Query Monitoring and Analysis for Database Privacy - A Security Automata Model Approach

Anand Kumar[1⋆], Jay Ligatti[2], and Yi-Cheng Tu[2]

[1] Teradata Corporation, San Diego CA 92127, USA
anand.kumar@teradata.com
[2] Department of Computer Science and Engineering,
University of South Florida, Tampa, FL 33620, USA
{ytu,ligatti}@cse.usf.edu

Abstract. Privacy and usage restriction issues are important when valuable data are exchanged or acquired by different organizations. Standard access control mechanisms either restrict or completely grant access to valuable data. On the other hand, data obfuscation limits the overall usability and may result in loss of total value. There are no standard policy enforcement mechanisms for data acquired through mutual and copyright agreements. In practice, many different types of policies can be enforced in protecting data privacy. Hence there is the need for an unified framework that encapsulates multiple suites of policies to protect the data.
We present our vision of an architecture named security automata model (SAM) to enforce privacy-preserving policies and usage restrictions. SAM analyzes the input queries and their outputs to enforce various policies, liberating data owners from the burden of monitoring data access. SAM allows administrators to specify various policies and enforces them to monitor queries and control the data access. Our goal is to address the problems of data usage control and protection through privacy policies that can be defined, enforced, and integrated with the existing access control mechanisms using SAM. In this paper, we lay out the theoretical foundation of SAM, which is based on an automata named Mandatory Result Automata. We also discuss the major challenges of implementing SAM in a real-world database environment as well as ideas to meet such challenges.

Keywords: Automata, Access Control, Differential Privacy, Security

1 Introduction

Data is often the most valuable asset to its owner or person whose information is captured in it. Certain information is private and should not be disclosed in any form. Various laws and mutual agreements between parties require the organizations to set strict policies in disclosing certain information. For example, laws

⋆ Author to whom all correspondence should be sent. This work was done at University of South Florida.

such as *Health Insurance Portability and Accountability Act (HIPAA)* mandate the organizations not to disclose personally identifiable information under any circumstances. In the past, enormous efforts have been made to control data access, optimize queries, and process queries efficiently. Little efforts are put to monitor queries to protect against privacy violations occurring through improper data usage [18].

On the other hand, there is the need to find out inherent patterns in the stored data for targeted advertising, marketing, and other business purposes. It is achieved through statistical analysis of the stored data, often done by external analysts. Whether the analysis is done internally or externally, it should follow the aforementioned rules in leaking private information. Thus, privacy-preserving data analysis/mining has become an active field of research and practice. A common practice is to anonymize the data [9] before disclosing to third party analysts. Unfortunately, the anonymous data sets can be combined with information from other sources to extract the private information. In one such example of attacks, users in an anonymized Netflix prize data were identified by using an IMDB movie rating data set [15].

Data protected by mutual agreements and copyright laws needs to be monitored for usages that may affect the businesses. For example, hobbies information of Facebook users should not be combined with Google maps location services. Even if the business has access to both data sets, it should not allow its employees or users to combine them. It may result in violation of privacy as well as affect the data owners financially. Therefore, monitoring data usage becomes an important task for organizations [18].

A number of privacy-preserving policies have been defined and enforced in current database management systems (DBMS) through access control mechanisms (ACM). The main limitation of these ACMs is that they only enforce "black and white" policies; the user is either granted access to the information or completely denied. In the presence of such mechanisms it is challenging to allow third-party analysts to access the database because the ACMs would have to restrict access to private information. Effective analysis is possible if some form of data aggregation, from sources who have private information, is made accessible, without revealing the private information. It is also important to monitor the accesses made by legitimate users. For example, a valid user may access, out of curiosity, important information about important personalities (or celebrities). Such unnecessary accesses should be monitored.

Our vision is to introduce an architecture named security automata model (SAM), shown in Fig. 1, to enforce privacy policies and restrict data usage. An important feature of the SAM architecture is that **we can enforce many types of database privacy policies within a single enforcement architecture.** The SAM model basically enforces ACMs and other declared privacy policies on the data and query results. SAM is a query monitor based on previous work in the area of software security by Ligatti and Reddy, namely, an automata called Mandatory Result Automata (MRA) [11]. All the actions taken by the DBMS for processing input queries and then returning their results are inspected before

presenting the results to the user. This inspection feature of SAM is very useful in monitoring authorized users accessing private information out of curiosity. The results that may generate or help infer confidential information are substituted or perturbed by the monitor through differential privacy (DP) [5].

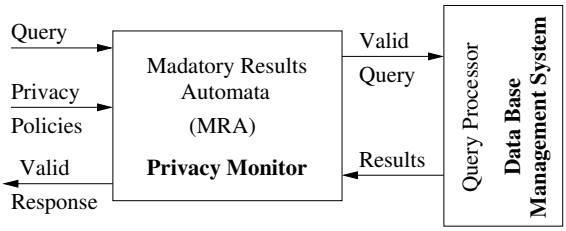

Fig. 1. Architecture: Basic monitor model

SAM works on the basic aggregates supported by state-of-the-art DBMS, while controlling access through ACMs. The data owner can specify policies on these aggregates rather than writing their own interfaces to protect the privacy. This eliminates need to inspect, monitor, and control the programs written by external analysts. The analysts don't have to have any knowledge of special parameters and accuracy requirements, to access the data. The major contributions of this work are:

1. A new privacy enforcement architecture parameterized by various privacy policies.
2. Utilizing SAM to monitor data usage and preserve privacy in databases.
3. Formal model of the architecture using mandatory results automata (MRA).
4. Proof that security mechanisms modeled in our framework can be constructed to enforce any of a large class of privacy policies.

Details of the problem studied and a brief summary of related work are presented in Section 2 and the privacy architecture is discussed in Section 3. A formal model to enforce policies and challenges involved are discussed in Section 4. We conclude our paper in Section 5 with some details of possible future work.

2 Problem Description and Related Work

SAM is designed for protection of data stored in relational database management systems (RDBMS). The RDBMS would incorporate SAM as an additional layer to monitor the data usage and preserve privacy. We envision the following three problems to be addressed using SAM.

1. **Monitoring Users:** Access to database from both unauthorized and authorized users must be monitored. The problem of restricting unauthorized

users falls into the realm of ACMs and hence we do not cover it in our work. Some authorized users may access private data out of curiosity. Such accesses must be monitored and appropriate action be taken to preserve privacy. Thus, SAM should automatically audit authorized users accessing data that should not be accessed during normal operations.

2. **Monitoring Data Usage:** Often business agreements and terms of usage restrict the way in which data is used. For example, the use of maps with location data could be restricted. The users of such data must be monitored and combining one data set with other should not be allowed. In such cases, prohibited operations on data are not allowed.
3. **Enforcing Policies:** Various privacy and data usage policies are expressed as predicates that are enforced by the mechanism implemented in SAM. The database administrator declares the policies to be enforced in a query language. Then, SAM enforces ACMs and declared privacy policies to protect the data.

Even though various policies are enforced on every query submitted by a user, there is need to monitor the queries based on history of data accesses. A sequence of relevant queries may give out private information. Therefore, SAM should enforce policies by looking at the history of queries submitted to the database. It becomes more challenging when multiple users collude to access private data. We look at the architecture of SAM to address these problems.

2.1 Related Work

The problem of security in databases is well studied in the past three decades. The security mechanisms of the past are categorized into four classes [1, 14]: Conceptual Models, Query Restriction, Output Perturbation, and Data perturbation. A detailed survey is presented in [1]. In the recent past an extensible platform for privacy-preserving data analysis has been studied [12]. There have also been some systems built to enforce privacy-preserving policies [16, 13]. These systems have focused mainly on providing database interfaces for the programs of data analysts. The data owner makes settings (specify parameters like privacy budget etc.) so that the programs don't reveal the private information. It is often hard to inspect the bugs and possible privacy breaches in programs written by external agents.

As there are different approaches to preserving privacy in databases, each approach has its own advantages and disadvantages. In our privacy architecture, we have presented a monitoring mechanism that we believe combines the best features of these approaches. It is capable of restricting the queries during normal operation and perturbing the results when it is absolutely necessary. The data owner can define all the privacy policies and input them to the monitor. The privacy model enforces the policies while monitoring external programs for breach in the data privacy. To the best of our knowledge, ours is the first framework for enforcing many types of database privacy policies within a single enforcement architecture.

3 The SAM Architecture

Any database query can be expressed as a relational algebra expression (or **expression** in short) on the values of database table attributes. The set of records R_C whose values satisfy the selection condition C in a given expression is called the **query set**. The basic aggregates used as examples in this model are: averages (AVG), sums (SUM), counts (COUNT), maximums (MAX), and minimums (MIN). The AVG query, and many other statistics, can be computed from the results of these basic aggregate queries. However, the model can be extended to support any type of complex statistical queries.

In this section, we present the basic privacy architecture for databases, using security automata. The runtime policy-enforcement mechanism works by monitoring the queries submitted to the database and the results returned. Runtime mechanisms are quite popular in monitoring database access-control policies, firewalls, operating systems, spam filters, web browsers, etc. We propose a privacy-monitor architecture for DBMS, based on the mandatory results automata (MRA) [11] to enforce the privacy policies in databases, as shown in Fig. 2. The monitor is in-lined with the DBMS, interposing on the input queries and output results. The monitor enforces policies by observing all the input queries and their results. It transforms all queries and results to ensure that the queries processed by the DBMS and the results to be returned to the users are valid. A **valid action** is actually a query or result that satisfies the desired policies of the DBMS. Some of the policies that are enforced by this architecture are discussed in Section 4.

Monitor Input Queries: Every input query goes through the *access control* module. Database access is granted only if the user submitting the query has authorization. Otherwise, the query is rejected. Once authorized, the query is inspected to see if the number of entities involved does not exceed the limit set by the database administrator/owner. It is one of the methods to restrict data access that is possible through a sequence of queries [4]. A history of entities and their attributes is maintained to check the overlapping entities in the input queries. A new entry is made in the log of a *history tracker* whenever a valid input query passes through the *entity check* module.

The aggregate queries are modified into their basic components, before submitting to the DBMS, by the *query rewrite* module. For example, an AVG query is split into COUNT and SUM queries. The results of these component queries are monitored to protect privacy.

Monitor Query Results: The results of component queries are inspected by different modules to detect possible situations in which the private information could be revealed. For example, the result of COUNT component of the AVG query is inspected by the *set size restricter*. This module restricts queries from disclosing results with very few or just one tuple [7]. A valid result is inspected by the *query tracker* module for presence of any tracker. A tracker is a query with auxiliary conditions padded to the original conditions that are invalid (or

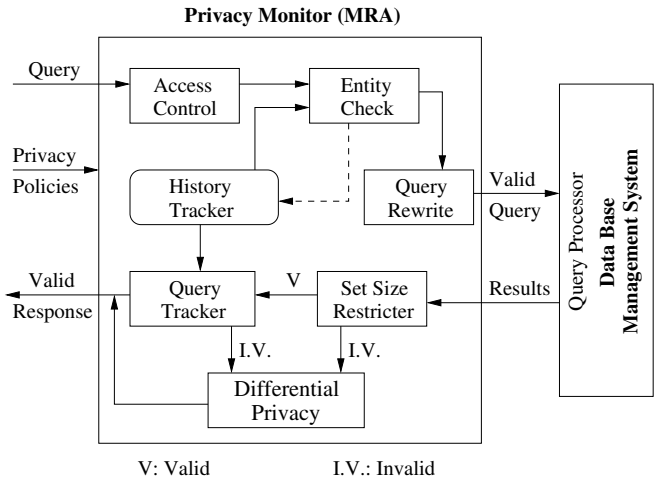

Fig. 2. Architecture: Detailed MRA

not allowed by the DBMS) [3]. A valid response from the query tracker returns results to the user. If the query set size restricter or the query tracker detect any possibility of revealing private information, the *differential privacy* module perturbs the results. Output perturbation [1, 14] is necessary to make sure that nothing is inferred by the user when the results are delivered. We enforce this requirement through differential privacy [5, 12].

Definition 1. Differential privacy [5]**:** *Any randomized algorithm f is said to provide ϵ-differential privacy if for all data set instances $G, H \in D^n$ differing in at most one record, and any set of possible outputs $S \subseteq Range(f)$,*

$$\Pr[f(G) \in S] \leq e^{\epsilon} \Pr[f(H) \in S]$$

where D is the domain of data records, and e is Euler's number $(2.71\ldots)$.

The parameter ϵ is called the *privacy budget* and is a basic requirement for enforcing differential privacy. The accuracy of the results is inversely proportional to the budget value, and hence the privacy guarantee. The database owner can fix the budget requirement through policy predicates that are enforced by SAM. Thus, the analysts do not have to specify any budget or accuracy requirements. This gives the data owner power to control the accuracy precisely to protect the private information.

The history tracker records information about the attribu-tes that are answered in the past queries. The overlap of attributes and results in a sequence of queries is computed and stored for future queries [4]. Whenever a query result violates tracker policy, the monitor applies differential privacy, making it a valid query and protecting the information. The size of the stored history can

be managed by specifying policies on the history tracker. Policies can be defined to discard the stored information whenever the database is updated. Thus, the history tracker plays an important role in protecting private information that could otherwise be accessed through a sequence of queries.

Complex Aggregates: It is often necessary for a DBMS to support user-defined, complex, aggregate queries, such as standard deviation, cluster center, etc. The private information that can be accessed through sequences of such complex queries and other standard aggregates needs to be protected. All the aggregate queries are modified into their basic components. The COUNT query is always inserted before every complex aggregate by the monitor, to inspect the number of unique records involved in the aggregate computation. When the number does not satisfy policies of any of the modules, the monitor should take appropriate action to apply differential privacy.

An interesting feature of using MRA for enforcing privacy is their ability to support many policies, such as differential privacy, access control, mechanism to sanitize results, etc. [11]. Policies can be specified in combination to strengthen monitoring process. For example, set-size-restricter and differential-privacy modules, shown in Fig. 2 can work together to decide when to perturb the results. It is also possible to plug-in new modules in this architecture to enforce complex policies, enabling users to tune the level of control. We look into details of the formal model in the following section.

4 Policy Enforcement

The power of SAM is in enforcing various privacy and data use policies. It has various modules that can be utilized to enforce policies effectively. We have identified three classes of policies that are necessary to protect privacy and data usage: query control, result control, and access control.

Query control policies identify various entities and attributes involved in the queries and check for violations of predefined conditions. Number of entities present in the queries can be monitored to identify any privacy violations. Restrictions related to allowed aggregates and joins can also be enforced.

Result control policies control the amount of data accessed by restricting the number of records returned in the query results. A malicious user may issue sequence of queries to access as much data as possible. However, SAM's *history tracker* monitors such activities and denies access. Often it is important to allow full access to the data for processing complex aggregates. In such cases, to preserver privacy, differential privacy is applied to obfuscate the results.

Access control policies are enforced by any standard RDBMS. However, in order to monitor authorized users accessing private data (e.g. celebrity), additional information is required. SAM can store such additional information in a SAM-specific private database and use it to monitor such activities.

It can be noticed that the requirements of data use restrictions fall into all of the above mentioned policy categories. Therefore, the problem of monitoring data use is challenging. Any new policies that can be enforced by SAM fall into one of the above mentioned categories. It is also important that there is need for a query language to define such policies. An example policy `P1` to dis-allow entities containing user location and hobbies is shown below.

```
P1: RESTRICT ENTITIES UserLocation, UserHobbies
P2: CHECK HISTORY ( SELECT Entities, Attributes
                    FROM History )
    WHERE Query.Entities IN History.Entities AND
          Query.Attributes IN History.Attributes
```

Policy `P2` checks the history of queries and returned results to check if any privacy violations have occurred. Specifying policies in a user friendly language is one of the challenges in enforcing data use restrictions and preserving privacy. Size of history could pose performance issues in efficient query evaluation. In Section 4.4 we discuss few possible solutions to overcome these difficulties.

4.1 Formal Model

The DBMS can be abstractly defined in terms of the queries it can process (actions) and the possible results of those queries (results). Any request from users of the DBMS is an aggregate query to be executed. The response to the queries are results computed by the system. We represent the set of actions on a database using the metavariable A, and results to these actions using R. An action can be a query submitted to DBMS. Both sets A and R are nonempty, possibly countably infinite, and $R \cap A = \emptyset$. An event in the DBMS is a query or a result, and it is denoted using E, where $E = A \cup R$.

The sequence of submitting queries to the DBMS and obtaining their results is called an **execution**. Each execution x is defined as a sequence of events with event set E. The set of all finite-length executions possible is denoted by E^*. A session is modeled as an execution. If the session terminates then the execution is finite; otherwise the execution is infinite. Only one infinite-length execution is allowed per user in our system. We denote the empty execution as ε. When an execution x' follows another execution x we denote it by concatenation, $x; x'$. When x is a prefix of x' it is written as $x \preceq x'$. Throughout this paper we abbreviate the formula $\forall x' \in E^* : x' \preceq x \Rightarrow F$ as $\forall x' \preceq x : F$. Finally, we notate the final query in execution x as Q_x.

An MRA M is a quadruple denoted as (E, S, q_0, δ), where E is the event set over which M operates, S is the set of possible states (finite or countably infinite) of M, q_0 is the initial state, and δ is a transition function of the form $\delta : S \times E \to S \times E$. An MRA in its current state takes an event, either an input query or result of a query, and transitions to new state and produces an output. The output can be a valid query to be executed by the query processor, or the result to be presented to the user (Fig. 2). We write $M \Downarrow X$ when M **produces**

an execution $X \subseteq E^*$ for input events that match the sequence of input events in X. Section 4.2 will define enforcement as requiring an MRA to produce exactly those executions that the desired policy allows.

MRA treat all actions as synchronous i.e., they finish processing the input action and return results before accepting the next input for processing. Their ability to transform the results of actions is novel and crucial for enforcing policies. This behavior is essential in DBMS and matches our requirement of enforcing privacy-preserving policies.

4.2 Privacy Policies

The SAM is able to enforce any policies that can be defined on the DBMS. In this section we look into different policies that are to be enforced on the queries. The important aspect of the MRA-based architecture is that the MRA can adapt itself to the changing queries over time by enforcing appropriate policies. The policies listed here are well studied in the literature [1, 2, 12], but have never been enforced collectively for protecting privacy.

A **policy** is a predicate defined over executions. A session (execution) $X \in E^*$ is said to satisfy policy P iff $P(X)$.[3]

With the definitions of policies and MRA producing executions we are now ready to define what it means for an MRA to enforce a policy. Definition 2 says that an MRA M enforces a policy P exactly when the set of executions M produces equals the set of execution P allows.

Definition 2. *An MRA M on a system with event set E* **enforces** *policy P iff*

$$\forall x \in E^* : (M \Downarrow x) \iff P(x)$$

We next describe different policies that can be enforced by MRA to protect privacy.

Query and Result Control Policies: In this category of privacy policies, the results returned by aggregates depend on the types of attributes involved in the queries. If the attributes involved carry private information, M should enforce the control policies. We examine broad categories of policies on aggregates that M is able to enforce to protect privacy.

Definition 3. *Any execution $x \in E^*$ is said to satisfy the* ***query-set-size*** *restriction policy iff there exists a predicate P_s such that,*

$$P_s(x) \iff \forall x' \preceq x : K \leq |R_{x'}| \leq N - K$$

where $R_{x'}$ is the query-set (result) of query $Q_{x'}$, N is the number of records in the database, and the value of K is restricted by $0 \leq K \leq N/2$.

[3] Technically, these policies are called "properties" in the literature on formal security models [10, 11, 17].

Query-set-size policies restrict the size of the query set returned by the query processor, to control the information disclosed to users [7]. Such policies can prevent results with very few or just one tuple. The set-size-restricter module of SAM enforces this property. However, when two queries are executed consecutively to get two query-sets G and H respectively, the privacy policy can be violated, as the set difference $|G - H| = 1$. Hence, additional measures should be taken to protect private data.

Definition 4. *Any execution $x \in E^*$ is said to satisfy the **query-set-overlap** restriction policy iff there exists a predicate P_o such that,*

$$P_o(x) \Longleftrightarrow \forall x' \preceq x,\ \forall x'' \preceq x : \big(Q_{x'} \neq Q_{x''} \Longrightarrow |M_{x'} \cap M_{x''}| \leq O\big)$$

where $M_{x'}$ is the set of entities involved in $Q_{x'}$ (and similarly $M_{x''}$ for $Q_{x''}$) and O is the restricted number of overlapping entities.

In this policy, the number of entities that overlap in successive queries are restricted [4]. When a sequence of queries is executed, the policy is either satisfied or not based on the size of the overlap. Given a minimum query set size K and the number of entity overlaps O allowed in successive queries, the minimum number of queries required to compromise is $1 + (K - 1)O$. Our monitor M can be made to look for this number to secure the data. The entity-check module of SAM is capable of enforcing this policy.

A restricted query result can be calculated with the help of two kinds of trackers [3]. One, using a general tracker when the query-set-size $K \leq N/4$. A policy to preserve privacy in the presence of a tracker can be enforced with the help of set-size restricter in M by changing the set-size conditions in Definition 3 to

$$P_s(x) \Longleftrightarrow \forall x' \preceq x : 2K \not\leq |R_{x'}| \not\leq N - 2K$$

The second type of tracker, called a double tracker, can reveal private information when the query-set-size $K \leq N/3$. This can be prevented with the help of the query tracker in M enforcing the desired tracker policy.

Definition 5. *Any execution $x \in E^*$ is said to satisfy the **tracker** policy iff there exists a predicate P_t such that,*

$P_t(x) \Longleftrightarrow \forall x' \preceq x,\ \forall x'' \preceq x :$

$$\left(Q_{x'} \neq Q_{x''} \Longrightarrow \begin{pmatrix} C_{x'} \cap C_{x''} = \emptyset \\ \vee\ K \not\leq |R_{x'}| \not\leq N - 2K \\ \vee\ 2K \not\leq |R_{x''}| \not\leq N - K \end{pmatrix}\right)$$

where $C_{x'}$ and $C_{x''}$ are the selection conditions of the queries $Q_{x'}$ and $Q_{x''}$ respectively, N is the number of records in the database, and the value of K is restricted by $0 \leq K \leq N/3$.

These trackers speculate the privacy compromise by computing all possible combinations of the table attributes that may be queried in the future to get answers to restricted queries. The "query tracker" and "history tracker" modules of M are responsible for enforcing tracker policies. Any algorithm to track quires can be plugged-in.

Differential Privacy Policies: The data owner can restrict the access and accuracy of user queries to the DBMS using differential privacy policies. The idea behind this mechanism is to allow user queries to execute on the actual data, but the results are perturbed before being delivered to the user. The perturbation is a function (differential privacy) $f(R_x)$ on the results R_x of any query $x \in E^*$. The important goal of perturbation is to prevent disclosing the correct value(s) of the result(s). Perturbation may include rounding up (or down) of the resulting values or adding (or subtracting) some pseudo-random numbers [7].

Definition 6. *Any execution $x \in E^*$ is said to satisfy the **differential privacy** policy iff there exists a function f satisfying Definition 1 such that,*

$$P_d(x) \Longleftrightarrow \forall x' \preceq x : R_{x'} = f(R_{x'}^{orig})$$

where $R_{x'}$ is the query-set (result) of $Q_{x'}$ and $R_{x'}^{orig}$ is the original result the DBMS returned for $Q_{x'}$.

The use of output perturbation satisfies the differential privacy policy, as the actual results are never disclosed to the users. The differential privacy function f perturbs results of each query differently, so that inference of private information using multiple queries is impossible.

The advantages of having differential privacy enforcement in SAM is that the analysts do not have to have knowledge about the budget requirements and accuracy needs. The data owner can set exact accuracy and budget requirements to restrict access to private data. Thus, analysts do not have to invoke special application program interfaces to access data through differential privacy mechanisms.

Access-Control Policies: There has been a tremendous amount of work done in enforcing access control policies. Current DBMSs allow users to access different parts of the data base by enforcing ACMs. The SAM can also enforce these policies, as it is based on MRA, which are capable of enforcing arbitrary access-control policies by monitoring and building a history of entire sessions.

The SAM can take advantage of existing ACM in the DBMS to enforce access-control policies. The underlying MRA M has to request permission from the DBMS before submitting its executions. The request is actually a verification of permission of the current query. When the DBMS grants permission, the MRA can take the action of the input query. This removes the burden of implementing ACMs in SAM. The existing DBMS also does not require any changes.

4.3 Model Properties

In the previous text, we discussed different policies that can be enforced on database queries. Now we need an enforcement mechanism that SAM can follow to preserve privacy. In this section, we show how to enforce the privacy policies defined above by constructing MRA and proving that these MRA can

enforce the policies. Given any query-set-size, query-set-overlap, query-tracker, or differential-privacy policy P, Theorem 1 shows that an MRA M can enforce P and Theorem 1's proof shows how to construct such an M.

Theorem 1. *For all policies P such that P is a query-set-size policy, query-set-overlap policy, query-tracker policy, or differential-privacy policy on a system with event set E, there exists an MRA M such that M enforces P.*

Proof. Given any such policy P, we show how to construct an MRA $M = (E, S, q_0, \delta)$ that enforces P on all executions $x \in E^*$. Note that δ is a partial function when δ is undefined on a given input, the MRA cannot transition, so the system effectively halts.

If P is a query-set-size restriction policy parameterized by N and K as in Definition 3, let $M = (E, \{0,1\}, 0, \delta)$, where δ is:

$$\delta(q,e) = \begin{cases} (0,e) & \text{if } q = 0,\ e \neq \mathit{query} \\ (1,e) & \text{if } q = 0,\ e = \mathit{query} \\ (0,e) & \text{if } q = 1,\ K \leq |e| \leq N - K \end{cases}$$

M enforces P because both produce/allow exactly those executions in which all query results r satisfy the $K \leq |r| \leq N - K$ constraint. Hence, the sets of executions produced by M and allowed by P are equal.

If P is a query-set-overlap restriction policy parameterized by O as in Definition 4, let $M = (E, E^*, \varepsilon, \delta)$, where δ is:

$$\delta(q,e) = \begin{cases} (q,e) & \text{if } e \neq \mathit{query} \\ (q;e,e) & \text{if } e = \mathit{query},\ q = e_1; e_2; \ldots; e_n \text{ and} \\ & \forall i \in \{1,\ldots,n\}\ :\ |M_e \cap M_{e_i}| \leq O \end{cases}$$

M enforces P because both produce/allow exactly those executions in which all queries only contain entity sets that never overlap previous queries' entity sets in more than O elements. Hence, the sets of executions produced by M and allowed by P are equal.

If P is a query-tracker policy parameterized by N and K as in Definition 5, let $M = (E, E^*, \varepsilon, \delta)$, where δ is:

$$\delta(q,e) = \begin{cases} (q,e) & \text{if } e \neq \mathit{query} \\ (q;e,e) & \text{if } e = \mathit{query},\ q = e_1; e_2; \ldots; e_n \text{ and} \\ & \forall i \in \{1,\ldots,n\}\ :\ (C_e \cap C_{e_i} = \emptyset \text{ or} \\ & K \not\leq |R_e| \not\leq N - 2K \text{ or} \\ & 2K \not\leq |R_{e_i}| \not\leq N - K) \end{cases}$$

M enforces P because both produce/allow exactly those executions in which all queries Q satisfy at least one of the following three constraints: (1) Q has no projection-condition attributes in common with previous queries, (2) $K \not\leq$

$|R_e| \not\leq N - 2K$, or (3) all the previous query results r satisfy the constraint $2K \not\leq |r| \not\leq N - K$. Hence, the sets of executions produced by M and allowed by P are equal.

If P is a differential-privacy policy parameterized by perturbation function f as in Definition 6, let $M{=}(E, \{0,1\}, 0, \delta)$, where δ is:

$$\delta(q,e) = \begin{cases} (0,e) & \text{if } q = 0,\ e \neq \mathit{query} \\ (1,e) & \text{if } q = 0,\ e = \mathit{query} \\ (0,f(e)) & \mathit{if}\ q = 1 \end{cases}$$

M enforces P because both produce/allow exactly those executions in which all query results are perturbed by the perturbation function f. Hence, the sets of executions produced by M and allowed by P are equal.

Hence, an MRA can enforce all query-set-size restriction, query-set-overlap restriction, query tracker, and differential-privacy polices.

4.4 Challenges

The MRA based monitor can secure the database by enforcing the policies defined above. In this section we discuss some of the challenges in enforcing these policies with MRA, and possible solutions.

Infinite-Length Executions: An infinite-length execution is one that never terminates. For simplicity, the definitions of privacy-preserving policies in Section 4.2 only consider finite-length executions. However, these definitions can be generalized to infinite-length executions straightforwardly by placing the same sorts of query-result constraints on every query and result in every execution (including those of infinite-length). All the modules of SAM are capable of monitoring sequences of infinite-length queries, without requiring any changes to their functionalities.

Colluded Attacks: As explained before, MRA is a synchronous automata allowing only one input event and producing a result for that event before processing the next. Thus, concurrent events can't be processed by the MRA. This leads to a possible attack in which multiple users collude and share their results. Since the MRA assumes one infinite-length execution per user, it can protect the private information from only one user, not colluded users.

A possible solution is to assume the queries from all users as events of a single infinite-length execution. Queries from all users can be serialized, possibly based on arrival timestamps. Thus, completely eliminating the possibility of compromise from colluding users. Another direction to eliminate the risk from colluded attacks is taking advantage of differential privacy parameters. The database administrators can specify different types of noise to be added to different users, so the users can't extract any private information by colluding. There could be some limitations on the number of different types of noises that can be generated at run time to a large number of users.

User-Defined Aggregates: It is often necessary to allow the analysts to write their own programs to analyze the data. The data owner can specify differential privacy policies on such programs. The MRA can enforce these policies directly without requiring any changes to the underlying architecture. When different types of aggregates are supported, specifying proper policies becomes a challenging task. The policies should be specified such that the privacy is preserved. It is analogous to specifying access control policies in standard databases.

Performance: Maintaining history of queries and results could impact overall performance of SAM. Logging during query evaluation and log-access for policy enforcement could overload SAM, impacting overall performance. Enforcing policies online as well as offline could alleviate the performance problems. Online setting utilizes only recent past of the history, while offline setting audits to check if any policy violations have occurred due to recent queries. High performance computing strategies can also be applied to utilize large portion of the history for faster processing. The problem of leveraging such strategies is another interesting research challenge.

History tracking is an interesting feature of SAM, which can leverage lineage of queries to preserve privacy and restrict data usage. Query lineage can be utilized for data usage audits [6, 8]. It becomes an effective tool to monitor curious (authorized) users.

5 Conclusions and Future Work

We have presented our vision of a privacy architecture, called SAM, which can enforce various privacy-preserving policies and restrict data usage in databases. MRA are constructed to enforce query, result, and access control policies. An interesting feature of SAM is that it allows accurate query results as long as the privacy is preserved. It can be set up to begin enforcing differential-privacy policies when none of the other policies are satisfied. The data owners can specify policies and input to SAM instead of writing their own database interfaces to preserve privacy. This also alleviates the need for analysts to specify any special parameters to access the database.

The MRA construction method can be followed to enforce the policies defined in this work. We believe that the MRA can be an independent component of the database, without affecting other database modules. Much has to be done to refine the SAM design and ensure effective and efficient implementation of the architecture, as discussed in Section 4.4. Here we want to emphasize the very interesting research topic of studying the possibility of utilizing these database modules to improve the performance. SAM can be implemented as part of the DBMS to monitor the results of different database operators. Monitoring operators inside DBMS would eliminate the need for the query-rewrite module in SAM, as the partial results can be monitored even before computing the final result. We believe that the formal architecture presented in this paper is capable of enforcing a wide range of database policies.

Acknowledgements: This project is supported by a grant (No. R01GM086707) from the National Institutes of Health (NIH), USA.

References

1. N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.
2. R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving OLAP. In *Proceedings of the Intl. Conf. on Management of Data*, SIGMOD, pages 251–262, 2005.
3. D. E. Denning and J. Schlörer. A fast procedure for finding a tracker in a statistical database. *ACM Transactions on Database Systems*, 5(1):88–102, 1980.
4. D. Dobkin, A. K. Jones, and R. J. Lipton. Secure databases: protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, 1979.
5. C. Dwork. Differential privacy: a survey of results. In *Proceedings of the Intl. Conf. on Theory and Applications of Models of Computation*, TAMC, pages 1–19, 2008.
6. D. Fabbri and K. LeFevre. Explanation-based auditing. *Proc. VLDB Endow.*, 5(1):1–12, 2011.
7. I. P. Fellegi and J. J. Phillips. Statistical confidentiality: Some theory and application to data dissemination. *American Economic Society Measures*, 3(2):101–112, 1974.
8. R. Hasan and M. Winslett. Efficient audit-based compliance for relational data retention. In *Symposium on Information, Computer and Communications Security*, pages 238–248, 2011.
9. C. Kushida, D. Nichols, R. Jadrnicek, R. Miller, J. Walsh, and K. Griffin. Strategies for de-identification and anonymization of electronic health record data for use in multicenter research studies. *Medical Care*, 50:S82–S101, 2012.
10. J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security*, 12(3):1–41, 2009.
11. J. Ligatti and S. Reddy. A theory of runtime enforcement, with results. In *Proceedings of the 15th European conference on Research in computer security*, pages 87–100, 2010.
12. F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the Intl. Conf. on Management of data*, SIGMOD, pages 19–30, 2009.
13. P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. Gupt: privacy preserving data analysis made easy. In *Proceedings of the Intl. Conf. on Management of Data*, SIGMOD, pages 349–360, 2012.
14. K. Muralidhar, D. Batra, and P. J. Kirs. Accessibility, security, and accuracy in statistical databases: the case for the multiplicative fixed data perturbation approach. *Management Science*, 41(9):1549–1564, 1995.
15. A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the Symposium on Security and Privacy*, S&P, pages 111–125, 2008.
16. I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: security and privacy for mapreduce. In *Proceedings of the Conference on Networked Systems Design and Implementation*, NSDI, pages 20–20, 2010.
17. F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

18. P. Upadhyaya, N. R. Anderson, M. Balazinska, B. Howe, R. Kaushik, R. Ramamurthy, and D. Suciu. Stop that query! the need for managing data use. In *Conf. on Innovative Data Systems Research*, 2013.