Sharing Private Information Across Distributed Databases

Michael Siegenthaler
Dept. of Computer Science
Cornell University
msiegen@cs.cornell.edu

Ken Birman
Dept. of Computer Science
Cornell University
ken@cs.cornell.edu

***Abstract*—In industries such as healthcare, there is a need to electronically share privacy-sensitive data across distinct organizations. We show how this can be done while allowing organizations to keep their legacy databases and maintain ownership of the data that they currently store. Without sending or mirroring data to any trusted, centralized entity, we demonstrate how queries can be answered in a distributed manner that preserves the privacy of the original data. This paper explains our distributed query execution engine, outlines how to bootstrap the system when only real world identifiers such as a name and date-of-birth are initially known, and offers details on the tradeoff between privacy and performance. We evaluate the scalability of this approach through simulation.**

I. Introduction

With the ubiquity of computers and the Internet, it has become commonplace to store data about people in electronic databases, and also to exchange that data with business partners and service providers electronically. In domains where much of the data is considered private and confidential, however, the technology to exchange that data in a secure manner has lagged far behind the ability to store such data locally. Different hospitals, for example, may each use a separate proprietary database that their own staff and certain closely affiliated external specialists may use, but any communication with an organization not in the system must be performed manually via paper printouts, faxes, and phone calls. We present a system that can support the exchange of such data while revealing only the minimum amount of information that is necessary to accomplish a particular task. Our technology is not domain specific, though it is perhaps most compelling for the healthcare industry because in that area there is both a desire to openly share information with anyone who needs it and a high expectation that data will not be exposed to public view or otherwise fall into the wrong hands.

We envision a system where data remains stored with the organization that generated it, rather than being handed over to some trusted centralized entity. Participating organizations may run SQL-like queries against the distributed data, subject to some policy which grants or denies access on a fine-grained level. In this paper we show how to discover the existence of remote data and how to run queries against it. We provide the following privacy features:

This work was supported, in part, by the Institute for Information Infrastructure Protection (I3P) and the National Science Foundation.

(a) *Data privacy*: The query asker learns only the answer to the query, and not any of the data used to compute it.
(b) *Query privacy*: The data owner does not learn the query, only that a query was performed against a particular user's information.
(c) *Anonymous communication*: Query askers and data owners do not know who the opposite party is.

It is possible, in our current system, to pose queries which directly or indirectly violate one or several of the above properties. We focus on legitimate queries in this paper; details on access control are in our prior publication [1].

Data privacy is the motivation for our work, since it permits data to be shared on a minimally revealing, need-to-know basis. Anonymous communication is also important because either the query asker or the data owner might be a specialist, for example an AIDS clinic, where merely revealing that the patient is in some way associated with an organization of that nature would constitute a privacy violation. Similarly, a query might contain information about a specific condition of the patient that some of the data owners do not already know about.

It must be noted that our model of data privacy is one of soft constraints. It would be unthinkable to impose restrictions so strict that an emergency room doctor does not have access to vital information for a patient who has been injured during distant travel from his home hospital and primary care provider. At the same time, there are less critical situations in which only limited information should be revealed because the data user, while authorized, may not be trusted to guard the information as stringently as the provider of that data does [2]. Our contribution is a technology that enables such heterogeneous use-scenarios by soft enforcement of privacy policy.

II. Related Work

Traditional database security techniques can be grouped largely into discretionary and mandatory access control. [3] Discretionary access control permits access controls to be specified over columns or entire tables either on a per-user or role-based [4] basis. Mandatory access control defines permissions in terms of an ordered set of classes such as "top secret" or "confidential". Each data object and each principal is labeled with a class, and reads and writes are restricted such

that information can never flow from a higher security class to a lower one.

Privacy protection additionally requires row-level access control, for which prior work exists in both the discretionary [5] and mandatory [6] security models. We believe that the discretionary access control offers more flexibility in a world of read-only data sharing among a diverse set of organizations, and will assume that such a model is in place. The specific policies may be described in a privacy policy specification language such as P3P [7] or EPAL [8].

Highly relevant is the prior work in the area of Hippocratic databases [9]. These databases are designed such that data collection and disclosure are only performed with the consent of the user who's data is in question. Limited disclosure has been previously explored for centralized databases [10]. Our work differs in that we focus not on policy enforcement itself, but on a distributed execution model that limits disclosure when multiple parties collaborate to answer a query.

In the area of distributed databases, the TIHI (Trusted Interoperation of Healthcare Information) project [11] uses a trusted mediator to determine which queries should or should not be allowed. More recently, several commercial systems have been developed that are actually centralized databases which information is pushed into from distributed sources. The most well known of these are Google Health [12] and Microsoft HealthVault [13]. These are consumer-facing services which allow the patient to enter and manage their own information, as well as create sub accounts to allow healthcare providers direct access. Each company provides an API to allow healthcare providers to push/pull information to/from the centralized record. All of the commercial systems are characterized by data storage at a centralized, trusted entity. While the convenience of a single comprehensive personal health record may outweigh the privacy risks for many people, these systems do not eliminate the need for providers to keep electronic medical records for their own internal use. We believe there will always exist a need for these providers to selectively share data from their internal records with affiliated organizations, as permitted by law and required by business processes, without requiring the patient's involvement.

The trusted third-party can be eliminated by cryptographic techniques, as investigated by Agrawal et al [14]. Their work is more theoretical and focuses specifically on the join technique; they show that it is possible to perform a join without involving a third party comparer as we do. Their technique relies on commutative encryption and shipping entire database tables across the network; one of the hosts can then compare the encrypted values and decide whether the plaintext matches, without seeing the plain text except for items that are already in that host's local database. Although we present an alternative join technique, the cryptographic methods could certainly be integrated into our system as well, and would be particularly useful if a need arises to perform joins on personally identifying data such as names.

Canetti et al. [15] showed how to solve a similar problem as ours where an aggregate value is computed such that the asker does not learn the source data and the data owners do not learn the query. Their work, however, concentrates on scenarios where the query and the data, but not the answer, is sensitive, and where the data owner does not find out which records were queried. For our application, an answer may sometimes be more than just a boolean or numerical value, and should be treated as potentially sensitive, while the knowledge that a particular user's record was queried does not reveal anything useful as long as our other privacy properties are met.

A large body of work exists on information leakage from supposedly anonymized databases [16], along with solutions including k-anonymity and l-diversity. [17] Although our work does not involve data publishing per say, some of the same attacks could conceivably be carried out by a party that has the opportunity to pose multiple queries in sequence. We do not address such data mining attacks in this paper, instead taking the view that once a user has authorized an organization to pose queries, minimum disclosure principles should be applied on a best-effort basis and not so strictly that the user might be denied proper medical care because a legitimate query could indirectly lead to a privacy violation.

Also relevant is the Security Assertion Markup Language (SAML) [18], the eXtensible Access Control Markup Language (XACML) [19], and related standards. These XML-based standards support exchanging authentication and authorization data, and declaring access control policies, respectively. We see SAML-based systems as providing a policy (possibly in XACML) for which types of queries are allowed and which are not, and our system then executing the query in accordance with the policy.

III. Data and Query Model

A. Provider Characteristics

Let us suppose that personal information is stored at organizations known as *data providers*. A data provider typically is an existing brick-and-mortar business which maintains information about its customers in some internal database with a proprietary schema. We are not concerned about this schema; it is sufficient for the organization to provide some read-only views into this data conforming to standardized schemas. Each view consists of tuples of the form $(id, attr1, attr2, \ldots)$ where id is a locally unique identifier for each person.

B. Network Model

Data providers are in a network with universal connectivity and asynchronous messaging. In this paper we use the Internet with a web services layer built on HTTP+SSL. Providers are themselves responsible for ensuring the availability and persistence of any data that they control. A provider may in general consist of a collection of nodes; individual nodes perform actions on behalf of the provider and the addition or removal of a particular node should not be visible outside of that provider.

C. Security Model

Standard cryptographic capabilities are assumed. Providers have public keys K and private keys k. A provider A communicating with B is able to sign messages it sends using a function $S(k_A, \mathsf{msg})$ and verify the signature on messages that it receives using $V(K_B, \mathsf{msg})$. It may also encrypt and decrypt messages using functions $E(K_B, \mathsf{msg})$ and $D(k_A, \mathsf{msg})$. Additionally, providers must be able to verify that any other providers with which they communicate are authorized to participate in the system. This is accomplished by having the public keys signed by a trusted accreditation agency.

D. Query Requirements

Participants in the system wish to run queries against data that is distributed throughout the system. Known identifying information, such as a social security number or a combination of name and date-of-birth is used to indicate whose data the asker is interested in; we do not consider the use of such information to be a privacy threat in this context because it is typically already known by the types of organizations targeted by our work. We do not consider erroneous identifiers or persons with multiple aliases in this paper.

Queries are written in a relational algebraic language similar to SQL. Standard select, project, and join operations are performed against tables that are fragmented across various providers, and the results may range from yes/no answers computed over the data to actual values of data found in the tables. Results must not contain any information that the asker is not authorized to discover. In this paper we focus on the query execution, leaving the decision of determining which queries should be permitted for future work.

All aggregation of information across table fragments at different organizations is performed at query time. Records are kept separate in the underlying databases and only the links between them are stored; this ensures that any erroneous linkages may be undone without leaving the original data in a polluted state.

E. Example Query

It is helpful to consider some specific examples of the types of queries our system supports before looking at the architecture in detail.

Example 1: A pharmacist needs to check if a patient might be vulnerable to a drug interaction. This involves creating a list of drugs that might conflict with the prescription currently being filled, and querying the system to see if any of those drugs have been taken by the patient in the past. A yes/no answer should be returned, but the pharmacist should not learn anything more about the patient's medical history. Any data providers who help answer the query (hospitals, other pharmacies) should be assured that the patient has authorized a query, but they should not learn what is being queried for, or from which pharmacy the query originates.

Example 2: Two companies have agreed to a joint-venture and wish to find out who their mutual customers are. Both companies should learn the intersection of their customer databases, but should not learn anything about customers of the other company who are not in the intersection. Third-parties cannot be trusted and thus should not learn anything that might be useful to them.

IV. Architectural Model

A. Requirements

The goal of the architecture is to support queries that reveal enough information so that organizations can go about their business, and no more. For the sake of clarity and to permit better optimization later, we split querying into two phases. Phase 1 performs a *global search* for records pertaining to the person in question and returns a set of *data handles*, each of which indicates the presence of a record somewhere in the system but does not reveal where that record resides. Phase 2 uses the data handles to execute a relational algebraic query, while keeping the original data hidden from the asker and keeping the query hidden from the data owners.

An anonymous communication primitive is required both in global search and query execution. For this, we rely on the well known technique of onion skin routing [20]. A message m from provider A to provider B will be encrypted first with the pubic key of B, K_B. The tuple $(B, E(K_B, m))$ will then be encrypted with K_C and the result combined into a tuple with the address of C, and so on, recursively. At some point, A sends the encrypted message to the provider D whose public key was used for the outermost layer of encryption. D decrypts the message and sees a tuple $(C, \mathsf{binary\ blob})$, and sends it to C, who does the same and sends the result to B, who finally uses its private key k_B to recover the original message m. A response may be sent by having each provider keep a log of forwarded messages and where each came from for a short interval, and routing responses along the path in reverse. The plaintext of the response may be encrypted with a symmetric key that was chosen by provider A and included with the original message m. This is done so that only provider A, and not those along the reverse path, can read the response. In addition to using routing that is reminiscent of the peeling of an onion, a robust anonymous messaging system divides communication into rounds with fixed message sizes and requires all nodes to send a message in each round, even when idle. This prevents a traffic analysis attack from uncovering who speaks with whom, and is only required if an adversary is assumed to have the capability of monitoring link traffic in multiple locations within the network.

B. Search Mechanism

Global search is a mechanism for discovering the existence of records pertaining to a particular person, without learning where those records are actually stored. A global search operation takes a commonly used, unique identifier such as SSN or name/DOB combo as input and returns a set of abstract data handles that may be used to route messages anonymously to the data owners.

Abstractly, global search may be thought of as a mechanism to broadcast an identifier to all organizations in the system

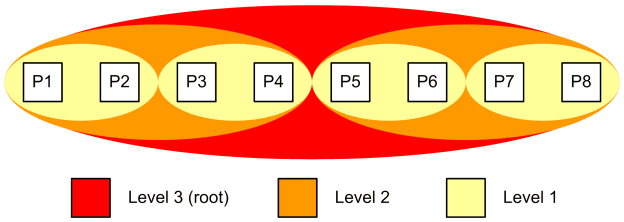

Fig. 1. Providers organize into a hierarchy of groups

and collect responses from those organizations at which there is a match. In practice, a global broadcast would not scale well, so we need to impose some intelligent limits on which organizations a search touches.

To achieve scalability, our system organizes into a hierarchy of groups, with the lowest grouping level having a handful of providers per group and the highest level containing every provider in the system. Figure 1 illustrates the resulting structure of groups and subgroups. Each group has an associated Bloom filter [21]. A provider inserts its customer list into the Bloom filter for every group that the provider belongs to.

A search is performed by sending a request to a designated provider who maintains a local copy of the *root level* Bloom filter. In general there are multiple such providers, to balance the load. This provider checks the Bloom filter for a match, and finding one, recursively sends the request to the designated providers for the subgroups composing the next lower level. Alternately, if no match is found, the request is silently discarded. The same process takes place at any non-root groups that a search request percolates down to. Eventually the non-discarded copies of the request arrive at the lowest level of the group hierarchy, where the base case of the recursion checks the request against the actual local database instead of another Bloom filter. Here again, the request may be discarded (due to a false positive in the Bloom filter), or if a match exists, a *data handle* is returned to the provider that initiated the search.

The data handle is an onion skin route, with a nonce to identify the matching record at its core. This permits the provider who initiated the search to communicate with the data owner in the future, without either of them knowing the identity of the other. For brevity of notation, we will use an $\odot$ to represent a data handle when the details of the onion skin route are not important. For example, a data handle pointing to nonce 34 at provider 2, and routed through provider 1 is simply:

$$\Big(P_1, E\Big(K_{P_1}, \big(P_2, E(K_{P_2}, 34)\big)\Big)\Big) = \odot(34)$$

In order for data handles to be returned to an anonymous search originator, each search request includes an onion skin route. Wherever a match occurs, this route can be utilized to reply to the search originator. This raises a few issues:

(a) The plaintext of the search response (data handle) must be encrypted with the public keys of all the providers on along the route, so that the encryption may be stripped off in layers and the plaintext recovered at the destination. Knowing these public keys, however, would defeat the anonymity of the route.

(b) The search originator, who created the route, may have chosen the outermost encryption layer (first hop) to be a colluding host who will reveal the identity of any data owner who sends a message via the route.

We solve (a) by having each search request include an *aggregate public key*, in addition to the onion skin route, which when applied in an encryption operation has the same effect as the application of each individual public key in sequence. Since encryption primitives are based on linear operations, this is not hard to achieve. The data owner can thus create an encrypted message such that the encryption may be stripped off in layers by the application of each private key along the route, hiding the message from intermediate hosts. The data owner cannot cannot learn the path because that would require checking exponentially many combinations of the public keys in the system to find which combination gives the aggregate public key. We solve (b) by allowing the data owner to add several additional layers of encryption to the outside of the onion skin route, thereby introducing additional hops between itself and the potentially colluding host in order to hide its identity.

To ensure privacy, it is important for each group to contain a diverse set of providers; otherwise information could be leaked by, for example, observing that there exists a record for SSN 112-55-3434 in a group where every provider in the group specializes in treating alcoholism. Additionally, false positives in the Bloom filter help to protect privacy. A false positive gives the appearance that a particular person has a record within a group, when in fact no such record exists. When a search triggers a false positive, it will proceed to search each subgroup, but will eventually be silently discarded when there is no match. Thus, an adversary who observes a match in a particular group cannot be certain whether it is a true match or a false one.

C. Bloom Filter Configuration

In order to balance the privacy provided by false positives during the search process against the performance hit of taking too many false positives, it is necessary to appropriately set the Bloom filter parameters, k and m. For a filter containing n records, the proportion of false positives r is given by [22]:

$$r = \Big(1 - 1\big(1 - \frac{1}{m}\big)^{kn}\Big)^k$$

Parameter m configures the length of the bit vector representing the Bloom filter, and k indicates how many unique hashes of each record are inserted into the filter. The value of k is typically a small integer (usually less than 10) in order to prevent the Bloom filter from becoming completely filled in with 1's (leading to 100% false positives) too quickly. We rewrite the above equation to find m for a given r, n, and k.

$$m = \frac{1}{1 - \big(1 - r^{\frac{1}{k}}\big)^{\frac{1}{kn}}}$$

Finding the parameters for an efficient (in terms of storage space) filter is thus reduced to trying each $2 \leq k \leq 10$ (for example) and choosing the one that results in the smallest m. Table I shows the storage space that would be required for a variety of false positive rates at population sizes that might be encountered in various levels of the search hierarchy.

TABLE I
STORAGE SPACE REQUIRED FOR EACH BLOOM FILTER BY FALSE POSITIVE RATE AND POPULATION SIZE

False Positives	Population					
	1 K	**10 K**	**100 K**	**1 M**	**10 M**	**100 M**
5%	780 B	7 KB	78 KB	780 KB	7 MB	78 MB
10 %	601 B	6 KB	60 KB	601 KB	6 MB	60 MB
25 %	360 B	3 KB	36 KB	360 KB	3 MB	36 MB
50 %	203 B	2 KB	20 KB	203 KB	2 MB	20 MB
75 %	124 B	1 KB	12 KB	124 KB	1 MB	12 MB

D. Join Technique

Once the data handles pertaining to the desired query have been discovered, the actual query must be executed. It is not safe for the organization that is asking the query to simply retrieve the records from the remote hosts, since those records might contain a substantial amount of private information which is used to compute the answer of the query but which should not itself be revealed. Nor is it acceptable to send the query to the data owners and ask them to return the answer; the query itself might contain information that should be kept private from the data owners, or the answer may depend on data from a set of providers.

We assume that select and project operations can always be performed at the location where the relevant data resides. Joins, on the other hand, are where the aforementioned problem arises because the tables being joined cannot necessarily be shared. Select operations may be rewritten as joins if it is required to hide the select condition from the data provider at which the table being selected from resides.

The procedure for performing a query involving a join is shown in figure 2. First, the query asker uses the data handle to route a message to the data owner expressing an interest in performing a join. Each provider selects a random number, transmits its hash, and upon receiving the other side's hash reveals the number it chose. The sum of these numbers is used as a random seed to choose a *blind comparer* to perform the join, and the hashes serve to prevent either provider from choosing its number such that the comparer will be a colluding host. Two onion skin routes are also created so that the query asker and the data owner each have a way to send messages to the comparer without revealing their own identities.

Next, the asker splits the query into two portions. The first part consists of a simple select statement which accesses the relevant data; this gets sent anonymously to each data owner, using the data handles as onion skin routes, along with instructions that the data owner should send the selected data to the blind comparer by way of the specified route. The second part consists of the original query, from the point of the join onward, and gets sent also to the blind comparer.

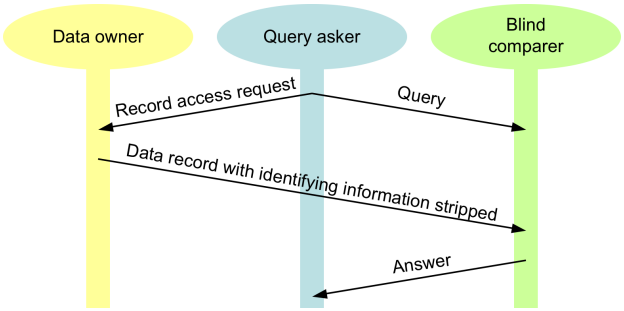

Fig. 2. Query execution overview

"Blind" in this sense means that the comparer cannot see where either the data or the query came from; the comparer does in fact see both the query and the data in plain text, which is a requirement for being able to perform the join. Queries are written such that any personally identifying data (name, SSN...) is projected away before performing a join; thus any sensitive facts learned by the blind comparer are useless because they cannot be traced to a particular person. Even if the facts themselves are enough to uniquely identify a particular individual, the host would need to have a substantial amount of background knowledge in order to make such an inference, and this is unlikely for a randomly chosen blind comparer.

Taking a step back, we observe a tradeoff between revealing the query to the data owner and revealing the data to some third-party, depending on the selectivity of the query portion that gets sent to the data owner.

(a) At one extreme, the data owner learns the full query.
(b) At the other extreme, the data owner must ship all of it out to a random blind comparer (who won't be able to do anything with it)
(c) A general question can be posed using method (a) to filter the data somewhat, and then the specific question of interest posed using method (b)

The filter function should be as specific as possible, without revealing too much to the data owner, in order to reduce the amount of data transferred to the blind comparer. We do not consider in this paper how to determine the filter function, but it could be either specified by the query asker or chosen automatically from a set of application-specific filters that are defined along with the database schema.

E. Query Execution

The query given earlier in example 1 may be written as:

```
SELECT EXISTS (
  SELECT * FROM conflicts
  CROSS JOIN nonces
  INNER JOIN remote(drug_history)
  ON nonces.nonce = drug_history.nonce
  WHERE conflicts.drug = drug_history.drug
);
```

The conflicts table referred to in the query is a simple list of the conflicting drugs. The nonces table contains the set of data handles for the patient, discovered via an earlier global search operation.

TABLE II
CONFLICTS

drug
A——
B——

TABLE III
NONCES

nonce
⊙(34)
⊙(56)

The parse tree, shown in figure 3, may be executed starting at the leaves for the local tables and proceeding up the tree.

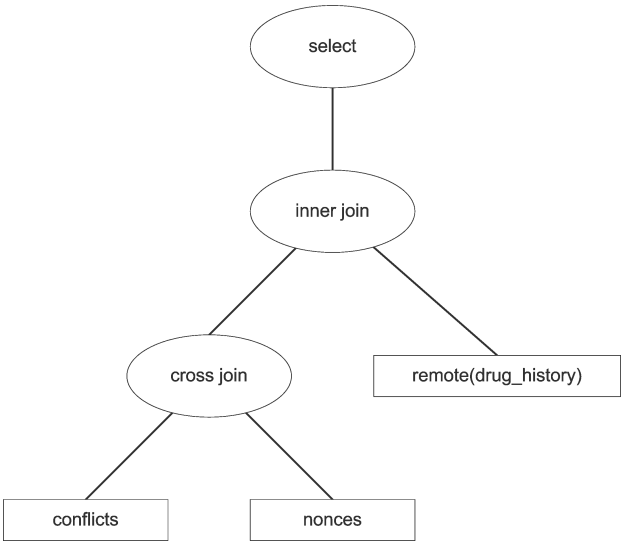

Fig. 3. Original query

When a join depending on remote data is reached, a filter query is sent anonymously to the data owner for each data handle, asking it to send the relevant data to a randomly selected blind comparer.

```
SEND (
  SELECT nonce,drug FROM drug_history
  WHERE drug_history.nonce = O(34)
);
```

The parse tree for this is shown in figure 4.

The intermediate table at the query asker (query_table) and the remaining portion of the query to be executed is sent to the blind comparer as well.

```
SELECT EXISTS (
  SELECT * FROM query_table
  INNER JOIN drug_history
```

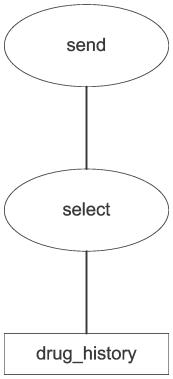

Fig. 4. Filter query, to be executed by the data owner

```
  ON query_table.nonce = drug_history.nonce
  WHERE conflicts.drug = drug_history.drug
);
```

This parse tree is shown in figure 5.

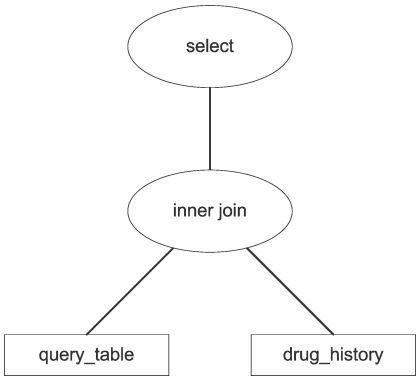

Fig. 5. Remainder of the original query that must be executed at the blind comparer

TABLE IV
QUERY_TABLE

drug	nonce
A——	34
A——	56
B——	34
B——	56

V. Scalability

To evaluate the scalability of our scheme, we performed some discrete event simulations on a realistic topology of healthcare providers and patients who have records at multiple providers. Due to the sensitive nature of real world data, several attempts to obtain an actual topology were unsuccessful; instead we generated our own, consisting of a small town with 35 providers and around 40 000 patients. Published statistics [23][24] indicate a rough power law distribution in the size of providers and in the frequency of patient visits, so the topology took this into account. A few simulations were also performed

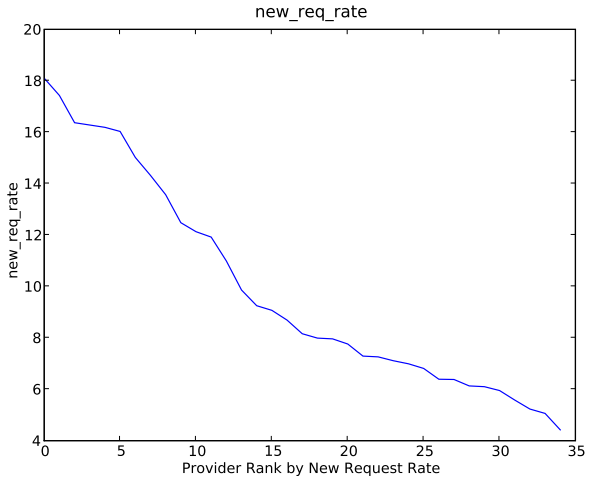

Fig. 6. Query generation rate

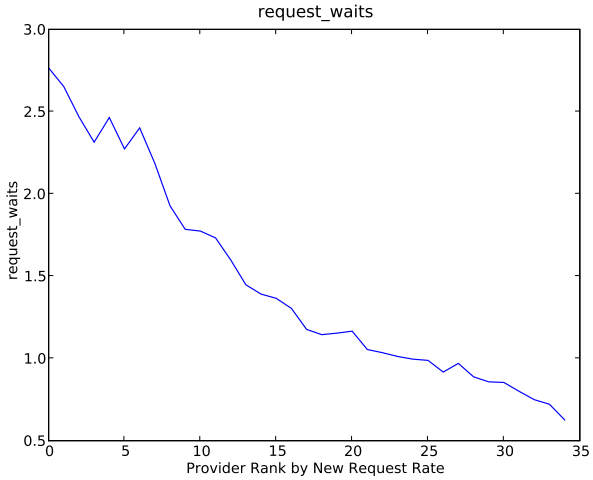

Fig. 8. Number of unanswered queries for which state is being kept

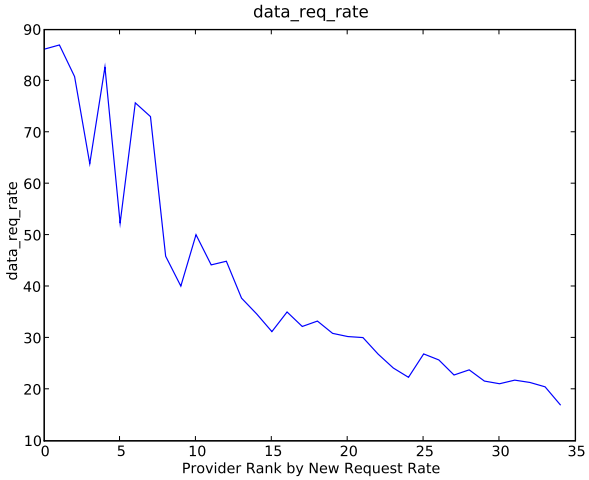

Fig. 7. Data request rate

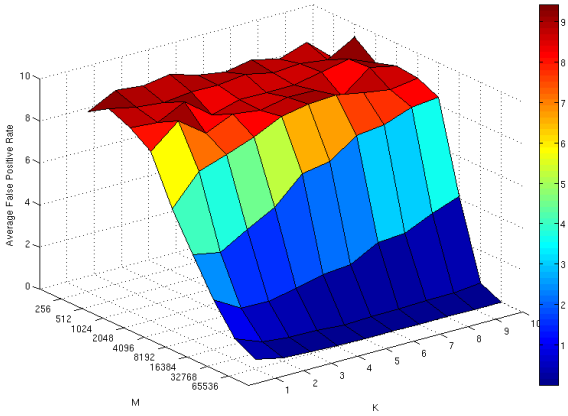

Fig. 9. Bloom filter parameters

with hundreds of providers and millions of patients; similar trends were observed as in the small town scenario.

Figure 6 shows queries being generated in the system at an aggregate rate of 350 queries per second. Some providers bear a larger portion of this load by virtue of having a higher number of active patients. Providers are sorted along the horizontal axis according to their load, and the same ordering of providers will be used in further graphs.

In figure 7 we observe actual requests for data as they arrive at the providers where the data is stored. Observe that the same group of providers who generate a high query load also bear a larger burden in serving those requests; this is good because those organizations will have more computing resources available.

Figure 8 gives a view into the amount of state that must be kept at each provider to keep track of unanswered queries. This of course depends directly on the latency in answering queries; the data shown assumes a 10 ms link latency between any two providers and an onion skin route length of 5 hops.

The Bloom filter used in searching for data records has adjustable parameters, and it is important to set them correctly in order to achieve enough false positives to protect privacy, yet not so many that every search request gets routed to every provider. In figure 9 we adjusted the the number of hashes, K, and the length of the filter array, M, while injecting 10 queries per second into the system (the results depend on the topology of patient to provider mappings, not on the query rate, hence the choice of a low value). In the red area, searches are essentially broadcast to every provider; in the dark blue, the Bloom filter is very selective and prevents false matches.

Within the middle zone of figure 9, false positives occur at various levels in the search hierarchy. Table V gives an example where only 2% of the searches actually get routed to a provider where no match exists, and the rest are eliminated in the hierarchy above that. Although those providers who are responsible for routing messages still see the searches at the

TABLE V
HISTOGRAM OF FALSE POSITIVE LOCATIONS FOR $M = 65536$ AND $K = 6$

Level	Cumulative proportion of false positives
1	2.0%
2	22.6%
3	81.3%
4	99.5%
5	100.0%

higher levels, this task can easily be distributed if the load is too high at any particular site.

VI. Access Control

Thus far we have presented a general framework for querying, which does not prevent the user from writing queries that deliberately reveal private data. Data privacy only holds to the extent that the answer to a query does not reveal the data used to compute it. Clearly, there must be a mechanism in place to decide which queries should be allowed based on a user-specified policy. Our system quantifies the privacy leakage that would result from a particular query, and uses a currency-like system to make the query asker pay for the right to know the answer. A more revealing query has a higher cost. The cost is not one of real-world money, but of symbolic tokens which are supplied by the system to organizations that are authorized to perform queries. Tokens are issued according to a budget that takes into account each organization's business needs and gives it enough to execute the required queries to fulfill those needs, but does not leave a significant excess to be used on extraneous queries. The full details of this technique are published elsewhere [1].

VII. Conclusion

We have presented here a system for storing and querying private data in a distributed manner. Anonymity of communication, data privacy, and query privacy are supported. The system scales and has parameters to adjust the tradeoff between better privacy and lower resource usage. More work is needed on the details, especially on the issue of deciding which queries are allowable and which are not, but the basic technology now exists to build an infrastructure that permits electronic sharing of sensitive data across organization boundaries.

References

[1] M. Siegenthaler and K. Birman, "Privacy enforcement for distributed healthcare queries," in *Pervasive Health 2009*, 2009.

[2] T. C. Rindfleisch, "Privacy, information technology, and health care," *Commun. ACM*, vol. 40, no. 8, pp. 92–100, 1997.

[3] R. Ramakrishnan and J. Gehrke, *Database Management Systems*. McGraw-Hill Science/Engineering/Math, 2002.

[4] R. S. S, E. J. C. K, H. L. F. K, and C. E. Y. K, "Role-based access control models," 1996.

[5] Oracle Corporation, "The virtual private database in oracle9ir2: An oracle technical white paper," 2002.

[6] T. level Vs. Element-level Classification, X. Qian, and T. F. Lunt, "Database security, vi: Status and prospects, 301–315. 1," 1992.

[7] L. Cranor, M. Langheinrich, M. Marchiori, and J. Reagle, "The platform for privacy preferences 1.0 (p3p1.0) specification," W3C Recommendation, Apr. 2002. [Online]. Available: http://www.w3.org/TR/P3P/

[8] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter, "Enterprise privacy authorization language (epal 1.2)," IBM, Tech. Rep., 2003. [Online]. Available: http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html

[9] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic databases," 2002, pp. 143–154.

[10] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt, "Limiting disclosure in hippocratic databases," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 108–119.

[11] G. Wiederhold, M. Bilello, V. Sarathy, and X. Qian, "A security mediator for health care information," in *Proceedings of the 1996 AMIA Conference*, 1996, pp. 120–124.

[12] Google Health, "http://www.google.com/health."

[13] Microsoft HealthVault, "http://www.healthvault.com/."

[14] R. Agrawal, A. Evfimievski, and R. Srikant, "Information sharing across private databases," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM Press, 2003, pp. 86–97.

[15] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright, "Selective private function evaluation with applications to private statistics," in *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2001, pp. 293–304.

[16] R. Anderson, "An integrated survey of medical databases," 1998. [Online]. Available: http://www.clcam.ac.uk/simrja14/caldicott/caldicott.html

[17] D. Kifer and J. Gehrke, "l-diversity: Privacy beyond k-anonymity," in *In ICDE*, 2006.

[18] OASIS, "Security assertion markup language (saml) v2.0 technical overview," 2008. [Online]. Available: http://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[19] ——, "extensible access control markup language (xacml) version 2.0," 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[20] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[21] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[22] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of bloom filters," *Inf. Process. Lett.*, vol. 108, no. 4, pp. 210–213, 2008.

[23] U. D. of Health and H. Services, "National ambulatory medical care survey: 2005 summary," 2007. [Online]. Available: http://www.cdc.gov/nchs/data/ad/ad387.pdf

[24] ——, "Summary health statistics for u.s. adults: National health interview survey, 2006," 2007. [Online]. Available: http://www.cdc.gov/nchs/data/series/sr_10/sr10_235.pdf