolerance.

With a centralized storage service, TP presents a single point of failure. If $\hat{T}$ is not available, the entire confederation is rendered dysfunctional. With the Origin architecture, the failing of one SCS in the confederation affects the policy lookup of all resources originating from that SCS. This is actually not bad as native accesses would not be possible either if the originating SCS is down. The User architecture appears to be the most fault tolerant: the failing of a policy storage provider will affect only the shared resources of the users who use that provider for storing policies. The situation will be worsen if there is an oligopoly in the cloud storage service industry.

Final Assessment.

Balancing various considerations, TP appears to be a viable foundational architecture on which future work can be built. Origin can be considered a compromise if one is willing to live with syntactic restrictions on the policy language. User does not seem to be viable as storage services outside of the confederation are required to participate in PFE.

3.2.5 *The Challenge of Policy Administration*

Implicit in the presentation of the protection model and the three architectures above is the assumption that every user must specify a shared access policy for every resource that she might want to share to other SCSs in the future. It is unlikely that a user will bother to go through such a tedious exercise. An obvious way to alleviate this problem is to allow the specification of ***default policies*** for various categories of resources. For example, a user can specify friends@org $\vee$ co-located@cur as the default shared access policy for *all* her photo albums across all the SCSs in the confederation. In fact, the provision of relative location tags has exactly this application in mind.

With default policies, keeping policies hidden from the SCSs becomes an unrealistic assumption. As default policies are shared by multiple resources, and different users tend to end up adopting similar default policies, curious SCSs can make educated guesses of what policies are involved in the PFE protocol[1]. Consequently, adoption of PFE for policy evaluation is not sufficient for guaranteeing privacy.

4. PRIVACY VIA SAFE FUNCTIONS

Rather than hiding the shared access policies from the SCSs in the confederation, in this section we consider a solution in which all shared access policies are allowed to be publicly known, thereby accommodating the use of default policies. In particular, we ensure that the policies are "safe," in the sense that even if a party obtains the output of the SMC protocol, it will not be able to deduce any information regarding the inputs to the function.

This approach employs regular SMC rather than PFE. At the centre of the approach is the static analysis of the shared access policies to detect if they are safe. When a

[1]That Facebook takes "friends-of-friends" as the default policy is a widely discussed and well documented issue.

shared access policy is detected to be unsafe by an SCS who needs to supply the truth values of situated queries, that SCS can refrain from participating in the SMC protocol, thereby preserving its privacy.

We begin our discussion by fixing some notations (§4.1). We then offer offer a novel formalization of safety (§4.2) and its application to the distributed evaluation of shared access policies (§4.3). We then study the time complexity of the static analysis for determining if a give policy is safe (§4.4). Lastly, we will look at useful policy idioms (§4.5).

4.1 Preliminaries

Let $f : A \to B$ be a function. We write $ran(f)$ for the range of f: i.e., the set $\{y \in B \mid \exists x \in A \,.\, f(x) = y\}$. We write $pre_f(y)$ for the preimage of f at y, which is the set $\{x \in A \mid f(x) = y\}$.

In the following, we use the term boolean functions to refer to functions with type signature $\mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. We are not particularly concerned with the representation of the functions (i.e., whether they are represented as propositional formulas, circuits, or binary decision diagrams, etc). We simply assume that the evaluation of the function can be performed in time polynomial to the size of the input and the size of the representation of the function.

Given a bit vector $\vec{u} \in \mathbb{B}^n$, and a set $I = \{i_1, i_2, \ldots, i_k\}$ such that $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, we write $\mathsf{proj}_I(\vec{u})$ to denote the bit vector $u_{i_1} u_{i_2} \cdots u_{i_k}$. We also write proj_i as a shorthand for $\mathsf{proj}_{\{i\}}$.

Given two bit vectors $\vec{u}, \vec{v} \in \mathbb{B}^n$, we write $\vec{u} =_I \vec{v}$ whenever $\mathsf{proj}_I(\vec{u}) = \mathsf{proj}_I(\vec{v})$.

4.2 Input Nondeducibility

Our formal definition of "safe" functions is based on Sutherland's classical definition of information flow via the notion of deducibility [29, §2], which we reproduce here.

Definition 5 (Information Flow). *Given a set of possible worlds W and two functions f_1 and f_2 with domain W, we say that* information flows from f_1 to f_2 *if and only if there exists some possible world w and some element z in the range of f_2 such that z is achieved by f_2 in some possible world but in every possible world w' such that $f_1(w') = f_1(w)$, $f_2(w')$ is not equal to z.*

The intuition behind this definition is the following. Suppose the world is in the state w. There is an observer O_2 who does not know that the world is in state w. Yet O_2 observes the world state indirectly through an information function f_2. His observation is $z = f_2(w)$. So O_2 is aware that the actual world state must be among the set $pre_{f_2}(z)$. In the meantime, O_2 is also aware of the existence of another observer O_1, as well as the fact that O_1 observes the world state through the information function f_1. Now O_2 wants to infer something about the observation of O_1. O_2 knows the following: (a) y is a potential observation of O_1 (i.e., $y \in ran(f_1)$); (b) for every possible world state $w' \in pre_{f_1}(y)$ that produces observation y, $f_2(w') \neq z$ (in other words, $pre_{f_1}(y) \cap pre_{f_2}(z) = \emptyset$). With this, O_2 can safely conclude that $f_1(w)$ cannot be y, and thus O_1 is not observing y, even though O_1 does not know that w is the current world state. Since such a deduction is possible, we say that information flows from f_1 to f_2.

Sutherland observed that information flow in the above sense is symmetric [29, §2]: i.e., information flows from f_1 to f_2 if and only if it flows from f_2 to f_1. So we only need to say that there is information flow between f_1 and f_2.

In our application, we are particularly concerned with the inference of input values from output values.

Example 6. *Suppose $f(x, y) = \neg x \vee y$. If we know that $f(x, y) = 0$, then we can infer that $x = 1$. However, if we know that $f(x, y) = 1$, then we cannot infer any information about x, for both $f(1, 1) = 1$ and $f(0, 1) = 1$.*

This notion of inference can be captured by the definition of Sutherland, by observing that the value of an input is simply the output of a projection function. Suppose we are dealing with a boolean function $f : \mathbb{B}^n \to \mathbb{B}$. The set of possible worlds here is $\mathbb{B}^n$. Let proj_i be the projection function for the i'th argument: that is, $\mathsf{proj}_i(x_1, \ldots, x_n) = x_i$. According to Sutherland's definition, one can deduce information about the i'th input from examining the output of f whenever information flows from proj_i to f.

Definition 7. *A function $f : \mathbb{B}^n \to \mathbb{B}$ is* input nondeducible *for a non-empty set I of size k, where $I \subseteq \{1, \ldots, n\}$, if and only if for every $b \in \mathbb{B}$, if there exists $\vec{w} \in \mathbb{B}^n$ for which $f(\vec{w}) = b$, then for every $\vec{v} \in \mathbb{B}^k$, there exists $\vec{u} \in \mathbb{B}^n$ such that $\mathsf{proj}_I(\vec{u}) = \vec{v}$ and $f(\vec{u}) = b$.*

If I is a singleton set $\{i\}$, then we simply say f is i'th input nondeducible.

In short, f is input nondeducible for I if and only if there is no information flow between proj_I and f, and f is i'th input nondeducible if and only if there is no information flow between proj_i and f.

4.3 Application and Generalization

To see how input nondeducibility captures the notion of safe function, consider this scenario. Suppose a shared access to resource r is requested in SCS i_c, and the shared access policy is $\phi = pol(r)$. Suppose further that ϕ is the boolean function $f(x_1, \ldots, x_n)$, where each input of f is the truth value of a situated query. Say the input x_j is the situated query $q@i$, for some $i \neq i_c$. Assume for the time being that i_c does not contribute any input to the SMC protocol π_f. In order for SCS i to be convinced that participating in π_f will not leak information about x_j to i_c, i will check that f is j'th input nondeducible. In fact, SCS i will repeat this check for every x_j that it contributes.

Now the above scenario has an assumption, that i_c, the SCS who obtains the authorization decision (the output of f), does not contribute any input to π_f. That is, ϕ does not contain situated queries that refers to i_c. We know that this may not be avoidable, especially when the relative location tag cur is involved. This situation requires special attention.

Definition 8. *Given a function $f : \mathbb{B}^n \to \mathbb{B}$, suppose $I, J \subseteq \{1, \ldots, n\}$ such that $I \neq \emptyset$ and $I \cap J = \emptyset$. Let $k = |I|$ and $m = |J|$. Function f is* input nondeducible for I despite J *if and only if for every $\vec{a} \in \mathbb{B}^m$ and $b \in \mathbb{B}$, if there exists $\vec{w} \in \mathbb{B}^n$ such that $\mathsf{proj}_J(\vec{w}) = \vec{a}$ and $f(\vec{w}) = b$, then for every $\vec{v} \in \mathbb{B}^k$, there exists $\vec{u} \in \mathbb{B}^n$ such that $\mathsf{proj}_I(\vec{u}) = \vec{v}$, $\mathsf{proj}_J(\vec{u}) = \vec{a}$, and $f(\vec{u}) = b$.*

If I is a singleton set $\{i\}$, then we simply say f is i'th input nondeducible despite J.

Note that Definition 7 is a special case of the definition above (when $J = \emptyset$). The set J is essentially the set of inputs contributed by the reader of the output (i_c). If a function f is i'th input nondeducible despite J, then even though the reader of the output knows the inputs in J, he still cannot infer the value of the i'th input. Therefore, in the general case, an SCS must ensure that f is i'th input nondeducible for J, for every input i that it contributes, with J being the set of inputs contributed by the SCS who reads the output. In summary, an SCS determines the safety of participating in an SMC protocol by deciding input nondeducibility.

4.4 Deciding Input Nondeducibility

We examine the computational complexity of deciding input nondeducibility. We begin with defining two corresponding decision problems.

Definition 9. IND_0 *is the set of all pairs* (f, I) *for which* f *is input nondeducible for* I. IND *is the set of all triples* (f, I, J) *for which* f *is input nondeducible for* I *despite* J.

The following result shows that IND_0 and IND are in the second level of the polynomial hierarchy [1]. We begin with an upper bound for the time complexity of IND (consult Appendix A for a proof).

Proposition 10. IND *is in* Π_2^p.

The following proposition gives a lower bound for the time complexity of IND_0 (see Appendix B for a proof).

Proposition 11. IND_0 *is* Π_2^p*-hard.*

Since there is a trivial polynomial-time reduction from IND_0 to IND, the following is a corollary of the above results.

Corollary 12. *Both* IND_0 *and* IND *are* Π_2^p*-complete.*

To implement the static analysis, one can encode the IND instance as a Quantified Boolean Formula (QBF), along the line of formula (3) in Appendix A, and then use an existing QBF solver to test the satisfiability of the QBF.[2] As (3) is a 2QBF (i.e., a QBF with two alternations of quantifiers), one can employ dedicated 2QBF solvers [17, 2] for better efficiency.[3] As we shall see in §4.5, safe policies are relatively small in practice, and thus present no performance challenge to these solvers.

4.5 Policy Idioms

To facilitate discussion, we adopt the following convention.

Convention 13. *Suppose the inputs of a function* $f(\vec{x})$ *is named by variables* $\vec{x}$ *(e.g.,* f *is represented by a propositional formula). When we assert that* f *is input nondeducible for* I*, the set* I *may be specified as a subset of* $\{\vec{x}\}$.

The following example demonstrates the rarity of input nondeducible functions.

Example 14. *Of the* $2^{2^2} = 16$ *boolean functions with two inputs, only XOR and its negation are* i*'th input nondeducible for* $i \in \{1, 2\}$*. That means the usual boolean combinations such as conjunction and disjunction do not guarantee input nondeducibility.*

[2]Examples of QBF solvers are Quantor, NanoFlex, DepQBD and Bloqqer by Armin Biere and Florian Lonsing, and Cirqit by Alexandra Goultiaeva and Fahiem Bacchus.

[3]Efficient implementations of 2QBF solvers include Mini2QBF and Free2QBF by Sam Bayless (`www.sambayless.ca`).

While XOR appears not to be particularly useful for composing policies, there are indeed boolean functions that are both useful and input nondeducible, so long as we are willing to consider boolean functions of higher arities.

Example 15 (Threshold Function). *Define the* threshold function $\tau_{m/n}(x_1, \ldots, x_n)$ *such that* 1 *is returned if at least* m *of the inputs are* 1*, and* 0 *is returned otherwise. Note that* $\tau_{m/n}$ *is* i*'th input nondeducible for each of the inputs* x_i *if* $1 < m < n$.

Observe that boolean disjunction and conjunction are respectively $\tau_{1/n}$ *and* $\tau_{n/n}$*. They do not satisfy the requirement of* $1 < m < n$*, and thus it is no surprise that they are not input nondeducible. The smallest* m *and* n *to satisfy* $1 < m < n$ *are* 2 *and* 3 *respectively.*

More generally, $\tau_{m/n}$ *is* i*'th input nondeducible despite* J*, for a set* J *of size* k*, if it can be guaranteed that* $1 + k < m < n - k$*. If* $k = 1$*, then the smallest* m *and* n *to satisfy the above requirement are respectively* 3 *and* 5*. That is, for* $\tau_{3/5}$*, if the adversary knows the output as well as one of the inputs, it still cannot infer any of the other* 4 *inputs.*

Example 16 (Conditional Function). *Consider the boolean function* if*:*

$$\mathsf{if}(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$$

Function if *is* i*'th input nondeducible for each* $i \in \{1, 2, 3\}$*. Intuitively, the input* x_1 *"confuses" the adversary who learns of the output, making it impossible for it to infer which of* x_2 *or* x_3 *that it is observing through the output.*

Note, however, function if *is not* i*'th input nondeducible despite* J *for* $J \neq \emptyset$*. If an adversary knows of the output and at least one other input, it may be able to infer the value of input* x_i*.*

A generalization of if *is the following function:*

$$\mathsf{switch}(x_1, \ldots, x_n; y_0, \ldots, y_{2^n})$$

Depending on which of the 2^n *values in* $\mathbb{B}^n$ *that the bit vector* $x_1 \cdots x_n$ *assumes, one of the* y_i*'s will be selected as output. The function* $\mathsf{if}(x_1, x_2, x_3)$ *is equivalent to* $\mathsf{switch}(x_1; x_3, x_2)$*.*

The input x_1 *in* if *and the inputs* $x_1, \ldots, x_n$ *in* switch *are called* **condition inputs***. The inputs* x_2 *and* x_3 *in* if *and the inputs* $y_1, \ldots, y_{2^n}$ *are called* **branch inputs***.*

When $|J| < n$*, function* switch *is* i*'th input nondeduciable despite* J*. If* J *is a singleton set, then the smallest* n *that meets the above precondition is* 2*.*

The two functions in Examples 15 and 16 offer us replacements for boolean disjunction and conjunction. Specifically, conjunction is used in policy formulation in strengthening the authorization condition, so that multiple criteria need to be met before access can be granted. The threshold function can be used in a similar way. So long as m of the n criteria are met, authorization will succeed. The adversary is "confused" by the surplus of n criteria, not know which of the m-subsets of the n criteria had been used for establishing access. On the other hand, disjunction is used in policy formulation by providing alternatives to justify access. The conditional function switch can be used as a replacement. The criteria are passed as y_is, and n additional x_is are passed as "confusion factors."

Input nondeducible functions have limited composability, as the following result shows.

PROPOSITION 17 (LIMITED COMPOSABILITY). *If a function is input nondeducible for I, then so is its negation. Suppose $\vec{x}$ and $\vec{y}$ are two disjoint lists of boolean variables, and $f(\vec{x})$ and $g(\vec{y})$ are input nondeducible for I_f and I_g respectively (where $I_f \subseteq \{\vec{x}\}$ and $I_g \subseteq \{\vec{y}\}$). Then $h_\vee(\vec{x},\vec{y}) = f(\vec{x}) \vee g(\vec{y})$ and $h_\wedge(\vec{x},\vec{y}) = f(\vec{x}) \wedge g(\vec{y})$ are input nondeducible for $I_f \cup I_g$.*

Considering the rarity of functions that are provably input nondeducible, and the limited composibility that input nondeducible functions have, it is unwise to leave it to the users to formulate their own shared access policies that can be safely evaluated by SMC. Instead the users are presented with templates of policies that are known to be "safe," and given guidance on how to instantiate the templates. This is an acceptable practice in the industry: even though there is a vast space of policies for Facebook-style Social Network Systems (FSNSs) [13, 12], Facebook offers a standard vocabulary of common policies for users to choose from (e.g., me-only, friends-only, friends-of-friends, everyone). In the following, we describe two safe policy templates that are based on threshold and conditional functions.

POLICY TEMPLATE 18 (THRESHOLD POLICIES). *This policy template guides a user into formulating a threshold policy ($\tau_{m/n}$). The user will be asked to specify five (5) situated queries, with the following restrictions: (a) each of* cur *and* org *can be referenced in at most one of the situated queries, and (b) each system identifier can be referenced as an absolute location tag in at most one situated query.*

After preprocessing, no system identifier appears more than twice (remember org *and* cur *must refer to two distinct system identifiers, as we are only concerned with shared accesses). If the identifier of the SCS to read the output appears twice, then the function to be evaluated by SMC will be $\tau_{2/3}$ with the remaining three (3) situated queries as arguments. This is safe (i.e., i'th input nondeducible) according to Example 15, as the three situated queries do not refer to the SCS to read the output. Otherwise, the identifier of the SCS to read the output appears no more than once, and the function to be evaluated will be $\tau_{3/5}$, with all the five (5) situated queries as arguments. According to Example 15, this is safe (i.e., i'th input nondeducible despite J, for $|J| \leq 1$).*

POLICY TEMPLATE 19 (CONDITIONAL POLICIES). *This template guides a user in formulating a conditional policy (*switch*). The user will be asked to specify four (4) situated queries called branch queries, plus two (2) situated queries called condition queries. The following requirements apply: (a) the system identifiers that appear as absolute location tags in the condition queries must be distinct, (b) the system identifiers that appear as absolute location tags in the branch queries must be distinct, and (c) each of* org *and* cur *must appear at most once, as a relative location tag in a branch query.*

After preprocessing, no system identifier appears twice in the branch queries (recall that org *and* cur *must refer to two distinct system identifiers, since we are only concerned with shared accesses). Denote by i_c the system identifier of the SCS to read the output of policy evaluation. If i_c appears at most once among all the situated queries, then the function to be evaluated will be* switch*, with the branch queries as branch inputs, and the condition queries as condition inputs. According to Example 16, this* switch *policy is i'th input nondeducible despite J, since $|J| = 1 < 2$.*

Otherwise, in the worst case, i_c appears in two branch queries and one condition query. That is, there are at least one condition query and at least two branch queries for which i_c does not appear as location tags. Call these "clean" queries. In that case, the function to be evaluated will be if*, with two clean branch queries as branch inputs, and one clean condition query as the condition input. (When there are more clean queries than is needed, break ties randomly.) According to Example 16, this* if *policy is i'th input nondeducible.*

5. RELATED WORK

In recent years, a great deal of attention has been invested on access control models for social computing applications [6, 13, 27, 8]. Although a protection model has been proposed in each of the above works, none of them deals with the protection of contents shared to other social computing systems.

The privacy problem of shared contents in SCSs was first identified by Ko *et al.* [20]. We are aware of two lines of previous work that address this problem. The first is the pioneering work of Ko *et al.* [19] and its follow-up work [26]. They proposed a cross-site interaction framework, *x-mngr*, which features cross-site policies (i.e., what we call shared access policy) for protecting shared contents. They proposed the use of a trusted third party for managing cross-site policies and arbitrating accesses. Another distinctive feature is their identity mapping feature, which is based on machine learning technologies.

We learned from their work the use of shared access policies, as well as the need of trusted third party for policy tracking (in the TP architecture). Our work differs from theirs in the following respects. First, the policies they considered are in the form of access control lists, and thus they do not represent typical policies found in social computing. Our policies are boolean combinations of atomic policies defined in the policy vocabulary of individual SCSs. This allows us to directly confront the issue of State Fidelity. Second, we see the account connection feature not only as a content migration mechanism, but also as a manual, on-demand identity mapping mechanism. Third, we consider not only a centralized architecture (TP) for policy tracking, but also other alternatives (ORIGIN and USER).

In the ambitious work of [28], Squicciarini *et al.* proposed a policy enforcement mechanism for content sharing across a distributed system based on *self-controlling objects* (SCO). An SCO is a movable data container, composed of *Content*, *Application* and *Network* components. The Content component encapsulates a shared data and its security policy. Application and Network sections are responsible for policy enforcement and synchronization between SCOs. Policy enforcement can be done at the time of access by each SCO itself, since the core security modules are embedded in the Application section. For preserving consistency of SCOs, modification to an SCO's content will be propagated to all the SCO copies with the help of a synchronization algorithm.

Their work has a much wider scope than the current one (i.e., content sharing in SCSs). As we pointed out in the introduction, their work considers policies that are not typical for social computing, and thus they do not have to consider the problem of State Fidelity, which our work is targeting to address.

A novel aspect of our work is the formalization of "safe" functions for SMC, together with static analysis for detecting safe functions, as well as the enumeration of idiomatic safe functions. The problem of defining "safe" functions and determining which functions are indeed safe, is an important and open research direction in the study of SMC protocols. We are aware of two lines of work that attempt to answer this question, yielding very different notions of "safe" functions. *Differential privacy* [11], when considered in the context of SMC [24], tries to add randomized noise to the output the function being computed in order to guarantee that any party's decision to participate in the protocol does not change the probability of any observable event except with a small probability (for all possible inputs). The definitions are strong and independent of the input distributions but also quite hard to achieve and often require adding significant noise to the desired output. *Belief tracking* [9], when considered in the context of SMC (e.g., see [23]), tries to reason for each party, whether it is beneficial to take part in the protocol or not. This is done by tracking the other participants' *believes* about the value of the party's input before and after the protocol takes place. Our definition of "safe" functions is the first one that is deterministic and non-probabilistic.

6. CONCLUSION AND FUTURE WORK

We proposed a protection model for shared contents in a confederation of SCS. The model allows the formulation of policies that are typical in social computing, and achieves all of Policy Fidelity, Mechanism Fidelity and State Fidelity through distributed evaluation of policies. We considered two approaches to achieve privacy in policy evaluation, one based on PFE, the other based on the novel characterization of "safe" functions.

Future directions include the following: (a) a framework for administrating shared access policies in our model, (b) supporting the resharing of contents, (c) extending the model so that it is resilient to the breaking down of part of Assumption 1 (i.e., when manual identity mapping through account connection is not consistent), and (d) supporting third-party applications.

7. ACKNOWLEDGMENTS

We thank Fahiem Bacchus (University of Toronto) and Vijay Ganesh (University of Waterloo) for introducing us to QBF and QBF2 solvers. This work is supported in part by the NSERC Discovery Grants of Fong and Mohassel.

8. REFERENCES

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge, 2009.

[2] S. Bayless and A. J. Hu. Single-solver algorithms for 2qbf. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'2012)*, volume 7317 of *LNCS*, Trento, Italy, June 2012. Springer.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Fault-tolerant Distributed Computing. In *Proceeding of ACM Symposium on the Theory of Computation (STOC'88)*, pages 1–10, 1988.

[4] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS'08*, pages 192–206, Berlin, Heidelberg, 2008. Springer.

[5] G. Bruns and M. Huth. Access Control via Belnap Logic: Intuitive, Expressive and Analyzable Policy Composition. *ACM Transactions on Information and System Security*, 14(1), 2011.

[6] B. Carminati, E. Ferrari, and A. Perego. Enforcing Access Control in Web-based Social Networks. *ACM Transactions on Information and System Security*, 13(1), Nov. 2009.

[7] D. Chaum, C. Crépeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *ACM symposium on Theory of computing (STOC '88)*, pages 11–19. ACM, 1988.

[8] Y. Cheng, J. Park, and R. Sandhu. Relationship-Based Access Control for Online Social Networks: Beyond User-to-User Relationships. In *Proceeding of SOCIALCOM-PASSAT'12*, pages 646–655. IEEE, 2012.

[9] M. R. Clarkson, A. C. Myers, and F. B. Schneider. Quantifying Information Flow with Beliefs. *Journal of Computer Security*, 17(5):655–701, 2009.

[10] I. Damgard, M. Geisler, M. Krøigaard, and J.-B. Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, PKC'09, pages 160–179. Springer, 2009.

[11] C. Dwork. Differential Privacy. In *Automata, languages and programming*, ICALP (2), pages 1–12. Springer Berlin Heidelberg, 2006.

[12] P. W. L. Fong. Preventing Sybil attacks by privilege attenuation: A design principle for social network systems. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (S&P'11)*, pages 263–278, Oakland, CA, USA, May 2011.

[13] P. W. L. Fong, M. Anwar, and Z. Zhao. A Privacy Preservation Model for Facebook-style Social Network Systems. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 303–320, Berlin, Heidelberg, 2009.

[14] O. Goldreich. *The Foundations of Cryptography – Volume 2.* Cambridge University Press, 2004.

[15] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *ACM Symposium on the Theory of Computation (STOC '87)*, pages 218–229, 1987.

[16] W. Henecka, S. Kogl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *ACM CCS'07*, 2010.

[17] M. Jonota and J. Marques-Silva. Abstraction-based algorithm for 2qbf. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'2011)*, volume 6695 of *LNCS*, Ann Arbor, MI, USA, June 2011. Springer.

[18] J. Katz and L. Malka. Constant-Round Private Function Evaluation with Linear Complexity. In *Advances in Cryptology ASIACRYPT*, volume 7073, pages 556–571. Springer, 2011.

[19] M. Ko, H. Touati, and M. Shehab. Enabling Cross-Site Content Sharing between Social Networks. In *Privacy, Security, Risk and Trust*, PASSAT'11, pages 493–496, 2011.

[20] M. N. Ko, G. P. Cheek, M. Shehab, and R. Sandhu. Social-networks connect services. *IEEE Computer*, 43(8):37–43, Aug. 2010.

[21] V. Kolesnikov and T. Schneider. A Practical Universal Circuit Construction and Secure Evaluation of Private Functions. In *Financial Cryptography and Data Security*, volume 5143 of *LNCS*, pages 83–97. Springer Berlin Heidelberg, 2008.

[22] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay–A Secure Two-party Computation System. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 20–20, Berkeley, CA, USA, 2004.

[23] P. Mardziel, M. Hicks, J. Katz, and M. Srivatsa. Knowledge-oriented Secure Multiparty Computation. In *Proceedings of the 7th ACM Workshop on Programming Languages and Analysis for Security*, PLAS'12, pages 1–12, New York, NY, USA, 2012.

[24] I. Mironov. Differential Privacy as a Protocol Constraint. In *Information Theory Workshop (ITW)*, pages 81–83. IEEE, 2012.

[25] P. Mohassel and S. Sadeghian. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. In *Advances in Cryptology EUROCRYPT 2013*.

[26] M. Shehab, M. Ko, and H. Touati. Enabling Cross-site Interactions in Social Networks. *Social Network Analysis and Mining*, 3(1):93–106, 2013.

[27] A. Squicciarini, F. Paci, and S. Sundareswaran. PriMa: An Effective Privacy Protection Mechanism for Social Networks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS'10, pages 320–323, New York, NY, USA, 2010.

[28] A. C. Squicciarini, G. Petracca, and E. Bertino. Adaptive Data Protection in Distributed Systems. In *Proceedings of the third ACM Conference on Data and Application Security and Privacy*, CODASPY'13, pages 365–376, New York, NY, USA, 2013.

[29] D. Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, pages 175–183, Gaithersburg, MD, Sept. 1986.

[30] A. Tapiador, V. Sánchez, and J. Salvachúa. An Analysis of Social Network Connect Services. *CoRR*, 1207, 2012.

[31] A. Yao. Protocols for Secure Computations. In *IEEE Symposium on Foundations of Computer Science (FOCS'82)*, pages 160–164, 1982.

APPENDIX

A. MEMBERSHIP OF IND IN Π_2^P

Proof. To demonstrate that IND is in Π_2^p, we note that the complement of IND is in Σ_2^p, as every triple (f, I, J) that does not belong to IND satisfies the following:

$$\exists \vec{a} \in \mathbb{B}^m \,.\, \exists b \in \mathbb{B} \,.\, \exists \vec{v} \in \mathbb{B}^k \,.\, \exists \vec{w} \in \mathbb{B}^n \,.\, \forall \vec{u} \in \mathbb{B}^n \,. \quad P(\vec{a}, b, \vec{u}, \vec{v}, \vec{w}) \tag{3}$$

where $P(\vec{a}, b, \vec{u}, \vec{v}, \vec{w})$ holds whenever:

$$(\mathsf{proj}_J(\vec{w}) = \vec{a}) \wedge (f(\vec{w}) = b) \wedge \big((\mathsf{proj}_I(\vec{u}) \neq \vec{v}) \vee (\mathsf{proj}_J(\vec{u}) \neq \vec{a}) \vee (f(\vec{u}) \neq b)\big)$$

Predicate P is obviously checkable in polynomial time. □

B. Π_2^P-HARDNESS OF IND_0

Proof. To demonstrate that IND_0 is Π_2^p-hard, we reduce a known Π_2^p-complete problem to IND_0. The specific Π_2^p-complete problem we will use is $\Pi_2\mathsf{SAT}$ [1, Example 5.6].

Problem: $\Pi_2\mathsf{SAT}$

Input: a boolean formula $\phi(\vec{x}, \vec{y})$, where $\vec{x}$ and $\vec{y}$ are disjoint lists of variables in ϕ

Question: Is it the case that $\forall \vec{x} \,.\, \exists \vec{y} \,.\, \phi$?

Let $\phi(\vec{x}, \vec{y})$ be a propositional formula. We specify below the construction of a pair (ψ, I).

- Let z_1 and z_2 be boolean variables that do not belong to $\{\vec{x}, \vec{y}\}$.
- Let $\psi = z_1 \rightarrow (\neg\phi \wedge z_2)$.
- Let $I = \{\vec{x}, z_2\}$.

We claim that $\phi(\vec{x}, \vec{y})$ is in $\Pi_2\mathsf{SAT}$ if and only if (ψ, I) is in IND_0.

Suppose $\forall \vec{x} \,.\, \exists \vec{y} \,.\, \phi$. Note that $\psi = 1$ is realizable by setting $z_1 = 0$. In that case, $\vec{x}$ and z_2 can assume any value and ψ would remain 1. Note also that $\psi = 0$ is realized when $z_1 = 1$ and $\phi = 1$. The supposition above guarantees that for every setting of $\vec{x}$, ϕ can be turned to 1. The value of z_2 does not matter. In summary, $(\psi, I) \in \mathsf{IND}_0$.

Suppose $\exists \vec{x} \,.\, \forall \vec{y} \,.\, \neg\phi$. Note that $\psi = 0$ is realizable by setting $z_1 = 1$ and $z_2 = 0$. For $\psi = 0$, it must not be the case that $(\neg\phi \wedge z_2) = 1$. Yet the above supposition guarantees there exists some $\vec{u}$ such that if $\vec{x} = \vec{u}$ then $\neg\phi = 1$ no matter what $\vec{y}$ is. Further setting $z_2 = 1$ renders $(\neg\phi \wedge z_2) = 1$. Therefore, $(\vec{x}, z_2)$ must not be $(\vec{u}, 1)$ when $\psi = 0$ is realized. In summary, $(\psi, I) \notin \mathsf{IND}_0$. □