Secure sharing in distributed information management applications: problems and directions

Piotr Mardziel, Adam Bender, Michael Hicks, Dave Levin, Mudhakar Srivatsa, Jonathan Katz

***Abstract*—**
Interaction between entities who may not trust each other is now commonplace on the Internet. This paper focuses on the specific problem of sharing information between distrusting parties. Previous work in this area shows that *privacy* and *utility* can co-exist, but often do not provide strong assurances of one or the other. In this paper, we sketch a research agenda with several directions for attacking these problems, considering several alternative systems that examine the privacy vs. utility problem from different angles. We consider new mechanisms such as economic incentives to share data or discourage data leakage and a hybrid of code-splitting and secure multi-party computation to provide various assurances of secrecy. We discuss how to incorporate these mechanisms into practical applications, including online social networks, a recommendation system based on users' qualifications rather than identities, and a "personal information broker" that monitors data leakage over time. We hope that this paper will spark ideas and conversation at ACITA about directions most worth pursuing.

I. Introduction

The rise of *distributed information management* (DIM) applications has followed the rise of the Internet. In these applications, *users* store information on a *site* for the purpose of sharing it with *recipients*. Users have incentive to share data; for example, users build *social capital* when sharing data on a social network like LinkedIn,[1] creating future opportunities for work or collaboration. On the other hand, sharing too much information is dangerous because the recipients of the information, or the system itself (assuming it is not controlled by the information owner), may have incentive to share sensitive data with *eavesdroppers* that a user has not authorized to view his data. For example, Alice could report to Bob's former employer that he has joined a company in violation of his former employer's IP agreement, having discovered this information on LinkedIn.

Government entities such as the military are in an analogous situation: they have incentive to share sensitive information—about potential targets, suspicious activities, technical problems, or vulnerabilities—with partners at differing levels of trust. Misuse of this information may result in harm. But harm is also possible if information is not shared, as the information could be necessary to prevent loss of life, assets, or advantage.

Our research aims to design mechanisms and protocols toward building applications that aim to balance these competing principles of *need to know* with *responsibility to share*. We would like both sites and users to be disincentivized or prevented from sharing information with eavesdroppers, while at the same time to be incentivized to share information, to add value to their work. In this paper, we outline a research agenda toward building more secure information management applications. We begin (§ II) by describing three styles of application that motivate our work—online social networks, information hubs, and sensor networks—and describe the goals and incentives for users and administrators of these applications. Next we discuss ideas for employing economic models and mechanisms for describing the incentives of participants in these applications with the goal of developing policies that balance the risk and reward of information sharing (§ III). In the following two sections we consider mechanisms useful for permitting no more sharing than necessary between mutually-distrusting parties (§ IV), including means to quantitatively estimate the information released during an interaction (§ V). Finally, we sketch future plans and conclude.

II. Applications

In this section we present three applications that serve as motivation for the mechanisms we discuss. The first, online social networks, have shown exponential growth in the number of users over the past few years. The second, a collaborative reviewing application, is a general class which encompasses many current and future uses. The third, sensor networks, has both scientific and military applications.

A. Online social networks

An online social network (OSN) is an application through which users can easily share certain types of information, such as personal expertise and interests, noteworthy professional or personal events, photos, or messages. One popular OSN is Facebook, which has upwards of 500 million active users, 50% of which log in at least once per day [1]. LinkedIn is another popular social network specializing in professional interactions, e.g., for finding employment and making business connections. The Pentagon has recognized that OSNs are valuable for military personnel, recently reversing a ban on the services [2] and launching its own social media hub to better facilitate intra-organization interactions [3].

In the terminology of distributed information management applications that we introduced above, the *site* is the OSN and *users* and *recipients* are OSN participant members. Participants are attracted to OSNs because they make it easy to organize information and share it among a select group, where such information sharing would have been tedious or non-scalable otherwise. Several OSNs serve as a platform for third-party

Mudhakar Srivatsa is located at the IBM T.J. Watson Research Center; all other authors are with the University of Maryland, College Park.

[1]http://www.linkedin.com

applications with which their users can interact, and these interactions are often customized based on a participant's personal data: one popular Facebook application is a calendar that tracks the birthdays of a user's friends [4].

Most OSNs are free to users, adding to their appeal, and generate revenue from advertising. Users' personal data, all of which is available to the site, is of particular interest, since it can be used to display targeted ads. For example, based on a user's status posts and profile information the site can attempt to select ads a certain user is more likely to be interested in; better selection leads to more profit.

To summarize, participants have incentive to share data with their friends and associates and with the site and third-party apps to improve their interaction experience. Third-party applications also often rely on advertising and thus want to attract as many users as possible to their own sites; customizing the experience based on user data is thus useful. The site has incentive to attract as many users as possible and to collect as much data as possible to increase its revenue. Indeed, researchers have observed that the terms-of-use agreement for Facebook is quite draconian [5]; it requires that users "not provide any false personal information on Facebook", "keep [their] contact information accurate and up to date", and grant Facebook a non-exclusive license to any content a user posts.

In the setting of OSNs, our interest is to develop techniques that facilitate sharing without unnecessary information release. Our approach is to design policies that permit users to make informed decisions about what information they wish to share and mechanisms that enforce fine-grained sharing policies.

B. Collaborative reviewing

Another class of application that must balance the privacy/utility trade-off is what we call an *information hub*. Popular examples of such systems are Slashdot [6] and Reddit [7]. The users of information hubs post snippets or links to articles they think might be of interest, and recipients respond with comments about the posting. Some sites bubble highly-favored posts to the top, incentivizing clever and interesting content from users. The hosting service itself has incentive to acquire as many readers as possible, usually to increase ad revenue.

Information hubs are valuable to the defense community. The U.S. intelligence community uses Intellipedia [8], a classified variant of Wikipedia [9], for organizing information of inter-agency interest. STRATCOM's 4-star blog aims to serve a similar purpose, encouraging contributions outside the chain of command [10]. Here there are similar incentives: posts/articles are rated directly or indirectly and posters/authors can be commentators and vice versa. This encourages posting of high-quality information in order to improve reputation.

In addition to such existing examples of information hubs, we also conceive of a collaborative reviewing system for academic research, a web-based system exhibiting many information sharing issues we are studying as well as being a useful tool for the research community. The site would host not only academic papers but also public reviews of these papers. Reviews include numerical scores and textual comments and serve to inform others about the papers they might be interested in reading. Users of the site will have a profile that includes relevant information, such as their name, job, papers they have authored, and their interests coupled with their expertise; e.g., Bob might be interested in embedded systems but not know that much about the topic, and Alice might be interested in sensor networks and be an expert. Subsequent consumers of these reviews can rate them as well, either with a simple "thumbs up" as with Reddit [7], or with more detailed information, such as whether the commenter agrees with a review's conclusion or if the review is technically sound or thorough. The weight of these ratings may depend on the rater's credibility. Applying these ideas, the site is ultimately useful for identifying high-quality papers and area experts, i.e., those whose reviews or papers are highly regarded.

Reviewers may sometimes wish to remain anonymous. This is for the same reason the scientific community uses anonymous peer-reviewing today—to protect a reviewer who provides a negative opinion from backlash. At the same time, anonymity (without further assurance) may reduce the credibility of the review. Thus there is incentive to share some information so that others can judge the review, but not enough information to identify the reviewer. Another problem is weighing a reviewer's self-proclaimed expertise. Users would like corroboration of these claims. Such corroboration could come from other users who may agree to the user's expertise, or it may come from artifacts, such as published papers (or program committee memberships) in the declared area of expertise. The reviews themselves add to their own credibility the more they stand alone in justifying their conclusions.

In short, there is incentive to share information—to increase reputation and to improve the overall value of the system—but some disincentive as well—due to the potential for negative backlash. If we imagine such a system used in a military setting, as with the STRATCOM blog, we can imagine a similar set of issues: commanders want the best information but may blame the bearer of bad news, thus incentivizing anonymity but in doing so potentially reducing credibility. Finding ways to balance these competing aims is an important research challenge.

C. Sensor Networks

Operators of sensor neworks must also balance the trade-off between sharing certain data (measurements and aggregates) while protecting others (device locations). An adversary that learns the locations of sensors can interfere with measurements or capture the sensors [11].

Sensors networks do not exactly fit the DIM model explored in this paper. Specifically, the *users* (sensors) cooperate with the *site* (the operator), which determines the policies for sharing with *recipients* (third-parties that wish to use sensor data). However, they can still benefit from the techniques we discuss. As an example, consider two operators of distinct networks, N_1 and N_2, located near each other. N_1 wishes to use readings from N_2 to augment its own data collection. For sharing to be successful, each must take precautions. At the very least, N_2 must aggregate (or otherwise mask) its data so that N_1 cannot determine the location of the sensors. N_2 may wish to further limit the data it shares so that N_1 cannot use it to attain a tactical or competitve advantage over N_2. On the other hand, N_1 must ensure that N_2 is not sharing bogus data. It can do so by, for instance, ensuring that readings from sensors closest to N_2's sensors are similar. It can also compare aggregated statistics (i.e., the average temperature over the past week is the same in both networks) for consistency.

III. Developing Sharing Policies

Each user of a DIM application must determine what information he is willing to share with each potential recipient. His choice is limited by the sorts of sharing policies the site will enforce; we consider mechanisms for enforcing fine-grained policies in the next section. For now, consider policies that are specified as access controls for users acting in certain roles. For example, Facebook policies apply to friends, friends of friends, the public, and sometimes user-specified groups.

To determine an effective sharing policy, users must balance the utility/privacy trade-off of sharing the information. To allow them to do so, we wish to determine what information can best inform their choices. Specifically, we ask: (1) What data can the system monitor or gather that describes the cost/benefit trade-off? (2) How can this information be gathered reliably and efficiently? (3) What metrics and algorithms can use this data to effectively inform a sharing decision? We present some exploration of these questions in the remainder of this section.

A. Valuing information: Indicators of whether to share or not

When DIM users consider whether to share information or not, their concerns will be abstract. For example, if a user Bob shares his religious beliefs on an OSN, he may encourage communication from like-minded people (whom he may not otherwise realize are like-minded) and develop stronger relationships with them. On the other hand, he may also tickle the prejudices of others and elicit a negative reaction. The question Bob must answer is: will the benefit outweigh the cost?

To answer such questions, users would like to gather evidence that support a decision to share or not to share. Such evidence could take many forms:

- **Positive vs. negative**: a decision to share can cause events that are either good or bad for the user.
- **Observed vs. provided**: a user may be alerted to the ramifications of a decision either through observed evidence (e.g., losing friends in an OSN) or through soliciting reactions directly from users (e.g., a posted article received many "thumbs-down" votes).
- **In-band vs. out-of-band**: the application by which the information is shared (e.g., the OSN) can gather the evidence, or it may be acquired via some means outside the system.
- **Trustworthy vs. untrustworthy**: The reputation of the evidence provider (whether a system, individual, or otherwise) may be taken into account when using it to assess value.

Creating policies that incorporate evidence requires evaluating various forms of evidence. A first step is to identify useful evidence by observation of existing DIM use.[2] By gathering data of user activities over time, we can create a taxonomy of evidence, its sources, and effects in real-world usage to help guide user policies. We can also observe a user's behavior over time. If a user's activity increases, then we can infer that some positive value is being received from interaction. By observing many users' activities, we can identify events that correlate with various levels of site usage.

As the above information arises from the DIM itself, it can be reliably gathered. We may also wish to correlate such data with externally-visible data, such as public Web sites. For example, a Georgia school teacher was recently fired because parents complained of information and photos posted on her Facebook page [13]. Such a negative outcome is a powerful disincentive to certain kinds of sharing and such knowledge is useful when determining policies. One challenge is developing means to automatically acquire such information and to assess the credibility of the source.

B. Economics-based metrics

To encourage productive sharing and discourage illicit information release, we can apply ideas from markets. Placing monetary value on sensitive data enables a broader range of policies. When a user shares data with a site, they agree upon a price for the data. The price does not have to be a fixed value—it can represent recurring payments, for example, $1 per month that the user allows the site to use the data. Compensating users for their data gives them some recourse if data is leaked.

This approach is similar to the existing Internet economic model built around advertising. Sites share their resources (the attention of readers) with a third-party (the advertiser) by placing an ad on their site and receive compensation in return. The type of ad and amount of compensation varies depending on the composition and size of the audience. Advertisers pay the site either a fixed price for a given amount of space on a Web page, or on a per-impression/per-click model that depends on how often the ad is displayed or clicked on. We borrow concepts from this model and show how they apply to users sharing their resources (data) with a third-party (a hosting site).

Marketplaces and pricing: There are many possible methods of data valuation. We envision a "data marketplace," much like a stock ticker, that reports on the current "going rate" of various pieces of information. When a user shares data with a site, the two parties enter into an agreement on what the data is worth and how the user will be compensated for it (possibly only in the case of leakage).

[2]One problem with such a data-based study on Facebook in particular is that user data is proprietary, and storing it, even when it is accessible, is a violation of the Facebook license agreement. The MySpace user data [12] archive may be useful in this regard.

Payment schemes: We present three different possible payment schemes: a one-time payment upon data transfer, a one-time payment upon data leakage, and a recurring payment (possibly depending on usage).

- **One-time payment upon data transfer.** In this scheme, the site pays the user the current going rate for each piece of data that is shared. Once the site has the data, it assumes ownership and may share the data with whomever it chooses (as it has no incentive to not share the data). One advantage of this system is that no monitoring system is required to detect leakage. However, care must be taken to ensure the user-provided information is correct—a user could easily create many accounts and share bogus data to collect multiple payments.
- **Recurring payment.** In this scheme, in addition to a possible up-front payment, the site makes payments on a regular basis of an amount that is a function of the data the user shares, how often it is viewed, how much the user interacts with the site, and other factors. With this scheme the user must employ a method to detect leakage but the incentive to create fake accounts is minimal, as the payment depends on the amount of interaction with the site, which imposes a cost on maintaining a fake profile. However, there is no guarantee that the site will erase or cease to use the user's data once the contract is broken; data transmission is irreversible. Hence this scheme is most appropriate for short-lived data, such as preferences.
- **One-time payment upon data leakage.** In this scheme, the user stores data with the site and trusts it to not share with eavesdroppers. The site and the user enter into a contract dictating how the site will compensate the user if and when the site ever leaks the data, intentionally or otherwise. This arrangement requires a monitoring system to inform the user of when data is leaked, and a method to prove that the site leaked the data. Here, a user has no incentive to create a fake profile, as there is no guarantee that the data will ever be leaked. Thus the information entered is more likely to be accurate. However, monitoring systems are expensive and often incomplete [14]. As this scheme provides the strongest disincentive to leak data, it is most appropriate for sensitive or unchanging data.

These policies are compatible with today's advertising-based sites (indeed, we expect that sites will be able to pay users with the money their earn by using the user's own data!). The first two policies allow sites to earn revenue in the same way they do today. The last still allows sites to show untailored ads and, because it is appropriate for sensitive data, allows the site to charge users for secure storage of their data. By using (a combination of) these three policies, a user and a site can enter into a mutually-agreeable contract that gives incentive to users to share data and to sites to protect that data.

IV. Enforcing Policies: The Principle of Least Sharing

One way to avoid unauthorized release of information is not to share it in the first place. For example, rather than send private data to OSN servers which perform ad selection, an alternative would be for the user to run the OSN's ad selection algorithm on locally-stored data and provide the OSN with the result. If the choice of ad does not identify the user, the OSN learns less about the user but can still serve targeted ads.

Such an approach is a form of *privacy-preserving computation.* Abstractly, the goal of privacy-preserving computation is as follows, simplified to client (user) and server (site). We wish to compute some function $F(x_1, \ldots, x_n, y_1, \ldots, y_m)$ where $x_1, ..., x_n$ are private data to the server and $y_1, ..., y_m$ are private data to the user. After the computation, both sides know the output of F, but do not learn anything about the other's private input, other than what is implied by the result.

A. Prior work

Prior work has taken two basic approaches to solving the privacy preserving computation problem, *computation splitting* (CS) and *secure multiparty computation* (SMC). The first is exemplified by Jif/Split [15], [16] and its descendant, Swift [17], and the second by Fairplay [18], [19]. We describe each in turn and then consider generalizations of these ideas that we think are worth exploring.

Computation splitting We can protect $x_1, ..., x_n$ and $y_1, ..., y_m$ by splitting the computation between client and server, using static information flow analysis [15]. Essentially both sides will declare policies about their data, including *declassifications* that allow some information about the data to be shared in particular ways to the other party. For example, rather than releasing all of a credit card number to the remote party, the declassification may release only its last four digits. The static analysis ensures that any such declassifications take place before any secret data is sent to the remote party. Code that operates on non-secret data can be placed on either the client or server, as the code is not considered secret (we will consider altering this assumption shortly).

In the end, both parties learn the result of the computation and the secret values declassified by the other side. The declassified values may be implied by the result, or may convey additional information. As degenerate cases: If there are no $x_1, ..., x_n$ to be protected, we can ship the entire code to the client to perform locally; conversely, if the client is willing to share the entirety of $y_1, ..., y_m$, then these can all be shipped to the server, and the computation can be performed there.

Secure multiparty computation Fairplay [18] requires no declassifications, but rather uses an interactive protocol and cryptographic techniques to protect data as it is shipped back and forth between client and server to compute F. It is more flexible than Jif/Split in that it ensures less information is released to the remote party, but also more expensive. Fairplay was originally developed for two-party computations, but has been generalized to n-way computations in FairplayMP [19].

B. Application to DIM applications: research problems

Return to the OSN example: suppose the server wishes to perform a computation on the user's data, and the user wishes to store his data locally, releasing as little of it as possible.

There are a number of problems to solve that are not addressed by previous work, and we sketch some of them here. First, the server might not only care to keep certain values hidden from the client, but to hide portions of the code as well. Second, as parts of this algorithm may now be computed by the client, the server must trust the client to execute them properly. The server would like assurance that the algorithm execution has not been tampered with. Finally, both the client and server are concerned about what private data the other side has learned, which depends on (in addition to the privacy-preserving mechanism) what computation has been performed. We need a way to track the information released to ensure it does not exceed some threshold.

C. Keeping F hidden from the client

Suppose the server wants to hide some or all of F. For example, suppose F is the OSN's ad selection algorithm. If the algorithm is very effective the OSN would prefer to keep it hidden from competitors. Most reliably, the entire computation would occur at the server, and as the computation proceeds, it would send read requests to the client, asking for bits of data it needs. However, this approach could reveal more data than the client desires. For example, suppose the site wishes to determine the average age of a user's friends. We might do this by defining $F(u)$ for a given user u as follows:

```
$n = 0$; $c = 0$;
**foreach** $f \in friends(u)$
    $n = n + 1$; $c = c + age(f)$;
**return** $c/n$
```

Consider three possible ways this code could be computed between client and server:

1) F runs entirely at the server. It must query the client for each of u's friends f, so that it can average the friends' ages.
2) F runs entirely on the client. Thus the user releases far less data—just the average age of his friends, not the number of friends or their identities. However, running F at the client reveals the function the server wishes to compute—if u just provides his friends list to the server, he does not know what about his friends the site finds interesting, protecting the site's algorithm.
3) As a middle ground the server could ask for the number of u's friends and their ages. Doing so does not reveal what the site will do with this, i.e., compute the average—the server could be computing the median, finding the maximum or minimum, etc.

Which division of computation is used is a matter of policy. The programmer could label portions of F as server-private, and the CS or SMC algorithm could ensure that none of that code runs at the client. For example, labeling all of F as private results in the first case, above; labeling none results in the second; labeling only the expression c/n results in the third.

A research challenge is to reconcile the constraints on code location imposed by privacy requirements of the client's data and the privacy requirements of the code. Moreover, even if "all" the code runs at the server, the client can infer properties of the algorithm by observing individual read requests that it makes. We must ask: at a high level, what can the client reason about the overall form of F based only on seeing the portion of its computation that runs at the client? How do we describe knowledge of a particular function, based on the computation?

While hiding the code would seem to make the splitting problem harder, it actually can make it easier in some cases. In particular, the static analysis performed by CS algorithms is designed under the presumption that the client will know when it could be receiving server-private values (or functions of them) because it knows the code. If we assume that some or all of the code is unknown, the client may not know the difference between a query $y_1 < 5$ and $y_1 < x_1$ (where x_1 is a server-private value that could be 5), and thus will not realize it is learning something about private server data.

D. Ensuring computation consistency

We might imagine that allowing a client to compute part of F is ill-advised because the client could lie about the result. Let F' be the portion of computation F that runs at the client, and suppose $F'(y_1, ..., y_m) = x$. When asked, instead of reporting x, the client could report some value x' instead.

Lying about computation results is not necessarily as bad as we might initially think. If F' were computed at the server instead, the client could provide the server with false input values $y'_1, ..., y'_m$ such that $F'(y'_1, ..., y'_m) = x'$, yielding the same false result. Facebook recognizes the problem of users lying about their details, and makes such deceit a violation of the terms of service (as mentioned in § II).

On the other hand, while the site cannot enforce user honesty, the server can check whether client-side computations are *consistent*. In particular, the server might like to ensure that the client does not change or lie about its private value in the middle of a computation. For example, suppose the server was interested in the following function F:

```
**if** $c_1 < 5$ **then** $x = h_1$ **else** $x = h_3$
**if** $c_1 < 10$ **then** $z = h_2$ **else** $z = h_3$
```

In this code, if the first if-branch succeeds, then the second if-branch should succeed too. If the first else-branch succeeds, either of the subsequent two would be OK. Now suppose that the server executes all of this code segment other than the two comparisons $c_1 < 5$ and $c_1 < 10$, which run at the client. As there is no c_1 such that $c_1 < 5$ and not($c_1 < 10$), the server should be suspicious if successive computations on the client produce contradictory answers. Ensuring consistency of this sort within a computation might, for example, assuage advertisers that they are getting a consistent view of user data.

To ensure client-side computations are mutually consistent we can take the following approach. For each computation $Q_1 = q_1, ..., Q_m = q_m$ where Q_1 is the function sent to the client (whose free variables include $y_1...y_m$) and q_1 is the result returned, the server can ensure that

$$Q_1 = q_1 \wedge ... \wedge Q_m = q_m$$

has a satisfying solution. If not, the client has lied or incorrectly computed one or more of the requests. Notice that inconsistency can arise even for a single request, e.g., if Q_1 is $y_1 \bmod 3$ and the client returns $q_1 = 4$.

Because some data changes over time—e.g., friends and personal likes/dislikes—the server cannot know for sure that a client is lying if successive series of queries are inconsistent. Models of data mutability can help distinguish between data that should remain fixed (e.g., a user's birthday) and data that may change infrequently (e.g., a user's favorite movie).

Satisfiability queries can be expensive and thus should not be used on the critical computation path. Assuming that clients lie infrequently, the server could take a sampling approach to verifying consistency and perform such verification offline.

E. Brokering private information

So far we have been concerned only with single executions of a privacy-preserving computation. When parties may interact more than once, computing different functions over the same data, each party must track what its past partners have learned so as to not reveal too much data over time. We propose using a guard that monitors what information an entity is releasing. We call this approach a *private information broker* (PIB): a record of what bits of data have been released and to whom they were released, as well as a model of how those entities are likely to behave in terms of sharing data with other entities. With this mechanism, users can try to understand what might happen if this information base is extended by revealing further information.

Each individual would run a service that maintains these models for them. Each time a user enters their personal information, such as when providing their qualifications for a review in the collaborative reviewing information hub, the information broker is informed and constructs a model of the hub and tracks the information it has learned. Before actually submitting the information to the site, the broker service checks to ensure that a user-specified security policy is respected, that is, sharing the new information does not push the amount of information that an entity can learn or infer over a certain threshold. The conservative assumption is that the hub could reveal the information to the public, but the user could refine this assumption by entering the hub's privacy policy as further information.

The PIB is a natural place to execute split computation. By having direct access to the user's secret data, it can monitor which bits are read during function execution using static or dynamic taint tracking. This simplifies the task of tracking released information. One party querying another would treat the other's PIB as a database and send an SQL-like query directly to the PIB to extract the relevant information. This query is likely to be less sensitive than the entire computation, and yet is sufficient for the user to analyze what bits of his data are going back to the server, allowing the PIB to improve efficiency while maintaining a record of all direct data access. However, data may leak indirectly, through side channels; we discuss this in detail in the next section.

V. Quantitative information flow

Whether splitting the computation of F between server or client or computing F entirely at the client, F's result may reveal information about the client's private inputs. At the extreme, if $F(y_1) = y_1$, then the result of F unveils all the client's private information (y_1). Even if the result of a single computation reveals only a small amount of information, many computations performed over time may reveal substantial portions of $y_1, ..., y_m$.

To measure (and ultimately, control) what information is released by a computation, a first attempt might be to track what portions of the secret values $y_1...y_m$ (respectively, $x_1...x_n$) are actually returned directly by the portion of F computed at the client (respectively, the server). But to do so would fail to track *implicit flows*. To illustrate, consider the following function F that runs at the client:

```
if $y_1 == 10$ then return $0$ else return $1$
```

In the case that F returns 0, the server knows the only possible value of y_1 is 10, despite the fact that F returns no portion of y_1 directly. Any assessment of information release must consider such implicit flows if the secret values are to be kept truly secret.

A. Quantitative information flow tracking

Directly learning the value of a secret variable is too simplistic to usefully characterize information flow. In the implicit flow example above, even if y_1 is not equal to 10, execution does leak information (that $y_1 \neq 10$), though not enough information to completely learn y_1. Assume $y_1 \in \{1, ..., 10\}$ and after learning the variable is not equal to 10, the server learns the output of the following function:

```
if $y_1 == 9$ then return $0$ else return $1$
```

At this point, the server knows $y_1 \leq 8$, Comparing the secret y_1 to successively smaller values, the server could eventually learn the exact value of y_1. A conservative analysis of the situation could imply that y_1 is leaked upon the very first function call due to the implicit flow, whereas a more fine-grained perspective would not only indicate some partial revelation of the variable but also conclude that a sequence of function outputs described above increase this partial amount, certainly beyond what is learned from just one function call.

One approach to quantify what is learned is to consider how many possible values the secret can take on given some observed output of the function. Consider a simple ad selection algorithm that operates on two user secret variables, $byear$, the user's birth year in the range $\{1900, ..., 1999\}$, and $gender$, the user's gender as a numerical value from $\{0, 1\}$, and returns an ad in the set $\{0, 1, 2\}$.

```
if $1980 \leq byear$ then return $0$
    else if $gender == 0$ then return $1$
    else return $2$
```

The set of possible secret *states* is $U = \{1900, ..., 1999\} \times \{0, 1\}$, containing 200 distinct possible secret pairs of values of $byear$ and $gender$. If the server (here, an advertiser) were to learn the output of this function is 0, it would know the set of possible secret values that could have resulted in this output is $O_1 = \{1980, ..., 1999\} \times \{0, 1\}$, which contains 40 distinct states. On observing an output 1, however, it would know the secret input was in $O_2 = \{1900, ..., 1979\} \times \{0\}$, or one of 80 possible states. One way to quantify the leak of information is in terms of the sizes of the sets of secret states [20].

In our example, observing an output of 0 reduces the set of secret states by a greater amount than observing an output of 1. Intuitively, this corresponds to sharing a greater amount of information. This intuition stems from the fact that, if the server was to model the secret state and sample from it uniformly the probability that the server could correctly *guess* the exact value of the secret information is twice as high in the first case as the second. Backes et al. [20] analyzed implicit flows in such a scenario, assuming that each possible secret state is equally likely. But this assumption does not hold for our example given the non-uniform distribution of birth years.

This unrealistic situation can be remedied by considering the probability distribution or *belief* of the secret states [21]. A belief δ can be thought of as a function from the set of secret states U to $[0, 1]$, having the usual probability distribution property, $\Sigma_{\sigma \in U}\ \delta(\sigma) = 1$. A more realistic belief of the secret state in our example would assign more probability mass to states denoting a younger user, or for example:

$$\begin{aligned}&\delta(byear, gender) = \\ &\quad \textbf{if } byear \leq 1949 \textbf{ then } 1/400 \textbf{ else } 3/400\end{aligned}$$

Beliefs can thus account for differing *a priori* probability distributions of secret states. For example, without learning anything from the client, the server believes that $(1949, 0)$ has a probability 0.0025 of being the true secret state, whereas it believes $(1950, 0)$ has a 0.0075 chance of being correct.

The belief signifies nothing of what the actual secret state is. It can, however, be quantitatively compared to the true secret state or other beliefs using *relative entropy* [21]. The relative entropy $D(\delta \to \delta')$ between two probability distributions δ, δ' is defined as follows:

$$D(\delta \to \delta') = \sum_{\sigma \in U} \delta'(\sigma) \lg \frac{\delta'(\sigma)}{\delta(\sigma)}$$

If $\dot{\sigma}$ is probability distribution that assigns 1 to the real secret state and 0 to everything else, then $D(\delta \to \dot{\sigma})$ is a quantification of the information known about the secret state and has the property that the probability of guessing the real state when sampling from δ is $2^{-D(\delta \to \dot{\sigma})}$. As such, users can compare different quantities of information about the secret state in terms of likelihood of guessing the secret state. If a server (or user) has beliefs δ and δ' with $D(\delta \to \dot{\sigma}) = D(\delta' \to \dot{\sigma}) + 1$ then it is twice as likely to guess the secret having belief δ than it would having belief δ'; gaining a *bit* as measured in this way corresponds to a doubling of the chances of guessing the secret state.

Evaluating the revision of a belief given some function output lets one compare the information gain that results from the function. The semantics of evaluation of programs over probability distributions as described in [21] let one do exactly that. However, this technique requires policy-enforcement mechanisms to reason about all possible secret states. In our ad-selection example only 200 states were possible, but large, sensitive computations are now the norm [22], [23], [24]. Propagating distributions of large state spaces over the execution of a (possibly distributed) program and computing the relative entropy over such distributions is not feasible.

Some of our present work is aimed at alleviating this problem in a constrained setting of programs that operate on expressions that contain only linear combinations of variables (variables are not multiplied together) and where probability distributions are uniform over linearly constrained regions. Our probability distribution δ above, for example, can be described by a set of *polyhedral regions* [25] within U:

$$P_1 = U \cap \{(byear, gender) : 1900 \leq byear \leq 1949\ ,\ 0 \leq gender \leq 1\}$$

$$P_2 = U \cap \{(byear, gender) : 1950 \leq byear \leq 1999\ ,\ 0 \leq gender \leq 1\}$$

Each state within P_1 has equal probability according to δ, and similarly within P_2. Inspired by ideas from abstract interpretation [26], we aim to evaluate programs on such sets of polyhedra of states instead of individual states and thereby make the task feasible in settings where the state space is too large to reason about otherwise.

B. Enforcing quantitative policies

Given a strategy to quantitatively track information leaked by executing a function F, a user can set a policy governing the maximum leakage permitted for his secret data. For example, it has been reported that zip code, birthday, and gender are sufficient information to uniquely identify 63% of Americans in the 2000 U.S. census [27], so we could imagine a user setting a policy that will permit (guarded or partial) access to these data assuming an attacker never has a better than $1/n$ chance to guess all three, for some n. By extending the private information broker (§ IV-E) to quantitatively track implicit flows, policies can more flexibly express what data can be released to different parties while maintaining rigorous estimates of the uncertainty of attacker beliefs, to prevent responses that would reveal too much information.

The PIB must enforce a quantitative policy in a way that does not itself violate the policy. If the PIB executes a function F locally and estimates how much the remote party could learn from the result, and if this amount is too great, it can intentionally cause the computation to fail (i.e., return nothing). The problem with this approach is that the remote party might assume that failure implies that the actual response would leak too much information, which can itself leak information. Consider the code

$$\textbf{if } byear = 1971 \textbf{ then return } 0 \textbf{ else return } 1$$

Suppose a client whose birth year is 1971 runs this code and discovers that returning 0 will reveal his birth year to the server, leaking too much information. But failing, where failure is known to be due to exceeding the allowed information leakage budget, reveals nearly the same amount of information, since the server knows it is likely the budget was exceeded due to having guessed the birthday.

There are several approaches to solving this problem. First, we could add nondeterminism into the result (e.g., by adding noise, or by failing randomly) of *every* query, which improves privacy of the data provider at a cost to the utility of the data consumer. Second, we could attempt to estimate the *potential* for a function to leak information before running it and refuse to execute a function if it could potentially reveal too much.[3]

A last problem is that *auxiliary information* must be considered when setting quantitative policies. The de-anonymization of the Netflix Prize dataset [24] shows how auxilliary information can be used to violate the privacy guarantees of "anonymized" data. The problem of handling auxiliary information has motivated a line of research now termed *differential privacy* [28]. The goal of differential privacy is to avoid presuming an *a priori* belief of attackers, and rather to reason that no matter what information an attacker might already have, a particular query will only increase that information by a relatively small amount. These techniques have largely been designed with the idea of aggregation queries over a database sampling from a large population, e.g., to determine the average age of people within a large geographic area. With such queries, information about an individual can be intuitively kept hidden. In the case of DIM applications operating on *individual* information, there is no population within which to hide. Thus we believe differential privacy is not universally applicable. Nevertheless, finding situations where differential privacy does apply would allow us to make use of, or extend, existing techniques with provable guarantees.

VI. Conclusions

This paper has presented an overview of the problem of securely sharing information among mutually-distrusting parties in a distributed system, along with directions of research toward solving the problems that arise. By examining online social networks in particular and community-based review and sensor network applications in general, we describe factors that should be taken into consideration when crafting information sharing policies, namely evidence of reactions to prior sharing efforts and economic incentives to abide by policy. We also discuss mechanisms for enforcing such policies, with a focus on collaborative computation, beginning with the ideas of secure multiparty computation and computation splitting. We propose that these mechanisms be augmented with means to keep sensitive code hidden, to ensure that collaborative computation is consistent and efficient, and to quantitatively track knowledge about private information that can be inferred from the results of computations. We believe we have sketched a rich agenda for research ahead at the cross-section of programming languages, cryptography, economics, and systems.

Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

[3]This notion of a potential information release is similar to the notion of function sensitivity in differential privacy [28], also discussed in this section.

References

[1] "Facebook | statistics," http://www.facebook.com/press/info.php?statistics, Aug. 2010.
[2] C. Weber, "Pentagon allows social networking on military computers," *Politics Daily (on-line)*, Mar. 2010, http://www.politicsdaily.com/2010/03/02/pentagon-allows-social-networking-on-military-computers/.
[3] "Dod social media hub," http://socialmedia.defense.gov, May 2010.
[4] "Birthday calendar," http://www.facebook.com/BirthdayCal.
[5] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM*, 2009.
[6] "Slashdot," http://slashdot.org, May 2010.
[7] "Reddit," http://www.reddit.com, May 2010.
[8] Reuters, "U.S. intelligence unveils spy version of Wikipedia," October 2006.
[9] "Wikipedia," http://en.wikipedia.org, May 2010.
[10] "Four-star blogging at STRATCOM," http://www.defenseindustrydaily.com/fourstar-blogging-at-stratcom-0239/, Mar. 2005.
[11] M. Anand, E. Cronin, M. Sherr, M. Blaze, Z. Ives, , and I. Lee, "Sensor network security: More interesting than you think," in *HotSec*, 2006.
[12] D. Worthington, "Myspace user data for sale," *PC World on-line*, Mar. 2010, http://www.pcworld.com/article/191716/myspace_user_data_for_sale.html.
[13] D. Dukes, "Facebook causes Barrow teacher's firing," *Fox 5 Atlanta (online)*, Nov. 2009, http://www.myfoxatlanta.com/dpp/news/facebook+causes+barrow+teacher's+firing+111009.
[14] M. Srivatsa, S. Balfe, K. G. Paterson, and P. Rohatgi, "Trust management for secure information flows," in *CCS*, 2008.
[15] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers, "Untrusted hosts and confidentiality: Secure program partitioning," in *SOSP*, 2001.
[16] L. Zheng, S. Chong, A. C. Myers, and S. Zdancewic, "Using replication and partitioning to build secure distributed systems," in *Security and Privacy*, 2003.
[17] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Secure web applications via automatic partitioning," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 31–44, 2007.
[18] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay—a secure two-party computation system," in *SSYM*, 2004.
[19] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: a system for secure multi-party computation," in *CCS*, 2008.
[20] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic discovery and quantification of information leaks," in *Security and Privacy*, 2009.
[21] M. R. Clarkson, A. C. Myers, and F. B. Schneider, "Quantifying information flow with beliefs," *J. Comput. Secur.*, vol. 17, no. 5, pp. 655–701, 2009.
[22] S. Hansell, "AOL removes search data on vast group of web users," *New York Times*, Aug. 2006.
[23] N. Prize, http://www.netflixprize.com/.
[24] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Security and Privacy*, 2008.
[25] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *POPL*, 1978, pp. 84–96.
[26] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*, 1977, pp. 238–252.
[27] P. Golle, "Revisiting the uniqueness of simple demographics in the us population," in *WPES*. New York, NY, USA: ACM, 2006, pp. 77–80.
[28] C. Dwork, "Differential privacy," in *ICALP*, 2006.