Maintaining Database Anonymity in the Presence of Queries

Ryan Riley[1], Chris Clifton[2], and Qutaibah Malluhi[1]

[1] Department of Computer Science and Engineering
Qatar University
{ryan.riley,qmalluhi}@qu.edu.qa
[2] Department of Computer Science
Purdue University
clifton@cs.purdue.edu

Abstract. With the advent of cloud computing there is an increased interest in outsourcing an organization's data to a remote provider in order to reduce the costs associated with self-hosting. If that database contains information about individuals (such as medical information), it is increasingly important to also protect the privacy of the individuals contained in the database. Existing work in this area has focused on preventing the hosting provider from ascertaining individually identifiable sensitive data from the database, through database encryption or manipulating the data to provide privacy guarantees based on privacy models such as k-anonymity. Little work has been done to ensure that information contained in queries on the data, in conjunction with the data, does not result in a privacy violation. In this work we present a hash based method which provably allows the privacy constraint of an unencrypted database to be extended to the queries performed on the database. In addition, we identify a privacy limitation of such an approach, describe how it could be exploited using a known-query attack, and propose a counter-measure based on oblivious storage.

1 Introduction

With the advent of cloud computing, the desire to outsource databases continues to grow. Database as a service is a quickly growing industry, attracting companies looking to reduce costs by maintaining fewer servers and IT personnel. However, as the usage of database outsourcing grows, so does the risk of privacy violations. In some cases this outsourcing may even conflict with privacy laws that are designed to safeguard the identities and the individuals the data is about. An outsourced database has a new threat to consider: the hosting provider itself.

Existing work has explored a variety of privacy constraints such as k-anonymity [1, 2], l-diversity [3], and t-closeness [4]. These works aim to provide metrics for the privacy protection of data stored in a database. Little work has been done to safeguard privacy in the queries themselves, beyond the extreme model of Private Information Retrieval [5] and related works that involved encrypting the entire database. In our model, we assume that the data is intentionally stored unencrypted so that the hosting provider can provide value added services such as address correction and analysis of the (anonymized) data.

Previous work involving unencrypted, anonymized databases ignores the impact of information contained in queries and essentially models queries as having been drawn randomly from the global pool of all possible queries, meaning they would not leak any sensitive information. Even the authors' prior work on querying anonymized data requires this assumption [6, 7]. This is rarely a reasonable assumption. The very existence of a query or set of queries can easily leak information about individuals in a database.

Consider the scenario that John is found collapsed on the street. The reason for his collapse is unknown. When he arrives at the ER, the doctors notice that John's arms contain punctures indicative of illegal drug use. In order to better determine John's situation, the doctor queries his medical records to determine if he has any of the diseases that may come from shared needles. The queries would look something like:

```
SELECT * FROM DB WHERE PATIENT = "John" AND (Disease = "HIV" OR
Disease = "hepatitis" OR Disease = "tuberculosis");
```

Given that those three diseases are considered high risk for illegal drug users but not for the general population, someone with a knowledge of those queries may be able to reasonably assume that John is an illegal drug user. (Why else would a doctor issue this particular set of queries?) Private information about John has been leaked, even if the database itself is stored in a privacy-preserving fashion. The queries themselves leak the information.

We model query privacy leakage based on the probability of a link between identifying information and sensitive information. For no leakage to occur, a query should not convey any private information that is not already revealed by the database itself. In the example above, queries for those three diseases increases the probability that John is at a high risk for diseases transmitted from blood. This knowledge, in turn, increases the probability that he has one or more of the diseases. A leak can be described as follows:

Given:
$t =$ An individual (or identifying information for that individual)
$v =$ A sensitive value
$D =$ A database
$Q =$ A sequence of queries
Private information is leaked if:
$Pr(t \text{ is linked to } v|D) < Pr(t \text{ is linked to } v|D, Q)$

In this work we propose a technique to build an anatomized database designed to safeguard the privacy of individuals whose data is being queried. We base our models on the principle of k-anonymity. The technique functions by separating individually identifiable users into buckets of size $\geq k$ and ensuring that queries to the database always involve at least an entire bucket. While our database model, described in more detail in [6, 7], allows INSERTs and UPDATEs, this paper only discusses SELECT queries. INSERTs and UPDATEs

SSN	*Name*	*Disease*
000-07-7083	Luis	HIV
000-26-9073	Donna	Diabetes
000-03-3060	Zachary	Hepatitis A
000-04-4396	Kenneth	Cancer
000-09-4349	Michelle	Tuberculosis
000-22-6531	Thomas	Hepatitis B

Fig. 1: Sample Database

inherently pose different privact risks because an attacker can analyze the before and after states of the database. Managing these risks imposes limitations on the statements and requires a certain amount of encryption [7]. The result is that INSERTs and UPDATEs that do not violate privacy based on the host comparing before and after states of the database inherently avoid the type of privacy violation described in this work.

The contributions of this work are as follows:

1. We identify and define the problem of private data leakage from the query in anonymized databases,
2. We provide a proof that can be used to demonstrate whether leakage can occur for many group-based privacy protection schemes,
3. We propose a hash based technique to prevent private data leakage through the query in a k-anonymized database,
4. We identify a type of privacy leak based on a known-query attack that would allow an attacker to violate query privacy, and
5. We propose the usage of oblivious storage as a mechanism to protect against known-query attacks.

1.1 Database Model

The basis for our database model is anatomization [8] with an encrypted join key [6, 7]. For the sake of simplicity of presentation we assume that the groupings provide only k-anonymity; however, similar privacy models may be also used without adjustments to our model.

In the anatomy model, the identifying information and the sensitive information are split into two separate tables, and a group number is used to link groups of items from both tables together. An attacker who is able to analyze the database cannot link a sensitive value to a particular identifying value, instead each can only be linked to the group it is a part of. An encrypted sequence number allows a client who knows the secret key to perform a query and then filter the results to determine the exact answer. Fig. 1 shows a simple database storing patient disease information. Now, suppose that we want to release this database while still maintaining the privacy of the individuals in it. We decide that we want to release the database to meet k-anonymity requirements with

SSN	*Name*	*GID*	*SEQ*
000-07-7083	Luis	1	1
000-26-9073	Donna	1	2
000-03-3060	Zachary	2	3
000-04-4396	Kenneth	2	4
000-09-4349	Michelle	3	5
000-22-6531	Thomas	3	6

(a) Identifier Table (IT)

HSEQ	*GID*	*Disease*
$H_{k_1}(1)$	1	HIV
$H_{k_1}(2)$	1	Diabetes
$H_{k_1}(3)$	2	Hepatitis A
$H_{k_1}(4)$	2	Cancer
$H_{k_1}(5)$	3	Tuberculosis
$H_{k_1}(6)$	3	Hepatitis B

(b) Sensitive Table (ST)

Fig. 2: Anatomized Database

$k = 2$, and so we ensure that each group contains at least two individuals in it. Fig. 2 shows the same database anatomized in this way. An attacker analyzing the database can only link a particular piece of sensitive information to a specific group, not to an individual within the group. The groups can be chosen using any group-based privacy criteria (such as l-diversity) in much the same way.

When a query is performed (either on the identifying information or the sensitive information) then all results from the corresponding group are returned. The client then uses the secret key to match the sequence number from the identifying information with the sequence number in the sensitive information in order to determine which elements of the group were actually queried. The details of query processing for such a database can be found in [6, 7].

We assume that one of the fields (in this case social security number) is used as the unique identifier for indexing the tables. We call this field the lookup key.

We explicitly assume an unencrypted database. While one might think that encryption should be used for data sent to a cloud provider, there are a number of good reasons not to do this:

– Databases commonly experience issues related to the accuracy and completeness of their data. Address information, phone numbers, zip codes, etc. may be incomplete. A cloud provider with an unencrypted database can provide "information fixing as service" to help fill in some of these gaps.
– Large, demographic queries that don't involve mixing identifiers and sensitive data (such as "How many customers do I have in Chicago?") do not require privacy protection, and can be performed on the unencrypted DB without a performance penalty. This could not occur in an encrypted DB.
– Data stored on the cloud can be offered to a third party for performing data analytics in order to extract useful information.

1.2 Threat Model

The owner of the database (the client) wishes to outsource their database to an outsourcing provider (the server). Before sending data to the server, the client anatomizes it. The client then queries the server requesting information about specific users, identifying them by their lookup key, as in this query:

```
   SELECT * from DB WHERE SSN="000-03-3060" AND
        Disease ="Hepatitis A";
```

The client should not issue queries that use any other field as the identifier. We assume the client has permission to access any record in the database.

Our attacker is the server and has full access to all data in the database as well as all queries issued. Given a query, the goal of the attacker is to determine which user a specific query is about. The server is honest-but-curious, meaning that it does not interfere with the correct operation of the database. Our assumption is that an active attacker (who alters queries, their results, or the database) would eventually be detected and the client would stop using their services. Therefore, it is in the best interest of the server to operate correctly while it attempts to learn private information.

2 Data Privacy of the Query

As a straightforward solution to this problem we propose that instead of performing queries based on individually identifying information, queries are performed on entire groups.

Returning again to the database from Fig. 2, instead of performing a query such as:

```
   SELECT * from DB where SSN="000-03-3060" and Disease = "Hepatitis
A";
```

The client would instead send the following:

```
   SELECT * from DB where GID=2 and Disease = "Hepatitis A";
```

This assumes that the individual identified by 000-03-3060 is in group 2; ways the client can efficiently learn this without violating privacy will be described in Sections 3.2 and 4.2. The client will receive back database entries where `Disease = Hepatitis A` for all users in group 2. The client then simply filters out entries for all users except the one it intended to query. This process does not need to be done manually. A simple query processing tool that runs at the client can make this process transparent. In order to perform this query the client must already know which group the SSN is in. How the client can learn this without causing a privacy violation will be discussed in Section 3.

We will now prove that performing group based queries can have the same privacy guarantees as the underlying grouping methodology.

2.1 Definitions and Notations

Throughout the paper, a table T has d identifier attributes, $A_1, \ldots, A_d$, and a sensitive attribute A_s. (This could easily be extended to multiple sensitive attributes, we use a single one for clarity.) We will use dot notation to refer to some attribute of a tuple (e.g., for a tuple $t \in T$, $t.A_i$ denotes t's value for the corresponding attribute where $1 \leq i \leq d$ or $i = s$).

Our work is based on the k-anonymity family of privacy definitions, which group individuals such that each individual is indistinguishable from others in the group with respect to the sensitive value that goes with each individual.

Definition 1 (Group/Equivalence class). *A group (also known as equivalence class) G_j is a subset of tuples in table T such that $T = \bigcup_{j=1}^{m} G_j$, and for any pair (G_{j_1}, G_{j_2}), where $1 \leq j_1 \neq j_2 \leq m$, $G_{j_1} \cap G_{j_2} = \emptyset$.*

Definition 2 (k-anonymity). *A set of groups is said to satisfy k-**anonymity**, iff $\forall$ groups G_j,*

$$|G_j| \geq k$$

where $|G_j|$ is the number of tuples in G_j.

While much of this work (particularly in this section) applies to any k-anonymity based model, our examples are based on the anatomy definition used in [6], which is a variation of that given in [8].

Definition 3 (Anatomy). *Given a table T partitioned into m groups using k-anonymity without generalization, anatomy produces an identifier table (IT) and a sensitive table (ST) as follows. IT has schema*

$$(A_1, \ldots, A_d, GID, SEQ)$$

where $A_i \in Q_T$ for $1 \leq i \leq d = |Q_T|$, Q_T is the set of identifying attributes in T, GID is the group id and SEQ is the unique sequence number for a tuple. For each $G_j \in T$ and each tuple $t \in G_j$, IT has a tuple of the form:

$$(t.A_1, \ldots, t.A_d, j, seq)$$

The ST has schema

$$(HSEQ, GID, A_s)$$

where A_s is the sensitive attribute in T, GID is the group id and HSEQ contains the output of a keyed cryptographic hash function denoted by $\mathrm{H}_{\bar{k}}(seq)$ where seq is the corresponding unique sequence number in IT for a tuple. For each $G_j \in T$ and each tuple $t \in G_j$, ST contains a sensitive value v in a tuple of the form:

$$(\mathrm{H}_{\bar{k}}(seq), j, v)$$

The key issue with the Anatomy model is that actual data values are preserved; the anonymization occurs by generalizing the link between identifying and sensitive values to the group level. Thus we expect user queries to be based on specific (rather than group level) values. This could communicate user knowledge about relationships between individuals and sensitive data to the server; a query that could convey such knowledge is deemed *sensitive*. Our goal is to preserve the privacy guarantees enforced on the underlying data even after a sequence of queries from a user with knowledge about the data that would violate privacy if revealed to the server.

Definition 4 (Query Privacy). *Any sequence of queries, $Q = < q_1, ..., q_j >$, preserves privacy of individuals if for every tuple $t \in IT$ and for every $v \in ST$ where $v.GID = t.GID$*

$$Pr(t \rightarrow v|T^*) = Pr(t \rightarrow v|T^*, Q)$$

where T^* is an $\{IT, ST\}$ anatomized table pair as in Definition 3 and $t \rightarrow v$ means that v is the sensitive value corresponding to t.

This definition states that a sequence of queries does not change the server's knowledge of the mapping between any individual and a sensitive value. While we do not formally prove it here, we claim that this is sufficient (although perhaps not necessary) to maintain the privacy guarantees of the k-anonymity family of measures. (While there are some special cases where this is not true, e.g,, data meeting k-anonymity with all sensitive values in the group being the same ($Pr = 1$) could meet definition 4 by maintaining the same probability while disclosing information that reduces the group size, we feel such cases reflect failure of the privacy metric to adequately protect sensitive information rather than a failure of query privacy.)

2.2 Query Privacy Preservation

Query streams that contain only information about the identifying attributes, or only about the sensitive attribute, clearly do not change the probability of the mapping and thus satisfy Definition 4. The problem is with queries that affect both:

Definition 5 (Sensitive Query). *A sensitive query, denoted by q, is a selection query in the form*

```
SELECT * FROM ⟨IT,ST⟩[3] WHERE P_IT and P_ST;
```

where P_{IT} is a predicate uniquely identifying one or more individuals in IT and P_{ST} restricts the range of sensitive values from ST.

To avoid revealing information, we require that at least one side of the sensitive query (either the identifying or sensitive information) not distinguish between any items in the group:

Definition 6 (k-anonymized Query). *Given a sensitive query, q, as in Definition 5, a k-anonymized sensitive query, denoted by q^*, is a selection query in either the form*

```
SELECT * FROM ⟨IT,ST⟩ WHERE P*_IT and P_ST;
```

or the form

```
SELECT * FROM ⟨IT,ST⟩ WHERE P_IT and P*_ST;
```

[3] This is not a join operation, it is the selection query described in [6] which is semantically equal to
`SELECT * FROM IT*, ST WHERE P_IT and P_ST` and involves client-server interaction

where $\mathtt{P}^*_{IT}$ is a predicate identifying a group in IT, $\mathtt{P}^*_{ST}$ is a predicate identifying a group in ST, (i.e., each $t' \in G_{t.GID}$ satisfies $\mathtt{P}^*_{IT}$ or each $v' \in G_{v.GID}$ satisfies $\mathtt{P}^*_{ST}$)

We now show that a stream of k-anonymized queries satisfies Definition 4. We show that if each single query satisfies Definition 4, any pair of queries that the groups queried are disjoint or the same, and if the sequence of queries groups either entirely in IT or entirely in ST, then the sequence of queries satisfies Definition 4.

Lemma 1. *Given a sequence of queries $Q =< q_1, ..., q_n >$ where $\forall i$, q_i satisfies Definition 4 and either*
$\forall i, j$: ${P_i}^*_{IT} = {P_j}^*_{IT}$ or ${P_i}^*_{IT} \bigcap {P_j}^*_{IT} = \emptyset$ or
$\forall i, j$: ${P_i}^*_{ST} = {P_j}^*_{ST}$ or ${P_i}^*_{ST} \bigcap {P_j}^*_{ST} = \emptyset$,
Q satisfies Definition 4.

Proof (By Induction). Base case: With only one query, by the preconditions of the lemma the query satisfies Definition 4.

Inductive case: Assume $Q' =< q_1, ..., q_{n-1} >$ satisfies Definition 4. Then $\forall$ individuals I, $Pr(t \to v|T^*) = Pr(t \to v|T^*, Q')$. Divide Q into two sets Q_d and Q_m, where Q_d consists of queries that have an empty intersection with q_n (${P_i}^*_{\mathtt{IT}} \bigcap {P_n}^*_{\mathtt{IT}} = \emptyset$), and Q_m consists of queries that exactly match q_n (${P_i}^*_{\mathtt{IT}} = {P_n}^*_{\mathtt{IT}}$). Definition 4 must hold for both Q_d and Q_m.

First, Q_d and Q_m each satisfy Definition 4, since we could have a query sequence consisting only of disjoint or only of matching queries (which by the inductive hypothesis we assume would satisfy the lemma.) Now we show that adding q_n still satisfies Definition 4.

For Q_m: For every individual $t \notin {P_n}^*_{\mathtt{IT}}$, then neither Q_m or q_n gives any information about t, and $Pr(t \to v|T^*) = Pr(t \to v|T^*, Q_m + q_n)$. For every $t \in {P_n}^*_{\mathtt{IT}}$, the information obtained from Q_m and q_n is exactly the same for all t, and $Pr(t \to v|T^*, Q_m + q_n) = Pr(t \to v|T^*, Q_m) = Pr(t \to v|T^*, q_n) = Pr(t \to v|T^*)$.

For Q_d: For an individual $t \in {P_n}^*_{\mathtt{IT}}$, no information is obtained from Q_d, and $Pr(t \to v|T^*, Q_d + q_n) = Pr(t \to v|T^*, q_n) = Pr(t \to v|T^*)$. Likewise, for $t \notin {P_n}^*_{\mathtt{IT}}$, $Pr(t \to v|T^*, Q_d + q_n) = Pr(t \to v|T^*, Q_d) = Pr(t \to v|T^*)$.

Extending this argument to Q_d and Q_m allows us to combine them, giving $Pr(t \to v|T^*, Q_d + Q_m) = Pr(t \to v|T^*, Q_d) = Pr(t \to v|T^*, Q_m) = Pr(t \to v|T^*)$.

The same argument holds if the group-level information is about the sensitive rather than identifying information ($\mathtt{P}^*_{\mathtt{ST}}$).

Theorem 1. *Transforming a sequence of sensitive queries, $Q = q_1, \ldots, q_n$, into a sequence of k-anonymized queries, $Q^* = q_1^*, \ldots, q_n^*$, protects the privacy of individuals based on k-anonymity and Definition 4.*

Proof. Let q be a k-anonymized query, ($\mathtt{P}^*_{\mathtt{IT}}$) be the group-level identifying information for q, and $v =\mathtt{P}_{\mathtt{ST}}$ be the sensitive value in the query. Let S be the multiset of sensitive values for the group $\mathtt{P}^*_{\mathtt{IT}}$ in T^*.

First, if $t \notin \mathtt{P}^*_{\mathtt{IT}}$, then the query discloses no information about t, and $Pr(t \rightarrow v|T^*, q) = Pr(t \rightarrow v|T^*)$.

If $v \notin S$, then $Pr(t \rightarrow v|T^*, q) = 0 = Pr(t \rightarrow v|T^*)$.

Finally, assume $v \in S$. We assume that the server/adversary has no reason to assume a particular $t \in \mathtt{P}^*_{\mathtt{IT}}$ is being queried, and that any mapping is equally likely. Therefore $Pr(t \rightarrow v|q) = 1/|\mathtt{P}^*_{\mathtt{IT}}|$, the same as $Pr(t \rightarrow v|T^*)$ (note that we are interpreting v as the particular instance of a value in a multiset; if there are multiple occurrences of $v \in S$, then we need to multiply both sides by the number of instances.)

Thus Definition 4 holds for q. By Lemma 1, the Theorem holds.

(Note that this theorem does not hold if the adversary has knowledge of the probability that $t \rightarrow v$ beyond that contained in the query stream and the dataset. Such background information raises problems with the underlying static data under many anonymization models, and is not considered here.)

3 A Basic Solution

At a high-level, simply querying entire groups is a straightforward and simple solution. There is a complication with it, however, that must be addressed: It is not clear how the client can determine which group a given user is in. The client may know the lookup key for the user, but there is not a straightforward way to translate that into a group. In addition, the client cannot request the group number for a given lookup key from the server, as this would leak which user the client is going to later request. It is also not reasonable for the client to store (or request) the entire mapping of lookup keys to groups, as part of the purpose of outsourcing a database is that you no longer need to maintain a local database.

3.1 Group Membership Constraint

There is an important constraint that must be discussed with respect to group membership in this model: Once a group is formed, the membership of that group cannot be changed without potentially leaking private information to a server that is performing a statistical analysis of which groups are queried. For example, assume that group 5 is being frequently accessed, and as such is somewhat of a hotspot. If a member of that group is removed, and the frequent queries stop, then the server can ascertain that the removed entity was the target of most of those queries. The same argument can be used in reverse to describe why a member can never be added to a group. (For further discussion of these issues in the context of INSERTs and UPDATEs, see [7].)

3.2 Solution Overview

We propose adding a separate translation table at the server that can be queried to determine the bucket for a specific lookup key. It is crucial, however, that

SSN	*Name*	*GID*	*SEQ*
000-07-7083	Luis	1	1
000-26-9073	Donna	1	2
000-03-3060	Zachary	2	3
000-04-4396	Kenneth	2	4
000-09-4349	Michelle	3	5
000-22-6531	Thomas	3	6

(a) Identifier Table (IT)

HSEQ	*GID*	*Disease*
$H_{k_1}(1)$	1	HIV
$H_{k_1}(2)$	1	Diabetes
$H_{k_1}(3)$	2	Hepatitis A
$H_{k_1}(4)$	2	Cancer
$H_{k_1}(5)$	3	Tuberculosis
$H_{k_1}(6)$	3	Hepatitis B

(b) Sensitive Table (ST)

Hash	*GID*
H_{K_L}(000-07-7083)	1
H_{K_L}(000-26-9073)	1
H_{K_L}(000-03-3060)	2
H_{K_L}(000-04-4396)	2
H_{K_L}(000-09-4349)	3
H_{K_L}(000-22-6531)	3

(c) Lookup Table (LT)

Fig. 3: Sample Anatomized Database With a Lookup Table

this operation does not reveal which lookup key is being queried. In order to accomplish this, the lookup table will store a keyed hash of the lookup key as well as the bucket that lookup key is in. The value of the key for the keyed hash is not known to the server, but is known to all clients that access the data.

The database needs to be initialized before sending it to the cloud provider:

1. Distribute entries into groups as is done in anatomization. The groupings should provide the group privacy protection (k-anonymity, l-diversity, etc.) that is desired. For the purpose of presentation, we assume that the groupings chosen are identical to the anatomization groupings, but they are not required to be.
2. Choose a random cryptographic key K_L.
3. Create a new table that maps H_{K_L}(Lookup Key) to the corresponding group for that entry. (With $H()$ being a keyed, cryptographic hash function.) See Fig. 3 for an example.

3.3 Operations

The following basic database operations can be supported as follows:

– *Select:* The client queries based on the hash of the lookup key instead of on the lookup key itself:
  `SELECT * from DB where idhash=`H_{K_L}(“000-03-3060”) `and glucose > 250;`
  The server then uses the value of `idhash` to determine the correct bucket from the lookup table and return all relevant results from that group.

– *Insert:* In terms of the data itself, inserts must be batched in groups and inserted with care to ensure the group based privacy guarantees are maintained. In short, tuples to be inserted are not inserted immediately, but are instead temporarily stored in an encrypted cache. Once enough new tuples in are the cache that they can be safely grouped together and added to the database without violating the privacy constraints, then they are inserted into both the anatomized database and the lookup table as an entire batch.
– *Delete:* As we have already described, removing an item from a bucket can potentially leak information. As such, data is not deleted from the tables; instead the (encrypted) join key is modified to show deletion.
– *Update:* Updates involving information other than the lookup key can simply be processed as is. However, it is important to note that during an update the server knows the identity of the user or users being updated. (As long as the server does not know the old or new value of the sensitive data, privacy is not violated.) Updating the lookup key requires generating a new K_L and completely refreshing the lookup table, which requires downloading and then re-uploading it. Due to the overhead of this approach, it is recommended these types of updates be batched or simply not permitted.

Further information on insert/delete/update can be found in Nergiz *et al.* [7]. While that paper does not discuss private queries or the hashing approach presented here, an extension of the solutions presented for regenerating the hash table are straightforward.

4 Known-Query Attack

Under this model, the identity of the user being queried is protected by the keyed hash. However, some information is still indirectly leaked. If the same user is constantly queried, then the same entry in the bucket lookup table will be referenced. The server won't know which lookup key is being accessed, but it will know that the same lookup key is being referenced repeatedly. Under the standard privacy definitions used thus far, *this is not considered a privacy leak.* However, with a small amount of outside information, an attacker could completely compromise all past and present queries for a given user.

Assume that our attacker, in addition to monitoring the database at the cloud provider, also has the ability to learn the original form of one query. We call this a *known-query attack.* For example, if we are storing medical information the attacker might observe someone visiting the hospital and correlate the timing of their visit with a database query made. From this information, the attacker could know which user a specific H_{K_L}(SSN) is associated with. This means that any future (or past, if they were logged) queries about this user can be individually identified by the attacker.

4.1 Oblivious Lookups

In order to prevent this information leakage, it must be ensured that different queries to the lookup table for the same individual are indistinguishable from lookups to other individuals in the same group. (We are only concerned with making it indistinguishable at the group level because the result of the query will ultimately reveal the group anyway.)

The classic approach to hiding the pattern of access to data is the oblivious RAM simulation [9–11]. Under oblivious RAM, a client performs a series of accesses to a RAM that is monitored by an attacker, but the client does not reveal which data she was interested in. A related concept is oblivious storage [12, 13], which is an adaptation of oblivious RAM techniques to make use of the primitives provided by cloud database providers.

As an inefficient solution to this problem, one could apply the simplest oblivious transfer technique and simply download the entire lookup table and query it locally. In this scenario, the server doesn't know which entry a client queried because the entire lookup table is downloaded every time. The problem, of course, is that every lookup to the table requires downloading it in its entirety. This would make the efficiency for a single lookup $O(N)$, where N is the number of individuals in the lookup table. This is unacceptable.

This overhead can be greatly reduced by making use of oblivious storage techniques. In [13], a method of oblivious storage is provided for the Amazon S3 [14] API. Their work is applicable to a variety of database models. Below we describe a method drawn from their work that satisfies our requirements.

4.2 Oblivious Storage Solution

As a solution to the known-query attack described above, we propose making use of the simple, square-root, miss-intolerant oblivious storage solution found in Goodrich et. al. [13]. In order to make use of this solution we must make the following assumptions:

1. There are N individuals to be stored in the lookup table.
2. The lookup table will contain $N + \sqrt{N}$ items.
3. The client performing the lookup has $2\sqrt{N}$ local storage space.
4. The client and server can exchange $\sqrt{N}$ items in one lookup. (For example, by the client issuing a range query.)
5. The client will only lookup an item that exists in the database. (The database lookups are miss-intolerant.)

While the details of the construction can be found in the original work, a brief summary is provided here. First, the lookup keys themselves (here the SSNs) are hashed using a key and a random nonce chosen by the client. Next, the values associated with the lookup keys (in this case the GIDs) are encrypted with a probabilistic encryption scheme which also includes a random nonce chosen by the client. (Such as $E(r||GID)$.) Note this usage of encryption does not violate our original goal of storing unencrypted data, as only the lookup table is encrypted while the original, anatomized data is not. The client also maintains a local cache of size $\sqrt{N}$ that stores items it has recently accessed. Initially, this cache is empty.

To perform a general lookup for a specific identifier S, the client:

Items in Lookup Table	**Server Storage**	**Client Storage**	**Amortized Accesses per Lookup**
10,000 items	10,014 items	27 items	13 accesses
100,000 items	100,017 items	34 items	13 accesses
1,000,000 items	1,000,020 items	40 items	13 accesses

Table 1: Real Values for Oblivious Storage Applied to the Lookup Table

1. Looks for S in its local cache. If it fails to find it there, it queries for S in the encrypted lookup table by searching for the keyed-hash value of it. The server returns the entry.
2. Requests that the server delete S from the lookup table.
3. Adds S to the local cache.
4. Once $\sqrt{N}$ items have been retrieved from the server, then the cache will be full. The client then obliviously shuffles all items in the cache and the lookup table, and also re-encrypts every item with a new random nonce. In this way the entire table can be shuffled without the server being able to tell which items are which.

As can be seen from this description, most lookups will require $O(1)$ database accesses. However, after the local cache is full then the client must reshuffle the entire lookup table, which requires $O(N/\sqrt{N})$ databases accesses. If we amortize these accesses, then it turns out that the amortized lookup time is $O(1)$.

There are some details missing from this description regarding what to do when a lookup is found in the cache, exactly how to perform the oblivious lookup using the client's limited memory, and a proof of the performance just described. This information can be found in the original paper.

In order to give an idea of what this performance would look like in practice, in Table 1 we present some real numbers based on this technique.

5 Related Work

The problem of query privacy has been most deeply studied with research on Private Information Retrieval (PIR) [5]. The goal with PIR is perfect confidentiality - no information is revealed about the query or what it returns. This results in high computational complexity (order of the size of the database for a single server, although there are some better results assuming non-colluding servers [5] or with quadratic preprocessing [15]). Our setting has somewhat different privacy constraints – it is not the privacy of the query that concerns us, but the privacy of the subjects in the data. Information disclosure from the query is only an issue if it leaks information violating the privacy of the data subjects. This allows us to avoid the impractical computational constraints imposed by PIR.

Closer to our model is Paulet et al. [16], where oblivious transfer is used to provide a limited form of k-anonymity for a query as well as to prevent the client from accessing records it should not. Oblivious transfer is used to guarantee the client only accesses 1 record out of k. Their technique, however, relies on the client requesting the record of interest as well as $k-1$ other *random* records. This provides k-anonymity for a single query, but a statistical attack performed by the server over multiple queries will be able to infer information.

Another related area is encrypted database. The seminal work in this area by Hacıgümüş et al. [17] follows an approach in that queries contain only a hashed value at the granularity of an entire block. Theorem 1 shows that this is sufficient to maintain the privacy constraints guaranteed by the underlying data model (in the case of [17], connecting any information at the block level only.) It is an interesting question how this model relates to the anonymization-based models we target – what capabilities and background knowledge (e.g., identity of a querier) would an adversary need to go from obtaining an encrypted block to being able to discern something about the values in that block? However, such a comparison is beyond the scope of this paper. Popa et al. [18] allow querying a fully encrypted database. Their work is focused on protecting the confidentiality of the data in the database, but the queries may be susceptible to a weaker version of the known-query attack described in Section 4. Future work in encrypted database, however, could focus on protecting the query as well and may be able to achieve many of the same goals as this work.

While we make use of research in the area of oblivious RAM and oblivious storage to hide which entry in our lookup table is being accessed, one could ask why oblivious RAM (o-ram) is not applied for all queries to begin with. While these techniques seem like an obvious solution to original problem in this work, there are a few reasons it is infeasible. First, o-ram requires the data being protected to be encrypted. As discussed in Section 1, in our scenario we assume an unencrypted database so that a cloud provider can provide a variety of services or allow unrestricted queries on non-sensitive data. (There do exist some o-ram schemes that do not rely on cryptography [9]; however, the efficiency is significantly worse than their cryptographic counterparts.) Another issue with applying o-ram to this scenario is the performance of such systems is still very low. Even the most efficient form of the algorithm currently known [11] has an $O((\log N)^2)$ amortized cost of with a $O((\log N)^3)$ worst-case cost.

Farnan et al. [19] addresses the issue of sensitive queries in a decentralized database by providing a way to specify privacy constraints as part of the SQL query. Their work is primarily concerned with ensuring that the various, decentralized databases involved in servicing a query not be aware of what information is being queried from each other. This differs from our centralized model, but still illustrates the importance of focusing on privacy leakage related to queries.

Most anonymization work sidesteps the issue of query privacy entirely. The use case of anonymization is traditionally privacy-preserving data publishing; the client will obtain a copy of the anonymized data, and thus queries will not be revealed to the server. In practice, Public Use Microdata Sets [20, 21] are often accessed through a query interface, but the server is presumed to be controlled by the agency holding the original data, so queries that enable the server to infer private information are only disclosing data already known to the server. With the rise in data outsourcing, it will be interesting to study if techniques such as the one presented in this paper will be necessary for other anonymization use cases where the agency holding the original data outsources the hosting and query processing to an external entity.

6 Conclusion

We have shown that given an anatomized database that meets a privacy constraint, the same constraint can still hold in the face of queries as long as those queries are performed at the group level. The complication in applying this result is to ensure that the client can determine the group a specific user is in without querying the server to ask. To solve this problem, we include a keyed hash based lookup table which can be used to determine which group an individual is located in. To provide even further privacy protection in the face of a known-query attack, oblivious storage is used to further protect the lookup table.

Future work should explore using a more robust oblivious storage technique that better supports multiple clients, applying these techniques to a more general data protection model such as fragmentation[22], investigating supporting any column as a potential lookup key, and expanding support to include both update and delete operations.

7 Acknowledgments

Special thanks to Erhan Nergiz for his assistance with the proofs in this paper. This publication was made possible by the support of the NPRP grant 09-256-1-046 from the Qatar National Research Fund. The statements made herein are solely the responsibility of the authors.

References

1. Samarati, P.: Protecting respondents identities in microdata release. IEEE Transactions on Knowledge and Data Engineering **13**(6) (2001) 1010–1027
2. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty Fuzziness and Knowledge Based Systems **10**(5) (2002) 557–570
3. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) **1**(1) (2007) 3
4. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007., IEEE (2007) 106–115
5. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM **45**(6) (1998) 965–981
6. Nergiz, A., Clifton, C.: Query processing in private data outsourcing using anonymization. In Li, Y., ed.: Data and Applications Security and Privacy XXV. Volume 6818 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2011) 138–153

7. Nergiz, A.E., Clifton, C., Malluhi, Q.M.: Updating outsourced anatomized private databases. In: Proceedings of the 16th International Conference on Extending Database Technology. EDBT '13, New York, NY, USA, ACM (2013) 179–190
8. Xiao, X., Tao, Y.: Anatomy: simple and effective privacy preservation. In: Proceedings of the 32nd International Conf. on Very Large Data Bases. (2006) 139–150
9. Ajtai, M.: Oblivious rams without cryptogrpahic assumptions. In: Proceedings of the 42nd ACM symposium on Theory of computing. STOC '10, New York, NY, USA, ACM (2010) 181–190
10. Pinkas, B., Reinman, T.: Oblivious ram revisited. Advances in Cryptology (CRYPTO) (2010) 502–519
11. Shi, E., Chan, T., Stefanov, E., Li, M.: Oblivious ram with o ((log n) 3) worst-case cost. Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT) (2011) 197–214
12. Boneh, D., Mazieres, D., Popa, R.A.: Remote Oblivious Storage: Making Oblivious RAM Practical. Technical Report MIT-CSAIL-TR-2011-018, Computer Science and Aritificial Intelligence Laboratory (March 2011)
13. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Practical oblivious storage. In: Proceedings of the Second ACM Conference on Data and Application Security and Privacy. (2012) 13–24
14. Amazon: Amazon Simple Storage Service (S3) `http://aws.amazon.com/documentation/s3/`.
15. Asonov, D., Freytag, J.C.: Almost optimal private information retrieval. In: Second International Workshop on Privacy Enhancing Technologies PET 2002, San Francisco, CA, USA, Springer-Verlag (April 14-15 2002) 209–223
16. Russell Paulet, M., Kaosar, G., Yi, X.: K-anonymous private query based on blind signature and oblivious transfer. In: 2nd International Cyber Resilience Conference. (2011) 55–62
17. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data. SIGMOD '02, New York, NY, USA, ACM (2002) 216–227
18. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM (2011) 85–100
19. Farnan, N.L., Lee, A.J., Chrysanthis, P.K., Yu, T.: Dont reveal my intension: Protecting user privacy using declarative preferences during distributed query processing. In Atluri, V., Diaz, C., eds.: Computer Security ESORICS 2011. Volume 6879 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 628–647
20. Moore, Jr., R.A.: Controlled data-swapping techniques for masking public use microdata sets. Statistical Research Division Report Series RR 96-04, U.S. Bureau of the Census, Washington, DC. (1996)
21. Subcommittee on Disclosure Limitation Methodology, Federal Committee on Statistical Methodology: Report on statistical disclosure limitation methodology. Statistical Policy Working Paper 22 (NTIS PB94-16530), Statistical Policy Office, Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC. (May 1994)
22. Ciriani, V., di Vimercati, S.D.C., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC) **13**(3) (July 2010) 22:1–22:33