# Updating Outsourced Anatomized Private Databases

Ahmet Erhan Nergiz
Department of Computer Science
Purdue University
West Lafayette, Indiana
anergiz@cs.purdue.edu

Chris Clifton
Department of Computer Science / CERIAS
Purdue University
West Lafayette, Indiana
clifton@cs.purdue.edu

Qutaibah M. Malluhi
Department of Computer Science and Engineering
Qatar University
Doha, Qatar
qmalluhi@qu.edu.qa

## ABSTRACT

We introduce operations to safely update an anatomized database. The result is a database where the view of the server satisfies standards such as $k$-anonymity or $l$-diversity, but the client is able to query and modify the original data. By exposing data where possible, the server can perform value-added services such as data analysis not possible with fully encrypted data, while still being unable to violate privacy constraints. Update is a key challenge with this model; naïve application of insertion and deletion operations reveals the actual data to the server. This paper shows how data can be safely inserted, deleted, and updated. The key ideas are that data is inserted or updated into an encrypted temporary table until enough data is available to safely decrypt, and that sensitive information of deleted tuples is left behind to ensure privacy of both deleted and undeleted individuals. This approach is proven effective in maintaining the privacy constraint against an adversarial server. The paper also gives empirical results on how much data remains encrypted, and the resulting quality of the server's (anatomized) view of the data, for various update and delete rates.

## Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*

## 1. INTRODUCTION

Data outsourcing is a growing business. Cloud computing developments such as Amazon Relational Database Service promise further reduced cost. However, use of such a service can be constrained by privacy laws, requiring specialized service agreements and data protection that could reduce economies of scale and dramatically increase costs.

Most privacy laws apply to data "relating to an identified or identifiable natural person"[10], data that cannot be directly or indirectly linked to an individual is not restricted. Some laws are even more specific; the U.S. Healthcare laws apply only to identifiable *health information*[14]. A private data outsourcing approach proposed in [20] encrypts the link between identifying information and sensitive (protected) information, only the client has the key to decrypt this link. As the server no longer has access to *individually identifiable* protected information, it is not subject to privacy laws, and can offer a service that does not need to be customized to the needs of each country- or sector-specific requirements; any risk of violating privacy through releasing sensitive information tied to an individual remains with the client.

The key idea behind [20] is that the server stores data using the model of anatomy[30] (or fragmentation[8] – we will use the term anatomy, as it does not have other meanings in the database literature.) A tuple containing private information is split between two tables, one containing identifying information, and the other containing sensitive information. The server can join these tables at the group level (as with anatomy); in addition, an actual sequence number is stored with each tuple, but encrypted in one table so that only the client (who has the key) can join actual values. An example is given in Figure 1. This supports a variety of privacy models, including $k$-anonymity[24, 26], $l$-diversity/discernibility[19, 22], and $t$-closeness[18]. This paper will use the privacy principle underlying $l$-diversity: the server should be able to determine the sensitive value for an individual with probability at most $1/l$.

The work in [20] provides safe and efficient means to support read queries (select/project/join/group by), but does not discuss how to handle insert/update/delete. One way to tackle this problem is to update the data in batches; while there has been work in this area [7, 31, 5, 27, 23, 11, 29, 13], it is based on a model where the data owner stores all of the original data, and releases updated anonymized datasets periodically. While these are valuable approaches, they do not support the database outsourcing model of [20]:

1. Previous incremental anonymization work assumes the anonymizer has access to the entire original dataset. In database outsourcing, requiring the client to keep the entire dataset defeats the purpose of outsourcing.

2. Batch update may not be feasible. Data may need to be available to other clients, and must be stored at the server to ensure persistence.

We address issue 2 by keeping an encrypted table of inserted/updated data at the server until there is sufficient data to form a group satisfying the privacy constraint. Deletion removes only the identifying information for the tuple; the sensitive value is left behind to avoid disclosing the sensitive data for the deleted individual and to ensure the $1/l$ constraint is met for the remaining individuals. There are further issues that complicate this basic approach. For example, even though the data is encrypted, the server learns when the (encrypted) data is inserted. In addition, the server knows the old value for updated tuples, making it more difficult to protect the new value. We give an example that further demonstrates these issues in Section 1.3, after we have given some additional background.

Our dynamic private data outsourcing model addresses all these issues, with insert, update, and delete operations that

- provably preserve the privacy of an individual under $l$-diversity,
- provide iterative anatomization,
- require data storage only at the server, and
- reduce client processing for anatomization.

We now expand on the threat model, and elaborate on related work. We provide definitions and notations for an anatomized database in Section 2. In Section 3, we show how insert, delete and update operations can be performed with low client-side cost, and prove that the privacy constraint is satisfied. Section 4 shows how to modify the query processing of [20] to work in this model. We give empirical results showing how rapidly data is decrypted (and other impacts on the quality of the server's anatomized view of the data) in Section 5. Section 6 concludes our paper with possible extensions for this private data outsourcing model.

## 1.1 Assumptions and Threat Model

Throughout the paper, we assume "server" is the data outsourcing service provider and "client" is the data owner.[1] The client modifies and queries the data stored in the server, and occasionally does housekeeping on the data. The outsourced data has finite domain where attributes are either strings or integers. The client(s) in our model are lightweight entities that have limited processing power and do not do any data storage except for the key (e.g., mobile devices). We assume that clients do not collude with the server, as any client with access to the database is assumed to have a key enabling full reconstruction of the data.

We assume the server has a directory enabling it to link unencrypted identifying information in the database to the actual individual. For updated data, we assume the server can track a record even if it is encrypted (and knows the individual it applies to), so encryption alone would not protect the old sensitive value.

We do not protect data *integrity* against a malicious adversary. Our only concern is that through analysis of the data, and of interactions with the client(s), the server could infer information that would violate privacy policy. As such, we provide *privacy* against a malicious adversary. We assume that an adversary cannot generate fake tuples; this could be detected through a (keyed) checksum of each tuple and the uniqueness of the sequence number and group ID (included in the checksum). Thus the only undetectable malicious behavior on the part of the adversary is to fail to return tuples. As this would be equivalent to a sequence of deletes/inserts causing the returned values to be the correct answer, and the method protects privacy given an arbitrary sequence of inserts/deletes/updates, the proofs given also hold against such an adversary. For clarity of presentation, we do not include the above details in the operations and algorithms, and only give proofs for a semi-honest adversary. In practice, the above approaches extend this to protection against a rational adversary (as the server would be out of business if it were discovered failing to correctly store data and return query results); existing research on methods ensuring integrity of outsourced data could also be applied.

[1]When there are multiple clients for a given database, all share the decryption key. Access control is managed by the server; this does raise some challenges that are left for future work.

## 1.2 Related Work

Private data outsourcing, also known as the database-as-a-service (DAS) model, was introduced by Hacigumus et al. [12]. They used bucketization over an encrypted database to enable the server to partially execute queries. In [15] this is extended to range queries. Damiani et al. [9] proposed another technique that uses hashing for bucketization and encrypted B+ trees for indexing, and introduce an aggregate metric for data disclosure. However, an aggregate metric fails to ensure the privacy of each individual.

Another approach is using searchable encryption for range queries [25, 4, 2]. This only supports search; queries such as group-by require processing all data at the client (including expensive decryption). Kantarcioglu et al.[16] show that an efficient private data outsourcing scheme based on encryption cannot be proven to meet cryptography-style definitions. Instead, they outline an efficient private data outsourcing scheme using encryption with provable security that uses tamper-resistance hardware at the server.

Aggarwal et al. [1] proposed vertical fragmentation instead of encryption, specifically to hide functional dependencies from an adversary. They require two non-colluding servers. Since finding non-colluding servers may be impractical, Ciriani et al. [8] proposed to vertically fragment a table and encrypt a portion of the data. This enables queries involving only the unencrypted data to be executed at the server. In this method the size of the database is $m$ times the original size where $m$ is the number of fragments, and results in complicated queries when both encrypted and plaintext attributes are used. As with the model we use, the client needs to decrypt the outcome and further process to get the final result. They also suggest fragmenting the tables and storing a small partition with sensitive values at the client. They prove that finding the optimal partitioning is NP-hard.

Continuous data publishing has been widely studied over the years. This is an orthogonal problem to data outsourcing, but relevant to this work as in a sense we view data as being "published" at the server. For a variety of reasons, continuous data publishing does not address the data outsourcing problem. Some work addresses a very different problem, for example, Wang et al. [27] supports releasing data for different subsets of a global set of quasi-identifiers based on k-anonymity, rather than changes to the data. Another approach is discussed in [11] where the release is protected against inference channels due to the existence of identical records across different releases; the primary issue here is not the change to the data, but the change in how data is grouped for anonymization.

Both [23] and [7] propose continues data publishing techniques supporting (only) insertion. [31] supports insert and

| Patient | Age | City | GID | SEQ |
|---|---|---|---|---|
| Ike | 41 | Dayton | 1 | 1 |
| Eric | 22 | Richmond | 1 | 2 |
| Olga | 30 | Lafayette | 2 | 3 |
| Kelly | 35 | Lafayette | 2 | 4 |
| Faye | 24 | Richmond | 3 | 5 |
| Mike | 47 | Richmond | 3 | 6 |
| Jason | 45 | Lafayette | 4 | 7 |
| Max | 31 | Dayton | 4 | 8 |

(a) $\text{Patient}_{\text{IT}}$

| HSEQ | GID | Disease |
|---|---|---|
| $H_{k^h}$ (1) | 1 | Cold |
| $H_{k^h}$ (2) | 1 | Fever |
| $H_{k^h}$ (3) | 2 | Flu |
| $H_{k^h}$ (4) | 2 | Cough |
| $H_{k^h}$ (5) | 3 | Flu |
| $H_{k^h}$ (6) | 3 | Fever |
| $H_{k^h}$ (7) | 4 | Cough |
| $H_{k^h}$ (8) | 4 | Cold |

(b) $\text{Patient}_{\text{ST}}$

Figure 1: 2-diverse Patient Table

| Patient | Age | City | GID | SEQ |
|---|---|---|---|---|
| Ike | 41 | Dayton | 1 | 1 |
| | | | | |
| Olga | 30 | Lafayette | 2 | 3 |
| Kelly | *36* | Lafayette | 2 | 4 |
| Faye | 24 | *Lafayette* | 3 | 5 |
| Mike | 47 | Richmond | 3 | 6 |
| Jason | 45 | Lafayette | 4 | 7 |
| Max | 31 | Dayton | 4 | 8 |
| *Michael* | *25* | *Richmond* | *4* | 9 |

(a) $\text{Patient}_{\text{IT}}$

| HSEQ | GID | Disease |
|---|---|---|
| $H_{k^h}$ (1) | 1 | Cold |
| | | |
| $H_{k^h}$ (3) | 2 | Flu |
| $H_{k^h}$ (4) | 2 | *Fever* |
| $H_{k^h}$ (5) | 3 | Flu |
| $H_{k^h}$ (6) | 3 | Fever |
| $H_{k^h}$ (7) | 4 | Cough |
| $H_{k^h}$ (8) | 4 | Cold |
| $H_{k^h}$ (*9*) | *4* | *Flu* |

(b) $\text{Patient}_{\text{ST}}$

Figure 2: Patient Table after Updates

delete by proposing a new generalization principle; m-invariance. Bu et al.[5] proposes an anonymization approach for serial publishing that considers both changing and permanent sensitive values hence supports also updates. [13] proposes a different type of attack called "value equivalence attack" where knowledge of an individual's sensitive value can reveal sensitive values of other individuals who replace that individual in a specific group in future increments and the authors give a graph-based anonymization algorithm against this type of attack. While the above works provide some solutions for insert, update, and delete; all assume the complete dataset is available. This is not appropriate for data outsourcing; we provide a solution where incremental anonymization requires the client to access only a subset of the data. While we specifically address the model of [20], this work could also be used to develop update methods for the above approaches.

### 1.3 Example

We now demonstrate some of the challenges with update. Figure 1 shows an outsourceable dataset based on the model in [20]. The table after a *delete*, an *insert*, and some *updates* is shown in Figure 2.

Knowing the old and new tables, the server can infer that *Michael* has *Flu*. Deletion reveals not only the sensitive value of the deleted tuple, but also increases the knowledge about the remaining tuples in the group (potentially violating $l$-diversity.) For instance, Eric was deleted, revealing that Eric had a Fever and now Ike has a Cold. When both $\text{Patient}_{\text{IT}}$ and $\text{Patient}_{\text{ST}}$ are simultaneously updated for the same tuple, as with age and disease information for Kelly, it reveals that Kelly had disease Cough but now has disease *Fever*. (This also occurs with an update to $\text{Patient}_{\text{ST}}$ based on a selection on $\text{Patient}_{\text{IT}}$.) Furthermore, patient Olga's disease is known with probability $\leq \dfrac{1}{l-1}$. This problem does not occur when the update operation takes place in a single subtable (e.g., the update to Faye's city does not disclose Faye or Mike's disease.)

To address these, we delay inserts until they can be safely anonymized, see Figure 3. The idea is to cache recent changes in an encrypted temporary table at the server ($\text{Patient}_{\text{I}}$) and

| Patient | Age | City | GID | ESEQ |
|---|---|---|---|---|
| Ike | 41 | Dayton | 1 | $E_k$ $(salt, 1)$ |
| | | | | |
| Olga | 30 | Lafayette | 2 | $E_k$ $(salt, 3)$ |
| | | | | |
| Faye | 24 | *Lafayette* | 3 | $E_k$ $(salt, 5)$ |
| Mike | 47 | Richmond | 3 | $E_k$ $(salt, 6)$ |
| Jason | 45 | Lafayette | 4 | $E_k$ $(salt, 7)$ |
| Max | 31 | Dayton | 4 | $E_k$ $(salt, 8)$ |

(a) $\text{Patient}_{\text{IT}}$

| SEQ | GID | Disease |
|---|---|---|
| 1 | 1 | Cold |
| 2 | 1 | Fever |
| 3 | 2 | Flu |
| 4 | 2 | Cough |
| 5 | 3 | Flu |
| 6 | 3 | Fever |
| 7 | 4 | Cough |
| 8 | 4 | Cold |

(b) $\text{Patient}_{\text{ST}}$

| SEQ | ENC | SS |
|---|---|---|
| 1 | $E_k$ (*salt*, *Michael*, *25*, *Richmond*, *Flu*) | 1 |

(c) $\text{Patient}_{\text{I}}$

| SEQ | Patient | Age | City | ENC | $\mathcal{S}^-$ |
|---|---|---|---|---|---|
| 1 | Kelly | *36* | Lafayette | $E_k$ $(salt, Fever)$ | $\{Flu, Cough\}$ |

(d) $\text{Patient}_{\text{U}}$

Figure 3: Our Model

send it to the client for anatomization when it has enough tuples, the result can then be merged into the published data. (This table must also be sent to the client for decryption/processing on any query.) Since the server could track updated tuples anyway, we disclose identifying information for updates ($\text{Patient}_{\text{U}}$), allowing some query processing to be done at the server. Finally, we delete only the identifying information, $\text{Patient}_{\text{ST}}$ retains "dead" data to protect Ike 's privacy (again, somewhat complicating query processing).

There are still some subtle inference channels remaining. For example, assume the tuples in Figure 3c and 3d are anatomized. Such anatomization creates only one group with sensitive values, *Flu* and *Fever*, and each tuple has a 1/2 probability of having one of these sensitive values. However, the fact that Kelly previously had *Flu* suggests it would not be in the new group, violating 2-diversity. Specifically, let $A$ and $A'$ be the current and previous disease of Kelly. Then,

$$\begin{aligned}
\mathcal{P}\{A = Flu\} =& \mathcal{P}\{A = Flu | A' = Flu\}\mathcal{P}\{A' = Flu\} \\
&+ \mathcal{P}\{A = Flu | A' = Cough\}\mathcal{P}\{A' = Cough\} \\
=& 0 \times 1/2 + 1/2 \times 1/2 = 1/4 \\
\mathcal{P}\{A = Fever\} =& \mathcal{P}\{A = Fever | A' = Flu\}\mathcal{P}\{A' = Flu\} \\
&+ \mathcal{P}\{A = Fever | A' = Cough\}\mathcal{P}\{A' = Cough\} \\
=& 1 \times 1/2 + 1/2 \times 1/2 = 3/4
\end{aligned}$$

Since $\mathcal{P}\{A = Fever\} > 1/2$, the new created group is not 2-diverse. To address this problem, we give a new anatomization algorithm that also considers sensitive values in the previous groups of the updated tuples.

## 2. ANATOMIZED DATA OUTSOURCING

We now present relevant definitions and notations based on the anatomized/encrypted data query processing work of [20], with extensions to support data manipulation.

### 2.1 Definitions and Notations

Definition 2.1 (Person specific table). *A table $T$ is said to be a person specific table for population $P$ if each tuple $t \in T$ corresponds to a unique individual $p \in P$.*

Throughout the paper, person specific table $T$ has $d$ identifier attributes, $A_1, \ldots, A_d$, and a sensitive attribute $A_s$. We will use dot notation to refer to an 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 $i = s$).

Definition 2.2 (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 2.3 ($l$-diversity). *A set of groups is said to be* **l-diverse**, *iff $\forall$ groups $G_j$,*

$$\forall v \in \pi_{A_s} G_j,\ freq(v, G_j)/|G_j| \leq 1/l$$

*where $A_s$ is the sensitive attribute in $T$, $freq(v, G_j)$ is the frequency of $v$ in $G_j$, and $|G_j|$ is the number of tuples in $G_j$.*

We only refer to a single sensitive attribute (or set of attributes that can be treated as a single combined value); having multiple sensitive attributes in a single table raises issues such as correlated attributes beyond the scope of this paper.

Anatomy is slightly redefined from [30, 20] to include the encrypted link between tuples:

Definition 2.4 (Anatomy). *Given a person specific table $T$ partitioned into $m$ equivalence classes using l-diversity without generalization, anatomy produces an identifier table (IT) and a sensitive table (ST) as follows. IT has schema*

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

*where $A_i \in I_T$ for $1 \leq i \leq d = |I_T|$, $I_T$ is the set of identifying attributes in $T$, GID is the group id of the equivalence class and ESEQ is the encryption of a unique sequence number, SEQ. For each $G_j \in T$ and each tuple $t \in G_j$ (with sequence number $s$), IT has a tuple of the form:*

$$(t.A_1, \ldots, t.A_d, j, \mathrm{E}_k(salt, s))$$

*The ST has schema*

$$(SEQ, GID, A_s)$$

*where $A_s$ is the sensitive attribute in $T$, GID is the group id of the equivalence class and SEQ is a unique sequence number for that tuple in ST, used as an input for the ESEQ of the corresponding tuple in IT. For each $G_j \in T$ and each tuple $t \in G_j$, ST has a tuple of the form:*

$$(s, j, t.A_s)$$

We will use $IT^d$ to refer to the $IT$ when $IT.ESEQ$ has been decrypted at the client. Given $IT^d$, $(IT^d)^e$ is $IT$ with different ciphertext in $ESEQ$ due to the random salts.

The difference between this definition and the one in [20] is the technique used to hide the actual link between $IT$ and $ST$. Instead of using a hash function, a semantically secure symmetric key encryption is used to enable non-1:1 mappings in the groups. The server needs to store whether a group has 1:1 mapping or not since this information is used in anatomization and query processing later on.

We now define the two new schema elements needed to support data manipulation. These two tables hold inserted and modified data until sufficient data is available to safely disclose a group.

Definition 2.5 (Temporary Insert Table $I$). *Given a table $T$ having the schema $A_1, \ldots, A_d, A_s$ as in Definition 2.4, table $I$ holds tuples that have been recently inserted. $I$ has three attributes, $\langle S_2, ENC_2, SS \rangle$, where $S_2$ is a sequence number serving as tuple id, $ENC_2$ is the encryption of all the fields of a tuple having $T$'s schema, and $SS$ is the snapshot index indicating when $t$ is inserted into $I$.*

*For each tuple $t \notin IT^d \bowtie ST$ pointing to a unique individual $p \in P$, $I$ has initially a tuple of the form,*

$$(s, \mathrm{E}_k(salt, t.A_1, \ldots, t.A_d, t.A_s), i)$$

(We assume only a single tuple referring to each individual, i.e., the identifying information is a key.)

Note that update cannot be modeled simply as delete followed by insert, as it would be apparent to the server that the newly inserted tuple was probably a modification of the deleted tuple. Given that the server can likely track updates anyway, we expose the identifying information of the updated tuple to the server, and keep further information to ensure privacy constraints are met in spite of this exposed information (this will be explained fully in Sections 3.3 and 3.4.) Now we give a similar definition to the one in [31] for a group's set of sensitive values which is used to ensure privacy constraints for updated tuples.

Definition 2.6 (Signature of a group/tuple). *Let $G$ be a group in IT and ST, the* ***signature*** *of $G$ denoted by $G.\mathcal{S}$ is the set of distinct sensitive values in $G$. Similarly, let a tuple $t$ be in a group $G$; the* ***signature*** *of $t$, $t.\mathcal{S}$, equals to $G.\mathcal{S}$ until $t$ is updated such that $t.A_s \notin G.\mathcal{S}$.*

In light of the findings in [31], $\mathcal{S}_{Ike}$ is $\{Cold, Fever\}$ and it must not change throughout the lifetime of the tuple representing Ike. We also introduce a negative signature, $t.\mathcal{S}^-$, to support updating of a tuple's sensitive value. $t.\mathcal{S}^-$ indicates that a tuple $t$ has been updated such that $t.A_s \notin t.\mathcal{S}$ anymore and $t$ must be in a group $G$ satisfying $G.\mathcal{S} \cap t.\mathcal{S}^- = \emptyset$.

Definition 2.7 (Temporary Update Table, $U$). *Given a table $T$ having the schema $A_1, \ldots, A_d, A_s$ as in Definition 2.4, table $U$ holds tuples that have been updated. $U$ has $d+3$ attributes $\langle S_1, A_1, \ldots, A_d, ENC_1, \mathcal{S}^- \rangle$, where $S_1$ is a sequence number serving as tuple id, $A_1, \ldots, A_d$ are as in Definition 2.4, $ENC_1$ is the encryption of $A_s$ (i.e., sensitive value), and $\mathcal{S}^-$ as defined above restricts the possible $\mathcal{S}$ that $t$'s future group can have.*

*When we insert a tuple $u$ into $U$, where $u$ is an update to tuple $t$, $t$ is removed from IT (but not ST), and a new entry $u$ is made into $U$ when $u.A_s \notin t.\mathcal{S}$ (if $u.A_s \in t.\mathcal{S}$, $u$ can remain in $t.G$ once $u.ESEQ$ is updated to reflect the new sensitive value). The entry into $U$ has the form:*

$$(s, u.A_1, \ldots, u.A_d, \mathrm{E}_k(salt, u.A_s), G.\mathcal{S})$$

*where $s$ is a system generated unique id for tuple $u$ and $G.\mathcal{S}$ is the signature of the group $t$ was in.*

As with $IT^d$, we use $I^d$ and $U^d$ to refer to $I$ and $U$ respectively with the attribute values decrypted at the client. $(I^d)^e$ and $(U^d)^e$ denotes $I$ and $U$ respectively with a different ciphertext in encrypted attributes. After an update operation on a tuple $t$, we use $t^o$ to refer to the old value of $t$ before the update.

It would appear that we could leave identifying information decrypted in $I$ as well as $U$. It turns out this poses additional constraints; proof is omitted due to space constraints. Since the server is assumed to be able to track tuples (and thus can infer which tuples are updated even if encrypted), exposing identifying information in $U$ does not reveal additional information to the server and thus must be made safe; this also imposes restrictions on groups tuples in $U$ can go into. This will be shown in Section 3.4.

A note on encryption. $\mathrm{E}_k(\cdot)$ is a symmetric key encryption (e.g., AES) under key $k$, *salt* is a pseudorandom number, and comma denotes concatenation. Note that these two schemas are for an encryption of one block (e.g., 16 bytes for AES), for larger tuple sizes, cipher-block chaining (CBC) can be used. In that case the schema must also contain the $IV$ used in CBC. There is no need to encrypt values larger than one block as long as the total number of distinct values for every attributes is bounded by one block size (e.g., $2^{8*16}$). A simple numbering for every value eliminates the need to encrypt more than one block.

While the above is sufficient to show how data manipulations are performed, for use in proofs that the server cannot violate privacy based on remembered history we introduce notation for historical values:

Definition 2.8 (Table Snapshot). *The i-th snapshot of table $T$, denoted by $T(i)$, represents the table after exactly $i$ incremental anatomizations. Similarly, $IT$, $ST$, $U$, and $I$ are denoted by $IT(i)$, $ST(i)$, $U(i)$, and $I(i)$ respectively during the i-th incremental anatomization. Moreover, a tuple $t(i)$ in $T(i)$ denotes an individual's tuple during the i-th incremental anatomization and each snapshot of $t$ represents the same individual.*

As is common in the literature, we use $T^*$ to refer to the anatomized version of $T$. In our model, $T^*$ is a set of tables, $\{IT, ST, I, U\}$, rather than a single table. Similarly, $(T^*)^d$ refers to $(IT^d \bowtie ST) \cup I^d \cup U^d$ which yields the table $T$.

## 2.2 Privacy Proof Technique

A final "definition" deals with the method we use to prove that the privacy constraint is maintained. As stated before, we are using the model of $l$-diversity as a privacy constraint. We assume that the initial dataset ($IT$ and $ST$) meets this constraint. We then use a simulation argument, similar to that used in the secure multiparty computation literature, to show that given a database meeting the constraint, the data manipulation operations maintain the constraint. If the information exposure (including history) from a dataset after a set of operations is no greater than that from an $l$-diverse anatomization on the whole dataset, then privacy of individuals has been maintained. The information exposure from the dataset that has been anatomized as a whole is defined as follows:

Definition 2.9 (simple $l$-diverse distribution). *An $l$-diverse group $G$ is said to have a* simple $l$-diverse distribution *if*

$$\forall p \in G \text{ and } \forall v \in \pi_{A_s} G, \mathcal{P}(p.A_s = v) = freq(v, G_j)/|G_j|$$

*where $p$ denotes an individual and $A_s$, $freq(\cdot)$ is as in Definition 2.3.*

A simple $l$-diverse distribution requires that no prior knowledge affecting this distribution exists about the individuals in $G$. Therefore it is crucial to prevent any inference channels that may exist between different snapshots of the dataset since they may cause the distribution of sensitive values in a group to not be simple $l$-diverse. This paper extends the technique of [31] to avoid such inference channels. This technique, based on induction, assumes an initial state where $l$-diversity is satisfied and provides the means to check $l$-diversity of the current state by comparing it only with the previous state. There is no need to store probability distribution for the previous state, since every state has a simple $l$-diverse distribution.

Now we give the privacy definition for our model ( similar to security definitions in cryptography literature):

Definition 2.10 (Privacy). *The proposed private data outsourcing technique is secure if for every probabilistic polynomial-time algorithm $A$ there exists a probabilistic polynomial-time algorithm $A'$ such that for every individual $p \in T(i-1) \cup T(i)$, every group $G \in T^*(i-1) \cup T^*(i)$ where $p \in G$, every pair of polynomially bounded functions $f, h : \{0,1\}^* \to \{0,1\}^*$, every snapshot $T^*(i-1)$ and $T^*(i)$, every transaction history $TH(i)$ between $T^*(i-1)$ and $T^*(i)$*

$$\mathcal{P}\{A(D_{A_s}, T^*(i-1), T^*(i), TH(i), h(G)) = f(D_{A_s}, p.A_s)\}$$
$$= \mathcal{P}\{A'(D_{A_s}, \perp, T^*(i), \perp, h(G)) = f(D_{A_s}, p.A_s)\}$$

*where $D_{A_s}$ is the domain of the sensitive attribute in $T$, $\perp$ in the equation indicates that $T^*(i)$ is the next snapshot after $T(0) = \emptyset$, and all the transaction history is the insertion of the tuples in $T(i)$.*

Instead of using computational indistinguishability as in the cryptography literature, we require our model to produce an exact probability distribution consistent with the ideal model (where anatomization is only done once). This shows that our model provides the same guarantees with the underlying data publishing technique (i.e., $l$-diversity for this paper). Moreover, Definition 2.10 accepts partial information about any individual $p' \neq p$ where $p' \in G$ denoted by function $h$; this lets us show that our model preserves the same privacy as $l$-diversity even when the server knows partial information about an individual's sensitive value.[2]

We assume that the anatomization algorithm is known by the server, assuming a secret algorithm is too strong to be practical based on Kerckhoffs's principle [17]. This provides resilience against attacks such as the minimality attack[28].

Theorem 2.1. *If $\langle IT(1), ST(1)\rangle$ has a simple $l$-diverse distribution in all groups and our model provides privacy as in Definition 2.10 then all possible snapshots $\langle IT(i), ST(i)\rangle$ have simple $l$-diverse distribution where $(1 \leq i \leq \infty)$.*

Proof By Induction. Base: $T^*(1)$ already has simple $l$-diverse distribution according to the theorem.

Induction: $T^*(i)$ without any knowledge of previous snapshot has simple $l$-diverse distribution since anatomization is done based on $l$-diversity. According to Definition 2.10, there is no difference to an adversary between knowing the previous snapshot or not knowing it. Hence $T^*(i)$ has simple $l$-diverse distribution given that $T^*(i-1)$ has simple $l$-diverse distribution. $\square$

## 3. DATA MANIPULATION

We now formally define SQL insert, update, and delete for our model, and prove that these operations do not violate the re-identification privacy principle of $l$-diversity. Note that this only includes insertion into the temporary $I$ and $U$ tables; the decryption of data in those tables and insertion into $IT$ and $ST$ will be discussed in Sections 3.4.

[2]Function $h$ also models the *equivalence-attack*[13], an attack based on partial knowledge about different individuals in the same group across different snapshots.

## 3.1 Insert

In our model, a new tuple is inserted into table $I$ from Definition 2.5, with values encrypted at the tuple level:

Definition 3.1 (Insert). *Given an insert statement*

$\texttt{INSERT INTO T VALUES}\ (\texttt{t.A}_1, ..., \texttt{t.A}_\texttt{d}, \texttt{t.A}_\texttt{s})$

*whose values are compatible with the schema of $T$ as in Definition 2.4, the client translates the statement into*

$\texttt{INSERT INTO I VALUES}\ (\texttt{s}, \mathrm{E}_\texttt{k}\ (\texttt{salt}, \texttt{t.A}_1, \ldots, \texttt{t.A}_\texttt{d}, \texttt{t.A}_\texttt{s}), \texttt{i})$

*where $s$ is the next sequence number for $I$, salt is a pseudo-random number to randomize the encryption $\mathrm{E}_k(\cdot)$, and $i$ is the current snapshot of the dataset.*

The current snapshot is stored each time a tuple is inserted, to keep track of tuples in $I$ that might be left over after an anatomization (which will be discussed further in Section 3.4).

For example, the tuple identifying *Michael* is inserted into table $I$ in Figure 3c as a tuple of the form,

$$\langle 1, \mathrm{E}_k\ (\mathit{salt}, \mathit{Michael}, \mathit{25}, \mathit{Richmond}, \mathit{Flu}), 1\rangle$$

Given semantic security of the encryption, the only thing learned by the server is the sequence number; in practice this is maintained at and added to the insert by the server.

## 3.2 Delete

Deleting a new tuple that is stored in $I$ or $U$ is straightforward; the tuple is simply removed as with a traditional delete operation. Since these tables reveal no information about the sensitive value (due to the encryption), there is no change in the ability of the server to link an individual to a particular sensitive value and the privacy is unaffected. However, if the tuple is in $IT$ and $ST$, then only the part of the tuple that is in $IT$ is deleted. If the portion in $ST$ were deleted, the simultaneous removal would reveal the sensitive value associated with the individual. Furthermore, any other tuple in the same group will match to the remaining sensitive values in the group with higher probability than previously. By deleting only the portion of the tuple in $IT$, the sensitive information associated with any tuple is unchanged. As an example to the deletion of a single tuple is shown in Figure 3a where Eric's tuple is deleted whereas his disease, Fever, is not deleted.

Delete is limited to predicates on identifying information. Deleting a tuple based on the sensitive value would reveal the sensitive value associated with that individual. (The mirror-image approach of deleting just the sensitive value and leaving identifying information introduces numerous issues, most obviously the reduction in diversity of values in the group.) Therefore we restrict deletion as follows:

Definition 3.2 (IT Predicate). *A predicate, $P$, is said to be an ITPredicate ($P_{IT}$) if all the attributes in $P$ are from table $IT$.*

The delete operation is as follows:

Definition 3.3 (Delete). *Given a delete statement*

$\texttt{DELETE FROM T WHERE P}_\texttt{IT}$

*where $P_{IT}$ is defined as in Definition 3.2, the client translates it to three substatements*

$\texttt{DELETE FROM IT WHERE P}_\texttt{IT}$ (3.3.1)

| Patient | Age | City | GID | ESEQ |
|---|---|---|---|---|
| | | | | |
| Mike | 47 | Richmond | 3 | $\mathrm{E}_k\ (salt, 6)$ |

(a) $\mathrm{Patient_{IT}}$

| SEQ | GID | Disease |
|---|---|---|
| 5 | 3 | Flu |
| 6 | 3 | Fever |

(b) $\mathrm{Patient_{ST}}$

| SEQ | ENC | SS |
|---|---|---|
| 1 | $\mathrm{E}_k$ (*salt, Michael, 25, Richmond, Flu*) | 1 |

(c) $\mathrm{Patient_{I}}$

Figure 4: After Delete Operation

$\texttt{DELETE FROM U WHERE P}_\texttt{IT}$ (3.3.2)

$\texttt{SELECT * FROM I}$

*and sends them to the server. After executing delete statement 3.3.1 and 3.3.2, the server marks all the groups in $IT$ where some tuples are deleted as having non-1:1 mapping (if all tuples in a group are deleted, the server also deletes all corresponding tuples of that group in $ST$) and sends table $I$ to the client. After getting the results for the final selection, the client decrypts $I$ to get a set of sequence numbers, $R$:*

$R = \texttt{SELECT SEQ FROM I}^\texttt{d}\ \texttt{WHERE P}_\texttt{IT}$

*and issues a second delete statement to the server*

$\texttt{DELETE FROM I WHERE SEQ} = \texttt{R}_1\ \texttt{OR} \ldots \texttt{OR SEQ} = \texttt{R}_\texttt{m}$ (3.3.3)

*where $m = |R|$.*

For example, assume that the client decides to no longer provide service in Lafayette and Dayton. To delete these patients, the client issues the delete statement to the server:

$\texttt{DELETE FROM T WHERE City} = \texttt{Lafayette OR City} = \texttt{Dayton}$

Based on Definition 3.3, outcome is shown in Figure 4.

If we are deleting only a single individual (and the client knows this in advance), this can be optimized – if the delete from $IT$ or $U$ succeeds, the second step involving $I$ can be ignored (since each individual appears only once.)

By deleting identifying information of individuals without any correlation to their sensitive information, the server gains no additional knowledge about the sensitive value associated with any individual. As will be discussed in Section 3.4, extra care must be taken during the anatomization of groups where individuals have been deleted.

Lemma 3.1. *Assuming an individual, $p$, who will be deleted in group $G$ can match any tuple in $G_{ST}$, deleting the tuple representing $p$ in $G_{IT}$ does not change the probability distribution of matching other individuals in $G_{IT}$ to sensitive values in $G_{ST}$.*

Proof. Let $\mathcal{D}$ be the probability distribution of matching $t \in G_{IT}$ to the tuples in $G_{ST}$. When a tuple $t' \in G_{IT}$ where $t \neq t'$ is deleted, $\forall s \in G_{ST}$

$$\begin{aligned}\mathcal{P}\{t \implies s\} &= \mathcal{P}\{t \implies s | t' \not\Longrightarrow s\}\mathcal{P}\{t' \not\Longrightarrow s\} \\ &\quad + \mathcal{P}\{t \implies s | t' \implies s\}\mathcal{P}\{t' \implies s\} \\ &= \frac{1}{l-1} \cdot \frac{l-1}{l} + 0 \cdot \frac{1}{l} = \frac{1}{l}\end{aligned}$$

where $l$ is the number of distinct sensitive values in $G_{ST}$. Since the probabilities of matchings between tuples in $G_{IT}$ and $G_{ST}$ remains the same for any $t \in G_{IT}$ and $s \in G_{ST}$, the distribution $\mathcal{D}$ remains the same for every $t \in G_{IT}$. $\square$

Delete lets us show an interesting issue with background knowledge. If the server knows $t' \implies s'$, $\mathcal{P}\{t \implies s\} = 1/(l-1)$ if $s \neq s'$ and $\mathcal{P}\{t \implies s\} = 0$ otherwise. However, this is true even if $t'$ is not deleted. Deletion does not change the underlying probability distribution of $G$ with respect to the server's background knowledge. This is critical, as it is easy to develop mechanisms where insert/update/delete meet privacy requirements in the absence of background knowledge, but allow much stronger inference based on background knowledge than the static dataset would.

THEOREM 3.1. *Assuming there is no correlation between $P_{IT}$ and sensitive values of the individuals satisfying $P_{IT}$, the delete protocol defined in Definition 3.3 does not change the probability distribution of the remaining tuples in $IT$, $ST$, $I$ and $U$.*

PROOF. Statement 3.3.1 does not violate privacy by Lemma 3.1. Statement 3.3.2 does not violate privacy since revealing identifying information in $U$ does not violate privacy. Statement 3.3.3 is does not violate privacy since the fact that all the remaining tuples in table $I$ satisfy $\overline{P_{IT}}$ when decrypted does not change the probability distribution of these tuples in the subsequent snapshots (proven in Section 3.4). □

## 3.3 Update

Updates that can be translated into an update on just $IT$ or just $ST$ (in other words, the predicate AND updated values are from the same subtable) can be performed directly (with appropriate client-side operations on $I$ and $U$ to keep them up to date), provided an update to $ST$ preserves diversity (if not, this is treated like the update to a particular individual below.) These either convey no new sensitive information, or no information distinguishing individuals, and are thus safe. Any update to sensitive values of tuples in $I$ sets the attribute $SS$ to the current snapshot of the dataset for that tuple (the reason will be apparent in Section 3.4.)

The interesting case is an update to sensitive information for a particular individual. If the tuple is in $\langle IT, ST\rangle$ then it is removed from $IT$ only and inserted into the temporary encrypted table $U$, along with its new sensitive value (encrypted) and previous group's signature.

If the tuple is in $U$, the client decrypts the sensitive value and updates it with the new encrypted value. If the tuple to be updated is in $I$, the client decrypts the tuple, updates it and then inserts it into $I$ again.

Update operations changing the sensitive value for multiple tuples (e.g., `UPDATE T SET Disease = 'Flu' WHERE ...`) may violate individual privacy. These update operations seem safe since the sensitive values of the updated tuples are stored encrypted in table $U$; however, the sensitive values of the update tuples in this operation would be revealed in $ST$ after the next anatomization and a simple intersection of groups containing these tuples can reveal more information than allowed by $l$-diversity. We disallow this kind of update operation, assuming clients will be dealing with one individual (e.g., patient or customer) at a time.

One likely case of multiple updates, global replacement, is allowed (e.g.,
`UPDATE T SET Disease = 'Influenza' WHERE Disease = 'Flu'`). As previously described, updates that only involve $IT$ or $ST$ do not pose a threat, provided updates to $ST$ do not reduce diversity of the group. This generalizes to updating the sensitive value of multiple tuples with a function as depicted in Definition 3.4. E.g., a sensitive salary value could be set to a new value based on (identifying) job title and location. We assume that the function for updating the sensitive value cannot be deduced by the server based on the update operation.

DEFINITION 3.4 (UPDATE). *Given an update statement*

$\mathtt{UPDATE\ T\ SET}\ \mathcal{A}, \mathcal{F}\ \mathtt{WHERE\ P_{IT}}$

*where $\mathcal{A}$ is a set of value assignments for some attributes in $IT$, $\mathcal{F}$ is a function updating the sensitive values in $ST$ that is **independent** of the sensitive values to be updated, and $P_{IT}$ is defined as in Definition 3.2, the client first retrieves the tuples to be updated:*

$R_1 = \mathtt{SELECT} * \mathtt{FROM}\ \langle \mathtt{IT}, \mathtt{ST}\rangle$ [3] $\mathtt{WHERE\ P_{IT}}$

$R_2 = \mathtt{SELECT} * \mathtt{FROM\ U\ WHERE\ P_{IT}}$

$\mathtt{SELECT} * \mathtt{FROM\ I}$

*The client then decrypts $I$ and computes:*

$R_3 = \mathtt{SELECT} * \mathtt{FROM\ I^d\ WHERE\ P_{IT}}$

$\mathtt{UPDATE}\ \overline{\mathrm{R}'_1} \cup \mathrm{R}_2^{\mathrm{d}} \cup \mathrm{R}_3\ \mathtt{SET}\ \mathcal{A}, \mathcal{F}$

*where table $R'_1$ consists of every tuple $t \in R_1$ where $G_t$ still contains the new sensitive value of $t$ and table $\overline{R'_1} = R_1 - R'_1$. Assuming t.seq denotes the sequence number of the tuple in $ST$ representing the updated sensitive value of $t$ in $G_t$, the client sends to the server*

$\mathtt{DELETE\ FROM\ IT\ WHERE\ A_1} = \mathtt{t^\circ.A_1} \wedge \ldots \wedge \mathtt{A_d} = \mathtt{t^\circ.A_d}, \quad \forall t \in \overline{R'_1}$

*and the server marks all the groups in $IT$ where some tuples are deleted or updated as having non-1:1 mapping (if all tuples in a group are deleted, the server also deletes all corresponding tuples of that group in $ST$). Then the client sends the following statements to the server to finalize the update operation*

$\mathtt{UPDATE\ IT\ SET}\ \mathcal{A}, \mathtt{ESEQ} = \mathrm{E_k}(\mathtt{salt}, \mathtt{t.seq})$

$\quad \mathtt{WHERE\ A_1} = \mathtt{t.A_1} \wedge \ldots \wedge \mathtt{A_d} = \mathtt{t.A_d}, \qquad \forall t \in R'_1$

$\mathtt{INSERT\ INTO\ U\ VALUES}$

$\quad (\mathtt{s}, \mathtt{t.A_1}, \ldots, \mathtt{t.A_d}, \mathrm{E_k}(\mathtt{salt}, \mathtt{t.A_s}), \mathtt{t.}\mathcal{S}), \qquad \forall t \in \overline{R'_1}$

$\mathtt{UPDATE\ U\ SET}\ \mathcal{A}, \mathtt{ENC} = \mathrm{E_k}(\mathtt{salt}, \mathtt{t.A_s}), \mathcal{S}^- = \emptyset$

$\quad \mathtt{WHERE\ SEQ} = \mathtt{t.SEQ}, \qquad \forall t \in R_2^d$

$\mathtt{UPDATE\ I\ SET\ ENC} = \mathrm{E_k}(\mathtt{salt}, \mathtt{t.A_1}, \ldots, \mathtt{t.A_d}, \mathtt{t.A_s}),$

$\quad \mathtt{SS} = \mathtt{i}\ \mathtt{WHERE\ SEQ} = \mathtt{t.SEQ}, \qquad \forall t \in R_3$

*where i is the current snapshot counter.*

For example, assume an (admittedly contrived) scenario where people in Dayton were mistakenly listed as Lafayette and having the wrong diagnosis. The client wants to execute

$\mathtt{UPDATE\ T\ SET\ City} = \mathtt{Dayton}, \mathcal{F}\ \mathtt{WHERE\ City} = \mathtt{Lafayette}$

where $\mathcal{F}$ changes Disease from Flu, Fever, Cough, and Cold to Fever, Cold, Flu, and Cough respectively. If the current snapshot of the dataset in Figure 3 is 2, the outcome of this update operation is illustrated in Figure 5.

[3]This is not a join operation, it is the selection query described in [20] which is semantically equal to $\mathtt{SELECT} * \mathtt{FROM\ IT^d, ST\ WHERE\ P_{IT}}$ and involves client-server interaction in this private data outsourcing setting.

| Patient | Age | City | GID | ESEQ |
|---|---|---|---|---|
| Ike | 41 | Dayton | 1 | $\mathrm{E}_k$ (*salt*, 1) |
| | | | | |
| Faye | 24 | *Dayton* | 3 | $\mathrm{E}_k$ (*salt*, 6) |
| Mike | 47 | Richmond | 3 | $\mathrm{E}_k$ (*salt*, 6) |
| | | | | |
| Max | 31 | Dayton | 4 | $\mathrm{E}_k$ (*salt*, 8) |

(a) $\text{Patient}_{\text{IT}}$

| SEQ | GID | Disease |
|---|---|---|
| 1 | 1 | Cold |
| 2 | 1 | Fever |
| 5 | 3 | Flu |
| 6 | 3 | Fever |
| 7 | 4 | Cough |
| 8 | 4 | Cold |

(b) $\text{Patient}_{\text{ST}}$

| SEQ | ENC | SS |
|---|---|---|
| 1 | $\mathrm{E}_k$ (*salt, Michael, 25, Richmond, Flu*) | 1 |

(c) $\text{Patient}_{\text{I}}$

| SEQ | Patient | Age | City | ENC | $\mathcal{S}^-$ |
|---|---|---|---|---|---|
| 1 | Kelly | *36* | *Dayton* | $\mathrm{E}_k$ (*salt, Cold*) | $\emptyset$ |
| 2 | Olga | 30 | *Dayton* | $\mathrm{E}_k$ (*salt, Fever*) | {Flu,Cough} |
| 3 | Jason | 45 | *Dayton* | $\mathrm{E}_k$ (*salt, Flu*) | {Cough, Cold} |

(d) $\text{Patient}_{\text{U}}$

Figure 5: After Update Operation

No change to sensitive information is visible to the server, and no information is given about sensitive values for individuals inserted into $I$ or $U$, so the probability that an individual is linked to a particular sensitive value does not change. While formal proof is omitted due to space constraints, the difficult parts essentially follow Theorem 3.1.

## 3.4 Anatomization Algorithm

We now give an anatomization algorithm that preserves $l$-diversity in spite of the historical and update information available to the server. Loosely speaking, the algorithm output prevents the server from getting additional information from the knowledge of write operations, satisfying Definition 2.10. We assume the adversary's background knowledge is limited to previous snapshots of the database. Knowing a single individual's data provides the same guarantee as $l$-diversity: it reduces the privacy guarantee for other individuals in the group to at worst $l-1$-diversity.

The first inference channel is due to an update operation on both identifying and sensitive values. This inference channel relies on the fact that updating the sensitive value replaces the old value with a different sensitive value.

Theorem 3.2. *Given a tuple $t$ in a group with signature $\mathcal{S}$ having simple $l$-diverse distribution, and an update operation modifying both identifying and sensitive information; moving $t$ into a group with $\mathcal{S}'$ where $\mathcal{S}\cap\mathcal{S}'\neq\emptyset$, $|\mathcal{S}|=|\mathcal{S}'|$, and $\mathcal{S}\neq S'$, causes the group to not have a simple $l$-diverse distribution.*

Proof. For a value $v\in\mathcal{S}\cap\mathcal{S}'$,

$$\begin{aligned}
\mathcal{P}\{t.A_s=v\} &= \bigcup_{v^o\in\mathcal{S}}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&= \bigcup_{v^o\in\mathcal{S}-\mathcal{S}'}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&\quad+\bigcup_{v^o\in\mathcal{S}\cap\mathcal{S}'-\{v\}}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&\quad+\mathcal{P}\{t.A_s=v|t^o.A_s=v\}\mathcal{P}\{t^o.A_s=v\}\\
&= |\mathcal{S}-\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|}\cdot\frac{1}{|\mathcal{S}|}\\
&\quad+(|\mathcal{S}\cap\mathcal{S}'|-1)\cdot\frac{1}{|\mathcal{S}|-1}\cdot\frac{1}{|\mathcal{S}|}\\
&\quad+0\\
&= |\mathcal{S}-\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|^2}\\
&\quad+(|\mathcal{S}\cap\mathcal{S}'|-1)\cdot\left(\frac{1}{|\mathcal{S}|}+\frac{1}{|\mathcal{S}|^2-|\mathcal{S}|}\right)\cdot\frac{1}{|\mathcal{S}|}\\
&= |\mathcal{S}-\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|^2}+|\mathcal{S}\cap\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|^2}+\frac{|\mathcal{S}\cap\mathcal{S}'|-|\mathcal{S}|}{|\mathcal{S}|^2(|\mathcal{S}|-1)}\\
&= \frac{1}{|\mathcal{S}|}+\frac{|\mathcal{S}\cap\mathcal{S}'|-|\mathcal{S}|}{|\mathcal{S}|^2(|\mathcal{S}|-1)}<\frac{1}{|\mathcal{S}|}
\end{aligned}$$

as long as $\mathcal{S}\neq S'$. And for a value $v\in\mathcal{S}'-\mathcal{S}$,

$$\begin{aligned}
\mathcal{P}\{t.A_s=v\} &= \bigcup_{v^o\in\mathcal{S}}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&= \bigcup_{v^o\in\mathcal{S}-\mathcal{S}'}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&\quad+\bigcup_{v^o\in\mathcal{S}\cap\mathcal{S}'}\mathcal{P}\{t.A_s=v|t^o.A_s=v^o\}\mathcal{P}\{t^o.A_s=v^o\}\\
&= |\mathcal{S}-\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|}\cdot\frac{1}{|\mathcal{S}|}+(|\mathcal{S}\cap\mathcal{S}'|)\cdot\frac{1}{|\mathcal{S}|-1}\cdot\frac{1}{|\mathcal{S}|}\\
&> (|\mathcal{S}-\mathcal{S}'|+|\mathcal{S}\cap\mathcal{S}'|)\cdot\frac{1}{|\mathcal{S}|}\cdot\frac{1}{|\mathcal{S}|}=\frac{1}{|\mathcal{S}|}
\end{aligned}$$

□

Corollary 3.1. *Given a tuple, $t$, in a group with signature $\mathcal{S}$ having simple $l$-diverse distribution and an update operation modifying both identifying and sensitive information such that $t.A_s$ is still $\in\mathcal{S}$; having $t$ in a group with $\mathcal{S}'$ where $\mathcal{S}=\mathcal{S}'$, maintains simple $l$-diverse distribution.*

Proof. For a value $v\in\mathcal{S}\cap\mathcal{S}'=\mathcal{S}$,

$$\mathcal{P}\{t.A_s=v\}=\frac{1}{|\mathcal{S}|}+\frac{|\mathcal{S}\cap\mathcal{S}'|-|\mathcal{S}|}{|\mathcal{S}|^2(|\mathcal{S}|-1)}=\frac{1}{|\mathcal{S}|}+0=\frac{1}{|\mathcal{S}|}$$

□

Corollary 3.2. *Given a tuple $t$ in a group with signature $\mathcal{S}$ having simple $l$-diverse distribution and an update operation modifying both identifying and sensitive information such that $t.A_s\notin\mathcal{S}$; having $t$ in a group with $\mathcal{S}'$ where $\mathcal{S}\cap\mathcal{S}'=\emptyset$, maintains simple $l$-diverse distribution.*

Proof. For a value $v\in\mathcal{S}'-\mathcal{S}=\mathcal{S}'$,

$$\begin{aligned}
\mathcal{P}\{t.A_s=v\} &= |\mathcal{S}-\mathcal{S}'|\cdot\frac{1}{|\mathcal{S}|^2}+(|\mathcal{S}\cap\mathcal{S}'|)\cdot\frac{1}{|\mathcal{S}|-1}\cdot\frac{1}{|\mathcal{S}|}\\
&= |\mathcal{S}|\cdot\frac{1}{|\mathcal{S}|^2}+0\cdot\frac{1}{|\mathcal{S}|-1}\cdot\frac{1}{|\mathcal{S}|}=\frac{1}{|\mathcal{S}|}
\end{aligned}$$

□

The next challenge is the anatomization of inserted tuples. A naïve approach would be to check for $l$-eligibility after every insert; this is expensive given that the client does not maintain any state. Instead, we anatomize inserted tuples when a fixed number of tuples have been inserted. This approach requires the anatomization algorithm to consider left-over tuples that cannot be included in the anatomized dataset since it is crucial to be able to use these left-over tuples in later anatomization batches. The challenge is that the fact that the tuples are left over shows that they could not be included in the current batch, revealing something about their possible values. The underlying model needs to consider this fact and ensure this knowledge does not enable the server to violate privacy of later anatomization batches. Next, we show encrypting both identifying and sensitive information achieves this goal.

Definition 3.5 (Partial Anatomization). *Given a table $I(i)$ as in Definition 2.5 and 2.8 where $l$-eligibility is not required, a partial anatomization algorithm based on $l$-diversity denoted by $PA$ outputs groups having strictly $l$ number of tuples with distinct sensitive values and a smallest set of encrypted tuples, $I(i+1) \subseteq I(i)$, that cannot be anatomized since $I(i+1)$ is not $l$-diverse.*

Pseudocode for $PA$ is given in Algorithm 2 based on the anatomization algorithm in [30].

Definition 3.6 (Left-over Analyzer: LA). *Given $PA(I(i))$, $PA(I(i+1))$ and a group $G \in PA(I(i+1))$ where $i \geq 1$, a left-over analyzer algorithm denoted by $LA$ outputs a set of sensitive values, $S_{LA} \subseteq G.\mathcal{S}$, that left-over tuples in $I(i+1)$ can match.*

For instance, assume another tuple with Flu is added into $I(1)$ in Figure 3c and then $PA$ with $l=2$ is run on $I(1)$. All the tuples are left-over changing $I(1)$ to $I(2)$. Now two more tuples are added into $I(2)$, one with Cold and the other with Cough. This time $PA$ outputs two anatomized groups such that $G_1.\mathcal{S} = \{Flu, Cold\}$ and $G_2.\mathcal{S} = \{Flu, Cough\}$ (giving no left-over tuples). In this case, $LA(PA(I(1)), PA(I(2)), G_1)$ outputs $\{Flu\}$ which is the only sensitive value that a left-over tuple can take in group $G_1$.

Lemma 3.2. *Given $PA$ and $LA$ as in Definition 3.5 and 3.6 and a group $G \in PA(I(i+1))$ as part of an input to $LA$, the server can only match a tuple $t \in G_{IT}$ to a sensitive value $v \in S_{LA}$ with $1/l$ confidence.*

Proof. Let $X = |S_{LA}|$ and $x$ be the actual number of left-over tuples in $G$ where $0 \leq x \leq X$. For each $t \in G_{IT}$ and $v \in S_{LA}$,

$$\mathcal{P}\{t \to v\} = \sum_{0 \leq j \leq X} \mathcal{P}\{t \to v | x = j\} \mathcal{P}\{x = j\}$$

For $0 \leq j \leq X$,

$$\mathcal{P}\{t \to v | x = j\} = \sum_{m \in \{0,1\}} \mathcal{P}\{t \to v | f(t,v) = m, x = j\} \mathcal{P}\{f(t,v) = m | x = j\}$$

where $f$ is a boolean function indicating whether $\langle t, v\rangle$ pair is a left-over.

$$\begin{aligned}\mathcal{P}\{t \to v | x = j\} &= \frac{1}{l-j} \cdot \frac{l-j}{l} \cdot \frac{\binom{X-1}{j}}{\binom{X}{j}} + \frac{1}{j} \cdot \frac{j}{l} \cdot \frac{\binom{X-1}{j-1}}{\binom{X}{j}} \\ &= \frac{1}{l}\left(\frac{\binom{X-1}{j} + \binom{X-1}{j-1}}{\binom{X}{j}}\right) = \frac{1}{l}\end{aligned}$$

Since $\mathcal{P}\{t \to v | x = j\}$ does not change, then

$$\mathcal{P}\{t \to v\} = \frac{1}{l} \cdot \sum_{0 \leq j \leq X} \mathcal{P}\{x = j\} = \frac{1}{l}$$

□

Theorem 3.3. *Assuming an algorithm $PA$ as in Definition 3.5, and an encryption algorithm $E$ having ciphertext indistinguishability; a set of tuples $I' \subseteq I(i)$ left over by $A(I(i))$ can be used in the next run (i.e., snapshot $i+1$) of $A$ without violating Definition 2.10 so long as the server only sees $E(t)$ for every $t \in I'$.*

Proof. Assume $T^*(i)$ satisfies simple $l$-diverse distribution (Definition 2.9). Including the output of $LA$ in $TH(i)$ of Definition 2.10, Lemma 3.2 proves that any tuple $t \in I'$ has simple $l$-diverse distribution once anatomized. □

The correctness of Theorem 3.3 relies on the server not knowing the identifying information of the left-over tuples. As stated in Theorem 3.1, multi-tuple delete and update operations cause the server to learn partial identifying information of tuples in $I$, specifically the server knows the remaining tuples from delete and unchanged tuples from update operations satisfy $\overline{P_{IT}}$ that is used in these operations. This knowledge can disrupt the uniformity of tuples being a left-over in $IT$ unless every tuple in a group in $IT$ satisfies the same predicate. As a result, based on Lemma 3.2, $\mathcal{P}\{f(t,v) = m | x = j\}$ values change and thus $\mathcal{P}\{t \to v\}$ is not $1/l$ for each $t \in G_{IT}$ and $v \in S_{LA}$.

To maintain the correctness of Lemma 3.2, we anatomize left-over tuples if the server does not even know their identifying information partially and that is why the snapshot index is stored for each inserted tuple. An anatomization algorithm can safely anatomize all the tuples in $I$ having a snapshot index greater than the one in which the last multi-tuple delete or update operation has occurred. Note that the possibility of being a left-over tuple in $IT$ can only be used if the server also knows the left-over sensitive values in the same group. In case of updating the sensitive value of a left-over tuple, any knowledge of the tuple in $IT$ being left-over is useless for the server. That is why the update operation also updates $SS$ field in $I$ to the current snapshot index when updating sensitive values in Definition 3.4.

Definition 2.10 also prevents us from converting non-1:1 mapping groups to 1:1 mapping groups. While needed for some optimizations for projection and group-by operations in [20], it raises the possibility that the server gains too much information. Even if such groups become a 1:1 mapping later, the server must not know it.

Based on Corollary 3.1, two groups, $G_1$ and $G_2$, that have missing tuples can only be merged if $G_1.\mathcal{S} = G_2.\mathcal{S}$. Now we show how to process merging of such groups.

Theorem 3.4. *Given two groups, $G_1$ and $G_2$, that have missing tuples and satisfy $G_1.\mathcal{S} = G_2.\mathcal{S}$; merging $G_1$ and $G_2$ is in accordance with Definition 2.10 if the server does not know whether the outcome group has a 1:1 mapping.*

Proof. Let $t$ be a tuple that has been deleted from $G_1$, $t.A_s = v$, $p$ be the individual represented by $t$, and $G_2$ is merged into $G_1$. Since it is not known if outcome $G_1$ has 1:1 mapping, $\mathcal{P}\{t'.A_s = v\} = 1/l$ for any $t' \in G_2$. □

If $p$ knows that $G_1$ has 1:1 mapping after the merge then $\mathcal{P}\{t'.A_s = v\} \neq 1/l$, since there is at least one assignment of sensitive values in $G_1.\mathcal{S}$ to the tuples of $G_2$ where $v$ is not assigned to any tuples of $G_2$. That assignment becomes impossible since at least one tuple from $G_2$ must be assigned to $v$ in order to make $G_1$ a 1:1 mapping.

The last problem is how to anatomize table $U$. All tuples in $U$ have a $\mathcal{S}^-$ signature, and must be anatomized into groups compatible with their signatures. The naïve solution is to generate a $\mathcal{S}$ for updated tuples where $\mathcal{S} \cap \mathcal{S}^- = \emptyset$ and $\mathcal{S}$ has the same distribution as the dataset. However, this solution violates Definition 2.10 since there might be some sensitive values in $D_{A_s}$ that are not possible during the time of anatomization. For instance, a tuple's disease cannot be updated to small pox in this era but the database itself might have tuples having small pox. Instead, we compare each updated tuple with every group having a non-1:1 mapping to check if it can be inserted into an existing group.

We now show how to do this comparison and simulate it to satisfy Definition 2.10.

THEOREM 3.5. *Given a tuple $t \in U$ and an $l$-diverse group $g \in G$ where* a) $|D_{A_s}| - |t.\mathcal{S}^-| - |g.\mathcal{S}| \geq l$, b) $t.\mathcal{S}^- \cap g.\mathcal{S} = \emptyset$,

*1. moving $t$ into $g$ and setting $g$ as non-1:1 mapping if $t.A_s \in g.\mathcal{S}$*

*2. or updating $t.\mathcal{S}^- = t.\mathcal{S}^- \cup g.\mathcal{S}$ otherwise*

*is in accordance with Definition 2.10 under $l$-diversity.*

PROOF. Assuming an adversary cannot get any information about $t.A_s$ other than $t.A_s \notin t.\mathcal{S}^-$ by examining all previous history, case 1 only allows the adversary to learn $t.A_s \in g.\mathcal{S}$ since condition b renders the information $t.A_s \notin t.\mathcal{S}^-$ useless based on Corollary 3.2. Case 2 does not hide the fact that $t.A_s \notin g.\mathcal{S}$ from the adversary by updating $t.\mathcal{S}^-$. The adversary only updates $t.A_s \notin t.\mathcal{S}^-$ and $l$-diversity is not violated since there are at least $l$ possible sensitive values for $t.A_s$ due to the condition b. □

Note that the condition $b$ in Theorem 3.5 prevents the algorithm from violating $l$-diversity since $t.\mathcal{S}^-$ might get too large without this condition and $t.A_s$ can take less than $l$ values. Although this seems to cause a similar problem as in the naïve approach (since some values $t.A_s$ can take might not be possible), the probability of satisfying $t.\mathcal{S}^- \cap g.\mathcal{S} = \emptyset$ is negligible when the size of $t.\mathcal{S}^-$ and $D_{A_s}$ are large enough compared to $l$ due to the birthday paradox.

Our anatomization algorithm is shown in Algorithm 1.

**Algorithm 1:** Anatomization of $U$ and $I$

```
   require: $n$ number of tuples in $U$ and $I$ combined
   input  : Table $U$, $I$ and the set of groups, $G$, from
            table pair $\langle IT, ST\rangle$ where $\forall g \in G$, g has non
            1:1 mapping
   output : A set of groups that is used to generate
            $\langle IT, ST\rangle$ table pair
 1 $G_{out}$=Anatomize($I^d$)                    // Algorithm 2
   // Lines 2-4 merges groups with same signatures
 2 put groups in $G$ into buckets $B$ based on their
   signature;
 3 foreach bucket $b \in B$ do
 4     merge groups in bucket $b$ into one group $g$;
   // Lines 5-10 moves updated tuples into groups
   // if possible
 5 foreach tuple $t \in U^d$ do
 6     foreach group $g \in G$ where
       $|D_{A_s}| - |t.\mathcal{S}^-| - |g.\mathcal{S}| \geq l$ and $t.\mathcal{S}^- \cap g.\mathcal{S} = \emptyset$ do
 7         if $t.A_s \in g.\mathcal{S}$ then
 8             move $t$ from $U^d$ into $g$;
 9         else
10             $t.\mathcal{S}^- = t.\mathcal{S}^- \cup g.\mathcal{S}$;
11 insert each group $g \in G$ into $G_{out}$;
12 increment snapshot counter by 1;
13 return $G_{out}$
```

## 4. QUERY PROCESSING

In general, the query processing of [20] still applies with our update model, with the addition that the encrypted temporary tables $I$ (and parts of $U$) must be sent to the client for decryption and merging with the final results. There are a few differences, mainly due to the invalid sensitive data left behind during deletes. We outline these differences below.

*Selection* is straightforward. Processing proceeds as in [20], except that $I$ is sent to the client for decryption and processing. When the selection criteria involves $IT$, the server can perform selection on $U$ and send only matching tuples, otherwise $U$ must be sent in its entirety. Invalid values in $ST$ resulting from deletion will automatically drop out, as they won't join with any of the corresponding tuples in $IT$ (but see projection, below.)

*Projection* can raise issues. Again, the table $I$ is sent completely, and projection on $U$ can be done at the server. However, deletion raises issues. If a projection includes attributes from $IT$ and $ST$, the deleted sensitive values will "drop out" when the client joins the groups (as with selection.) However, if the projection is only to $ST$, the server must also return the sequence numbers from corresponding groups in $IT$ (but only for groups having non-1:1 mapping).The client hashes these sequence numbers and compares with the values in $ST$ to determine which are valid.

A second issue arises with duplicate elimination. In [20], some duplicate elimination can be done on the server. This is challenging when the projected attributes come from both $IT$ and $ST$, as the actual matching is not known, and if something is a duplicate may depend on which way the match occurs. This is even more difficult with invalid sensitive values left from deletion. However, the server can determine which groups may have invalid values as it knows which groups have non-1:1 mappings; groups having non-1:1 mapping are removed from the server-side duplicate elimination of [20] and returned to the client to complete processing.

*Group-By* queries face the same issues as duplicate elimination: deleted values must not be used in the result. The work in [20] optimizes processing at the server side when all of the tuples in a group have the same value with respect to the grouping criterion. This cannot be done for groups with deleted tuples; these (and $I$ and $U$) must be processed at the client. Note that the work in [20] already handles client-side group-by processing for groups that have different values with respect to the group-by criterion; this simply adds additional data to the existing client-side processing.

## 5. EXPERIMENTS

We have proven that this approach preserves $l$-diversity (and with a different anatomization algorithm, could support other privacy measures.) However, this does reduce the ability of the server to efficiently process queries; less processing can be done on $U$ and non-1:1 mappings, and none at all on $I$. How big a problem is this?

To evaluate this, we give a simulation using the first 100,000 individuals from the IPUMS Census data [3], the same data used in [30]. The sensitive attribute can take 50 different values. We anatomize the first 20k individuals, giving the initial $\langle IT, ST\rangle$ pair. The remaining 80k tuples are inserted, along with updates and deletes at varying rates. Each up-

**Algorithm 2:** Anatomize: Modified Anatomization

```
input  : table $I^d$
output : A set of $l$-diverse groups
1  $i \leftarrow$ the last snapshot of $T^*$ that a multi-tuple delete
   or update has been issued;
2  if $i$ is the current snapshot of $T^*$ then
3      $I' \leftarrow$ SELECT $*$ FROM $I^d$ WHERE SS $=$ i;
4  else
5      $I' \leftarrow$ SELECT $*$ FROM $I^d$ WHERE SS $>$ i;
6  put tuples in $I'$ into buckets $B$ based on $A_s$ value;
7  while $|B| \geq l$ do
8      $B' \leftarrow$ largest $l$ number of buckets in $B$;
9      create group $g$;
10     foreach bucket $b \in B'$ do
11         randomly select a tuple, $t \in b$;
12         put $t$ into $g$;
13         send 'DELETE FROM I WHERE SEQ $=$ t.SEQ';
14     put $g$ into the set of groups $G$;
15 return $G$;
```

date or delete is done against a single randomly selected tuple. Updates change the sensitive attribute to a randomly selected value based on the distribution of the data. Update and delete rates are per block of 200 inserts. Tables $I$ and $U$ are anatomized every 200 inserts.

In each snapshot of $T^*$, the simulation keeps track of the number of tuples and groups as well as the number of groups having 1:1 and non-1:1 mapping. In addition to reporting average values, we will use $C_{90}$ to denote the best of the worst 10%, and $C_{100}$ to denote the worst, of all measured counts across the 400 anatomizations.

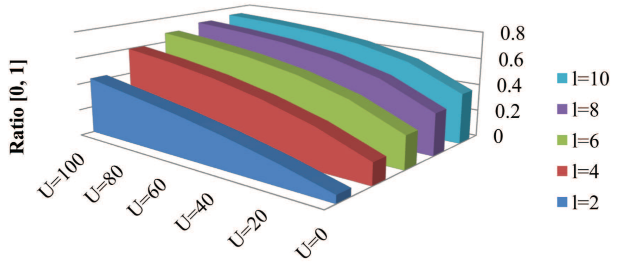

Figure 6: Ratio of tuples in groups having non-1:1 mapping to the tuples in $\langle IT, ST \rangle$ where $D = 20$

As mentioned in Section 4, both projection and group-by benefit from groups having 1:1 mapping. Figure 6 shows the average ratio of tuples in non-1:1 mapping groups to the tuples in $\langle IT, ST \rangle$. While this increases with update rate (or $l$, as larger $l$ gives larger groups), the ratio is logarithmic with the increase of either since there is higher chance for a tuple in a non-1:1 mapping group to get updated with the increase of such tuples. As $l$ increases, this ratio increases rapidly but then converges. Moreover, $C_{90}$ and $C_{100}$ values are roughly the same for any $U, D, l$ value and close to the average (e.g., average ratio is 0.73, both $C_{90}$ and $C_{100} \approx 0.77$ when $l = 10, U = 100, D = 20$). As both update and delete can convert a 1:1 mapping group into a non-1:1 mapping, and an anatomized update tuple only contributes 1 for the non-1:1 mapping count, the effect of update and delete rate on this ratio is similar.

Our experiment shows that the size of $I$ is less than $l$ on average after each anatomization. When $l = 10$, the max $C_{100}$ for the size of $I$ is 19 and the max $C_{100}$ for the duration an inserted tuple stays in $I$ is 7 among different $U$ and $D$ rates picked from $\{0, 20, \ldots, 100\}$. Tuples in $I$ do not stay encrypted long provided the inserts are uniformly distributed over the domain of the sensitive attribute.

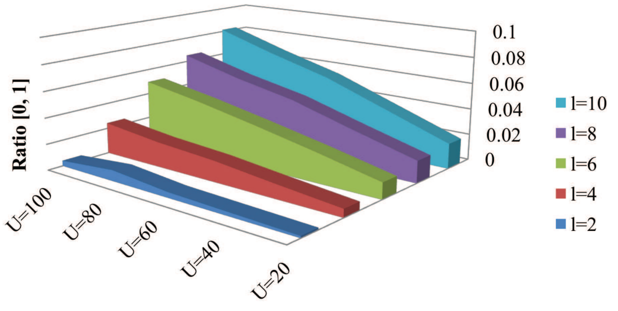

Figure 7: Ratio of tuples in $U$ to the tuples in $T^*$ where $D = 20$

Another issue that might effect the query processing is the size of $U$, although only the sensitive value is encrypted in $U$. Figure 7 shows that the ratio of the size of $U$ to the size of $T^*$ is linear with the increase of either $l$ or $U$. When $l$ increases, the $\mathcal{S}^-$ of a tuple increases as well, which decreases the chance an update tuple is anatomized since finding a non-1:1 group with no intersection with $\mathcal{S}^-$ is more difficult when the sets are larger due to the birthday paradox. However, the delete rate does not have a major effect; the average ratio is 0.079 when $D = 0, U = 100, l = 10$ and 0.085 when $D = 100, U = 100, l = 10$. Moreover, both $C_{90}$ and $C_{100}$ for this ratio is close to the average value. For instance, both $C_{90}$ and $C_{100} \approx 0.1$ when $l = 10, U = 100, D = 100$.

## 6. CONCLUSION AND FUTURE WORK

This paper presents a solution to updating outsourced private data. This is the first solution to this problem that does not assume either complete data encryption, or storage of all of the original data at the client - both impractical assumptions in the developing data outsourcing market. While there are still issues, such as dealing with multiple tables (although this can largely be dealt with through the anonymization methods of [21]), this brings us much closer to practical private data outsourcing.

The results suggest that this model is primarily useful where updates/deletes are rare. When inserts exceed 80% of the data modifications, 95% of the data is visible, and there is little dead data, allowing the server to perform substantial query processing. There do remain opportunities to optimize the anonymization process, although we have shown that in many cases, the approaches used are necessary to prevent privacy-violating inference channels. Databases meeting these criteria are not uncommon; Altoros Systems recently developed a set of database benchmarks that includes a workload with high insert/low update rate [6]. In particular, private data about individuals (e.g., transactional data, financial records, or medical data) is often retained indefinitely, but regulations typically require individuals be given the ability to correct and delete data.

Using this model the server can independently analyze the data. Building data mining models from data with such known types of inconsistencies poses an interesting challenge for future research. In particular, when the models involve combining the identifying and sensitive information, how to effectively build models when the server only knows group-level matching demands novel solutions.

*Acknowledgments.*

We thank Dr. Bechara al Bouna for his helpful comments on the proposed private data outsourcing model.

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.

## 7. REFERENCES

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *In Proc. CIDR*, 2005.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proc. SIGMOD*, Paris, France, June 13-18 2004.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proc. CRYPTO'07*, pages 535-552, 2007.

[5] Y. Bu, A. W. C. Fu, R. C. W. Wong, L. Chen, and J. Li. Privacy preserving serial data publishing by role composition. *Proc. VLDB*, 1(1):845-856, Aug. 2008.

[6] S. Bushik. A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak. *Network World*, Oct. 22 2012.

[7] J.-W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *Secure Data Management*, pages 48-63, 2006.

[8] V. Ciriani, S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Trans. Inf. Syst. Secur.*, 13(22):1-33, July 2010.

[9] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proc. 10th ACM CCS*, pages 93-102, Washington D.C., USA, 2003.

[10] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities*, No I.(281):31-50, Oct. 24 1995.

[11] B. C. M. Fung, K. Wang, A. W.-C. Fu, and J. Pei. Anonymity for continuous data publishing. In *Proc. EDBT'08*, pages 264-275, 2008.

[12] H. Hacıgümüş, B. R. Iyer, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. SIGMOD*, pages 216-227, Madison, Wisconsin, June 4-6 2002.

[13] Y. He, S. Barman, and J. Naughton. Preventing equivalence attacks in updated, anonymized data. In *ICDE*, pages 529-540, Apr. 2011.

[14] Standard for privacy of individually identifiable health information. *Federal Register*, 67(157):53181-53273, Aug. 14 2002.

[15] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proc. VLDB*, pages 720-731, 2004.

[16] M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. In *Data and Applications Security XIX*, pages 325-337, 2005.

[17] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 9(1):5-38, 1883.

[18] N. Li and T. Li. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proc. ICDE '07*, Istanbul, Turkey, Apr. 16-20 2007.

[19] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. $l$-diversity: Privacy beyond $k$-anonymity. *ACM Trans. on Knowledge Discovery from Data (TKDD)*, (1), Mar. 2007.

[20] A. E. Nergiz and C. Clifton. Query processing in private data outsourcing using anonymization. In Y. Li, editor, *Data and Applications Security and Privacy XXV*, pages 138-153, 2011.

[21] M. E. Nergiz, C. Clifton, and A. E. Nergiz. Multirelational k-anonymity. *IEEE Trans. on Knowl. and Data Eng.*, 21(8):1104-1117, Aug. 2009.

[22] A. Øhrn and L. Ohno-Machado. Using boolean reasoning to anonymize databases. *Artificial Intelligence in Medicine*, 15(3):235-254, Mar. 1999.

[23] J. Pei, J. Xu, Z. Wang, W. Wang, and K. Wang. Maintaining k-anonymity against incremental updates. In *SSBDM '07.*, page 5, July 2007.

[24] P. Samarati. Protecting respondent's privacy in microdata release. *IEEE Trans. on Knowledge and Data Engineering*, 13(6):1010-1027, Nov./Dec. 2001.

[25] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Proc. IEEE Symposium on Security and Privacy*, pages 350-364, Oakland, 2007.

[26] L. Sweeney. k-anonymity: a model for protecting privacy. *Intnl. Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, (5):557-570, 2002.

[27] K. Wang and B. C. M. Fung. Anonymizing sequential releases. In *Proc. KDD*, pages 414-423, 2006.

[28] R. Wong, A. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *Proc. VLDB*, pages 543-554, 2007.

[29] R.-W. Wong, A.-C. Fu, J. Liu, K. Wang, and Y. Xu. Global privacy guarantee in serial data publishing. In *ICDE*, pages 956-959, march 2010.

[30] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proc. VLDB*, Seoul, Korea, Sept. 12-15 2006.

[31] X. Xiao and Y. Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *Proc. SIGMOD*, pages 689-700, New York, NY, USA, 2007.