Towards Robustness in Query Auditing

Shubha U. Nabar* Bhaskara Marthi# Krishnaram Kenthapadi* Nina Mishra*
Rajeev Motwani*

ABSTRACT

We consider the *online query auditing problem* for a database containing private information about individuals: given a sequence of queries posed by an attacker, when should queries be denied to prevent privacy breaches. Historically, all research in auditing has focused on static datasets and known algorithms do not work in the presence of updates. We consider a new auditing problem where records can be *inserted*, *deleted* or *modified* and obtain algorithms for auditing sum queries, max queries and bags of max and min queries in this scenario under the classical notion of compromise.

In [11], a new probabilistic notion of compromise was proposed due to the limitations of classical compromise. Not much is known about auditing different kinds of queries under this definition and we present new algorithms for auditing max queries and bags of max and min queries under probabilistic compromise.

We conclude with a study of a particular dimension of the *utility* delivered by an auditor and obtain initial results for the utility of sum auditing under classical compromise. We show a positive result for large datasets - in a sequence of random queries, the number of queries that will be answered before even the first denial occurs is a constant fraction of the size of the dataset. This implies that the auditor will not return answers that are riddled with denials.

1. INTRODUCTION

Consider a database containing sensitive information about individuals, such as a company database containing salaries of its employees or a hospital database containing medical records of its patients. There is often a need to enable the computation of useful statistics from such databases while protecting the privacy of individuals. For example, a statistician may want to know the average salary of all the female employees in a company, but he cannot be allowed to glean the salary of any one female employee in particular.

Let $X = \{x_1, \ldots, x_n\}$ be a dataset of n records. Each x_i is the sensitive attribute value of the ith individual in the database and in our scenario is taken to be real-valued from a bounded or unbounded range. A statistical query $q = (Q, f)$ specifies a subset of the data entries $Q \subseteq \{1, \ldots, n\}$ and a function f (such as sum, max or median). The result - $f(Q)$, is f applied to the subset of entries $\{x_i \mid i \in Q\}$. In this paper, Q is referred to as the *query set* of q.

We consider the *online query auditing problem:* Given a sequence of queries $q_1, \ldots, q_{t-1}$ that have already been posed, answers $a_1, \ldots, a_{t-1}$ to these queries that have already been supplied and a new query q_t, should q_t be truthfully answered or should it be denied to prevent a privacy breach. Here, each of the previous answers a_i, is either the true answer to the query $f_i(Q_i)$, or a denial. In this paper, we attempt to enhance the robustness of the notion of online auditing by exploring the scenario of non-static databases, the new notion of compromise introduced in [11] and the heretofore unexamined dimension of *utility*.

The auditing problem may be viewed as a game between the auditor and an attacker. The auditor monitors a sequence of queries posed by an attacker and denies queries whenever answers to these and previous queries may be stitched together to glean information about any one individual. This brings us to the definition of compromise. In most previous work, a privacy breach occurs when the sensitive attribute value of any individual can be uniquely determined. We call this the *classical notion of compromise*. In addition, the authors in [11] introduce a *probabilistic notion of compromise* for bounded range data where a significant change in the attacker's confidence about the range of a data point constitutes a privacy breach.

The notion of *simulatable auditing* was introduced in [11]. Simulatability in this scenario means that an attacker should be able to simulate the auditor's decision to answer or deny. This requires that the auditor must not look at the answer to the current query when deciding. He may only look at past queries and corresponding answers supplied to the attacker. The need for simulatability arose out of the authors' discovery that denials based on the answer to the current query can leak considerable information. The auditors we look for should thus be online *and* simulatable.

A naive simulatable solution to the online auditing problem would be to simply deny all queries. This is not a very satisfying solution as it does not provide much utility to the user. To the best of our knowledge, no previous work has attempted to quantify the utility of an auditing scheme. In this paper, we consider a particular dimension of utility and obtain initial utility results for the auditing algorithm for sum queries described in [4, 11]. The result pertains to classical compromise and is a positive result for large datasets.

We also consider updates to a database and changes that must be made to existing auditing algorithms to accommodate these. As far as we know, all previous auditing algorithms work under the assumption that no changes are ever made to the database and this is clearly an unrealistic assumption to make - datasets are in practice, quite dynamic.

*{sunabar, kngk, nmishra, rajeev}@cs.stanford.edu
#bhaskara@cs.berkeley.edu

We consider sum queries, max queries and combinations of max and min queries under classical compromise when updates can be made to the database. In doing so, we introduce a new algorithm for auditing bags of max and min queries where none was known before.

The next dimension in which we attempt to further our understanding of online auditing is the new probabilistic definition of compromise introduced in [11] to overcome the shortcomings of the classical definition. Here we introduce new simulatable algorithms for max queries and combinations of max and min queries. No algorithms were previously known for these cases.

The remainder of this paper is organized as follows. We discuss related work and preliminaries in Section 2. A more general overview of the field can be found in [1]. In Sections 3, 4 and 5 we discuss our new results for updates, probabilistic compromise and utility respectively. This is followed by conclusions and future work in Section 6.

2. BACKGROUND

2.1 Related Work

The online auditing problem was first studied in [7] and [14]. Here the authors look at the sum auditing problem in particular and restrict the sizes and pairwise overlaps of queries that can be posed to prevent privacy breaches. Under this scheme, if the size of each query set is restricted to be at least k and if each pair of query sets is allowed to overlap in at most r elements then $(2k-(l+1))/r$ distinct queries can be answered in total where l is the number of x_is known to the attacker beforehand. So if $k = n/c$ for some constant c and $r = 1$ then after only a constant number of distinct queries, the auditor would have to deny all further queries since there are only about c queries where no two overlap in more than one element. This motivates a search for auditing schemes that could provide greater utility.

In [3], the authors consider the *offline auditing problem* for sum, max, max-and-min and sum-and-max queries. In the offline version of the problem, we are given a sequence of queries $q_1, \ldots, q_t$ that have already been posed and truthfully answered and are required to determine whether compromise has already occurred. The sum-and-max auditing problem was shown to be NP-hard while efficient algorithms were derived for all the others.

In [10], the authors consider the problem of auditing subcube queries where queries are specified by a string of 0s, 1s and *s (don't cares). The elements to be summed up are those whose non-private attribute values match the pattern specified by the query string. The authors in [12] consider the boolean auditing problem where the attribute values to be protected are boolean and the queries are sum queries. They also provide a max-auditor for real-valued data. These results once again pertain to the offline auditing problem.

In [11], the online auditing problem is looked at again. The authors illustrate how denials that depend on the answer to the current query can leak information and introduce the notion of simulatability to tackle this. They provide simulatable algorithms for auditing sum queries and max queries under classical compromise. In addition, they introduce a probabilistic notion of compromise and provide an algorithm for auditing sum queries over real-valued data drawn uniformly from a bounded range under this notion.

2.2 Preliminaries

Classical Compromise: Under classical compromise, a compromise occurs if any one private data point can be uniquely determined, i.e. in all datasets with answers $a_1, \ldots, a_t$, to queries $q_1, \ldots q_t$, some data point x_i would be the same.

Probabilistic Compromise: As noted in [2, 10, 5, 13], the classical notion of compromise has certain limitations:

- Even though a private value may not be uniquely determined, it may still be deduced to lie in a tiny interval or even in a large interval with a heavily skewed distribution - and some may consider this to be compromise.
- There are situations when no query would ever be answered. For instance in the case of sum queries over boolean data, in the interest of simulatability, no sum query would ever be answered, since there could be a data set where the answer to the query could be 0 implying that every single x_i in the query set is 0.

This new notion of privacy defined in [11] arose out of the limitations of classical compromise for bounded range data and essentially models the change in the attacker's confidence about the value of a data point. It bounds the ratio of the posterior probability that a value x_i lies in an interval I to the prior probability that x_i lies in I. Given an arbitrary dataset $X \in [0,1]^n$, queries $q_1, \ldots, q_t$, answers to these queries $a_1, \ldots, a_t$ and predefined security parameters, λ and α,

$$S_{\lambda,i,I}(q_1, \ldots, q_t, a_1, \ldots, a_t) =$$
$$1 \quad if\ (1-\lambda) \leq \frac{Pr(x_i \in I | \wedge_{j=1}^{t} f_j(Q_j) = a_j)}{Pr(x_i \in I)} \leq 1/(1-\lambda)$$
$$0 \quad otherwise$$

Let $\mathcal{I}$ be the set of intervals $[\frac{j-1}{\alpha}, \frac{j}{\alpha}]$ for $j = 1, \ldots, \alpha$.

$$S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_t) =$$

$$\bigwedge_{i \in [n], I \in \mathcal{I}} S_{\lambda,i,I}(q_1, \ldots, q_t, a_1, \ldots, a_t)$$

$S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_t)$ thus evaluates to 1 if every single data point, x_i is "safe" with respect to every single interval, $I \subset [0,1]$. The attacker poses a query, q_t in each round, t for up to T rounds. The auditor can choose to deny a query and the attacker wins if $S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_t) = 0$ in some round. This is the (λ, α, T)-privacy game and an auditor is $(\lambda, \delta, \alpha, T)$-private if for any attacker, A:

$$Pr[\text{A wins the } (\lambda, \alpha, T) - \text{privacy game}] \leq \delta$$

Simulatable Auditing: The concept of simulatable auditing was introduced in [11]. In the context of online auditing, simulatability implies that the attacker should be able to predict when a denial is about to occur. The need for simulatability can be illustrated with a simple example. Consider an attacker who asks for $\max\{x_a, x_b, x_c\}$ and is returned the answer 9. Later the attacker asks for $\max\{x_a, x_b\}$. If the answer to this query is less than 9 then the attacker can determine that x_c must be 9. If however, the answer is exactly

9, answering this query would not leak information. In this situation, if the auditor does look at the answer to the query when deciding to deny - a denial would immediately imply that x_c must be 9 and privacy is breached. The auditor should thus not look at the true answer to a query when deciding. In the case of classical compromise this implies that the auditor must consider if there is any possible answer to the current query, consistent with past queries that could cause compromise. In the case of probabilistic compromise this implies that the auditor must determine if compromise would occur in a large fraction of datasets drawn from the original distribution and consistent with past queries.

Synopsis Computing Blackbox $\mathcal{B}$: In Sections 3.3, 4.1 and 4.2 we use a blackbox in our algorithms that is introduced in [3]. The blackbox is used for the offline auditing of max queries over real-valued data containing no duplicates. The blackbox takes as input a set of max queries and answers and converts them to a synopsis $B_{\max}$. $B_{\max}$ contains predicates of the form $[\max(S_i) = M_i]$ and $[\max(S_j) < M_j]$ where S_i and S_j are subsets of the dataset and $S_i \cap S_j = \emptyset$. Thus every x_i can occur in at most one such predicate and the synopsis is of size $O(n)$. The authors show that it suffices to consider just these query sets in detecting compromise instead of the entire sequence of queries that has been posed. The blackbox can similarly be given a set of min queries and their answers and it outputs a synopsis $B_{\min}$ containing predicates of the form $[\min(S_i) = m_i]$ and $[\min(S_j) > m_j]$ where S_i and S_j are disjoint subsets of the dataset.

3. NON-STATIC DATABASES

To the best of our knowledge, existing auditing schemes for sum and max queries only work for static datasets. We consider extending these algorithms to handle updates to the database as well. As queries are likely to be posed over a period of time, we cannot expect the records in a database to remain unchanged throughout. For example, consider the company database containing salaries of its employees. Three types of changes can be made to the database:

- Insertions: When new employees join the company.
- Deletions: When employees leave the company.
- Modifications: When employees gets promoted or demoted.

A robust query auditing scheme should work in the presence of updates, i.e. provide answers to as many queries as possible while ensuring that no individual's privacy is breached. We first need to understand the level of privacy desired when updates occur. Should we protect the very fact that a database record was updated? Is it fine to ignore the privacy of deleted or modified records? In the case of modifications, should we protect the change in the value of the record in addition to its past and current values? What should we protect when there are several modifications to the same record? In this paper we assume that the attacker has access to the fact that a record was either inserted, deleted or modified and we simply wish to protect all past and present values of any record as well as individual modifications made to them. This is a reasonable assumption to make since an attacker can easily observe that an employee joined/left the company or was promoted.

We describe schemes for handling updates in the case of sum queries, max queries and bags of max and min queries under the classical notion of compromise.

3.1 Sum Auditing with Updates

Existing algorithm: We consider sum queries over real-valued data taken from an unbounded range. The existing algorithm for sum queries can be found in [4, 11]: each query is expressed as a row in a matrix with a 1 for every record that is accessed by the query and a 0 for every record that is left out. The columns of the matrix thus correspond to the records and the rows to the queries. The row corresponding to a query is called the "query vector". The matrix is maintained in diagonalized form through a series of elementary row-space-preserving row operations and column changes. If the diagonalized form of the matrix contains a row with a 1 in only one column and 0s in all others then some element can be uniquely determined. When a new query is posed, the auditor checks to see if the new query vector is already in the vector space spanned by the rows of the matrix in $O(nm)$ time, where m is the rank of the matrix. If it is, then the query is answered (the new query vector is not added to the matrix). If not, then adding the new query vector and rediagonalizing the matrix also takes $O(nm)$ steps.

Consider using this auditing scheme, as is, in the presence of updates. The result would be an overly conservative auditor - one that denies even when unnecessary. For example, suppose a user asks for $x_a + x_b + x_c$ and this is followed by an update to the database where x_a is updated to $x_a + \Delta$. If the attacker now asks for $x_a + x_b$, the query will be denied since the auditor will believe that x_c can be determined. In reality however, what the user can determine is $x_c + \Delta$ and there is in fact no privacy breach. While the existing scheme thus continues to ensures the privacy of individuals, it reduces the utility provided to the user since denials now occur even when answering queries would definitely not leak information. We therefore look for an auditing scheme that works with updates as well.

Updates: The method we propose is a simple extension to the existing algorithm and involves only just modifications to the query matrix that is maintained. Insertions and deletions of records can be handled easily. When a new record x_j is inserted, we introduce a new column in the query matrix corresponding to x_j (full of zeros since no previous query involved x_j). On the other hand, when a record x_j is deleted, we leave the column as it is, but any new query that accesses x_j [1] will have a 0 in x_j's column since in reality the record no longer exists. We next consider the case of modifications - if a record x_j is modified to $x_j + \Delta_j$, we introduce a new column in the matrix corresponding to $x_j + \Delta_j$ and all new queries that involve this modified record will have a 1 in this new column and a 0 in the old x_j column. Using this as the query matrix in the sum auditing algorithm will ensure the privacy of both x_j and $x_j + \Delta_j$. We can show formally that 1) adding such new columns for insertions and modifications and 2) modifying query vectors if they access deleted or modified records – ensures the privacy of all past and present values of records in the database even in the presence of multiple updates to the same record. We omit the details due to space constraints but one can easily see

[1] An attacker may try to fool the auditor by asking for $x_i + x_j$ before and after x_j was deleted in order to determine x_i.

the intuition: all values that the attacker can deduce must lie in the span of the query vectors.

A problem with this method for handling modifications however, is that the update to a record can itself still be determined. If an attacker asks for the sum of salaries of Alice and Bob both before and after Alice got a raise, he can figure out the exact amount of her raise since our scheme does not protect the privacy of the Δs. This can easily be fixed by maintaining two matrices. If a record x_j was updated multiples times by amounts Δ_j^1, $\Delta_j^2, \ldots$, we wish to protect the values x_j, $x_j + \Delta_j^1$, $x_j + \Delta_j^1 + \Delta_j^2, \ldots$ as well as the individual updates - $\Delta_j^1, \Delta_j^2, \ldots$. The first matrix we maintain, A_1, contains columns corresponding to $x_1, \Delta_1^1, \Delta_1^2, \ldots, x_j, \Delta_j^1, \Delta_j^2, \ldots, x_n, \Delta_n^1, \Delta_n^2 \ldots$. The second matrix A_2, contains columns corresponding to $x_1, x_1 + \Delta_1^1, \ldots, x_j, x_j + \Delta_j^1, x_j + \Delta_j^1 + \Delta_j^2, \ldots, x_n, x_n + \Delta_n^1, \ldots$. Now any query can be expressed as a row in each matrix in exactly one way and by performing Gaussian elimination on both these matrices, we can ensure that no x_j, Δ_j^i or $x_j + \Sigma\Delta_j$ can be uniquely determined.

An interesting avenue for future work would be to protect the privacy of arbitrary sums of the Δs. A company may as a policy wish to protect the privacy of the total raise an employee has received over a period of time for instance. In this case, in addition to protecting the individual Δs, we also need to protect the sums of consecutive Δs.

3.2 Max Auditing with Updates

Existing algorithm: We now consider the problem of auditing max queries over real-valued data taken from an unbounded range. The algorithm for auditing max queries can be found in [11] and works by maintaining upper bounds for all elements and *extreme elements* for query sets. An extreme element is an element which could possibly be the max element of the query set. For example, if the queries $\max\{x_a, x_b, x_c\} = 9$ and $\max\{x_b, x_d, x_e\} = 8$ have already been posed and answered, we know that x_b is upper-bounded by 8 and thus cannot be an extreme element for the first query set. When a new query is posed, the auditor checks to see if there is any possible answer that is consistent with past answers and that would cause any query set to have only one extreme element (and thus privacy to be breached). If such an answer exists, then the query is denied. The authors show that it suffices to check only a finite number of points in $(-\infty, \infty)$ as possible answers to the new query.

Updates: As in the case of sum auditing, insertions and deletions can easily be handled, by adding new variables corresponding to inserted elements and removing variables from new query sets that access already deleted records. In the case of modifications, if a record x_j is modified to $x_j' = x_j + \Delta$, we consider different possibilities for what the attacker knows beforehand about Δ:

- Δ is known to be a non-zero real number: In this case the value of x_j' is independent of that of x_j and we can treat it as a new insertion in to the database.
- Δ is known to be positive: This translates to the fact that any upper bound on x_j' is now also an upper bound on x_j. With this in mind we perform the same auditing algorithm as before. For instance, suppose an attacker asked for $\max\{x_a, x_b, x_c\} = 9$ and x_a was subsequently increased to x_a'. If the attacker now asks for $\max\{x_a', x_b, x_d\}$ - if no correlation between x_a and x_a' is known to the attacker then the second query can be answered since no matter the answer, both query sets will have more than one extreme element. However, if it is known that $x_a' > x_a$, then in reality, the attacker is asking for $\max\{x_a, x_a', x_b, x_d\}$ and the second query must be denied. This is because if the answer to the second query is less than 9, then x_c will be revealed to be 9. Thus we handle positive updates to variables by introducing a new variable corresponding to the updated value and including both the old and new variables in query sets involving the updated value.
- Δ is known to be negative: This translates to the fact that any upper bound on x_j' is now also an upper bound on x_j. We handle this by including a variable corresponding to x_j' in all past query sets that accessed the old value of x_j. For example, suppose an attacker asked for $\max\{x_a, x_b, x_c\} = 9$ and x_a was later decreased to x_a'. If the attacker now asks for $\max\{x_a', x_b, x_d\}$, this query should be denied since the answer to the second query could be greater than 9 thereby revealing the value of x_d. In the auditing algorithm, this is detected by extending the first query set to $\max\{x_a, x_a', x_b, x_c\} = 9$ thereby placing an upper bound on x_a' as well.

Thus, with simple manipulations to query sets, the max query auditor can be extended to work with updates.

3.3 Max-and-Min Auditing with Updates

Prior to this work, no simulatable algorithm was known for auditing bags of max and min queries in an online fashion. So we begin by first describing the algorithm for the static database scenario and then proceed to deal with the issue of updates. Here we make the assumption that the dataset does not contain any duplicates, i.e. if $i \neq j$ then $x_i \neq x_j$.

Given a set of previously posed max and min queries, $q_1, \ldots, q_{t-1}$ with answers, $a_1, \ldots, a_{t-1}$, we would like to determine if there is any possible answer to a new query q_t that is consistent with previous answers and would cause a data value to be uniquely determined. Checking all possible answers a_t in $(-\infty, +\infty)$ would be impossible but it turns out that it is sufficient to check only a finite number of points. Let $Q_1', \ldots, Q_l'$ be the query sets of previous queries that intersect with Q_t, ordered according to their corresponding answers, $a_1' \leq \ldots \leq a_l'$. Let $a_{lb}' = a_1' - 1$ and $a_{ub}' = a_l' + 1$ be the bounding values. Our algorithm only checks $2l + 1$ points - the bounding values, the above l answers and the mid-points of the intervals determined by them. Algorithm 1 gives the details.

1: **for** $a_t \in \{a_{lb}', a_1', \frac{a_1'+a_2'}{2}, a_2', \ldots, a_{l-1}', \frac{a_{l-1}'+a_l'}{2}, a_l', a_{ub}'\}$ **do**
2: **if** a_t is consistent with $a_1, \ldots a_{t-1}$ AND if $\exists 1 \leq j \leq n$ such that x_j is uniquely determined **then**
3: Output "Deny" and **return**
4: **end if**
5: **end for**
6: Output $f(Q_t)$ and **return**

Algorithm 1: Simulatable Auditor for Max and Min Queries

We next address the questions of checking whether a value is uniquely determined and whether an answer a_t is consistent with answers to previous queries.

Checking if a value is uniquely determined: The idea here is once again to determine the set of extreme elements for every single query set posed by the attacker. The extreme elements of a query set, Q_i, are the x_js whose values could potentially be the answer a_i to the query q_i. For each query set, we would like to eliminate those x_js that could certainly not be extreme elements and in the end if we are left with only one x_j that is an extreme element for a query set, we know that privacy has been breached. If a query set has only one extreme element, then the element is said to be *strictly extreme* for the query set.

Given a set of queries and answers, the upper bound μ_j of an element x_j is defined to be the minimum over the answers to the max queries containing x_j i.e. $\mu_j = \min\{a_k | x_j \in Q_k \text{ and } Q_k \text{ is a max query}\}$. Similarly, the lower bound λ_j of an element x_j is defined as $\lambda_j = \max\{a_k | x_j \in Q_k \text{ and } Q_k \text{ is a min query}\}$. We determine the extreme elements E_k for a query set Q_k as follows:

1: For $k = 1$ to t, $E_k = \emptyset$.
2: If Q_k is a max query set, $x_j \in Q_k$ and $\mu_j = a_k$ then $E_k = E_k \cup \{x_j\}$. Similarly if Q_k is a min query set, if $x_j \in Q_k$ and $\lambda_j = a_k$ then $E_k = E_k \cup \{x_j\}$.
3: For a max query Q_k look at all other max query sets with answer a_k and look at the largest set of extreme elements E common to all the query sets. Set $E_k = E$. This follows because there are no duplicates in the database. For all other x_j that were previously in E_k, we know that $x_j < \mu_j = a_k$, i.e. a_k is a strict upper bound for x_j. The analogous procedure is applied for min query sets.
4: If any extreme element, x_j, of a max query set, Q_M, is strictly extreme for some min query set, Q_m and $a_M \neq a_m$ then $x_j < a_M$ and $E_M = E_M - \{x_j\}$ (and similarly for extreme elements of a min query set).

Algorithm 2: Determining Extreme Elements for Max and Min Query Sets

The last step in this procedure could spark a *trickle effect* and computing the final set of extreme elements for each query set takes $O(t^2 \Sigma_{i=1}^{t} |Q_i|)$ time. Once extreme elements have been computed in this way, we can show the following theorem (proved in the appendix).

THEOREM 1. *Given a set of queries, $q_1, \ldots, q_t$ and corresponding answers $a_1, \ldots, a_t$, the database is secure if and only if every max or min query set has more than one extreme element and there does not exist any max query, q_i and min query, q_j such that $a_i = a_j$.*

Checking for consistency: Checking for consistency can also be done in polynomial time.

THEOREM 2. *Given a set of queries $q_1, \ldots, q_t$, responses $a_1, \ldots, a_t$ are consistent if and only if a) every max and min query set has at least one extreme element b) for every element x_i, $\mu_i > \lambda_i$ if either the upper bound or lower bound for x_i is strict and $\mu_i \geq \lambda_i$ otherwise and c) if $a_i = a_j$ for some min query q_i and some max query q_j, then Q_i and Q_j should have only one extreme element in common.*

This theorem is proved in the appendix. We now make the following claim also proved in the appendix.

THEOREM 3. *For $1 \leq j \leq n$, x_j is uniquely determined for some value of a_t in (a'_s, a'_{s+1}) if and only if x_j is uniquely determined when $a_t = \frac{a'_s + a'_{s+1}}{2}$. Furthermore, the values of a_t in (a'_s, a'_{s+1}) are either all consistent or inconsistent.*

This completes our proof of correctness. Since we run through the for loop in Algorithm 1 $2l + 1$ times, the running time of the algorithm is $O(t^3 \Sigma_{i=1}^{t} |Q_i|)$.

The "no duplicates" assumption: Note that the assumption of no duplicates in the database can easily be achieved by perturbing a dataset if it does contain duplicates. This however leads to a conservative auditor. Consider a scenario when a user asks for $\max\{x_a, x_b, x_c\}$ and later for $\max\{x_a, x_d, x_e\}$. Due to the imposition of the constraint that there be no duplicates in the dataset, the second query will be denied since if both queries were to have the same answer, the value of x_a would be revealed. If duplicates are allowed however, both queries can be answered.

Although the "no duplicates" assumption thus reduces the utility delivered to the user, it does have an advantage. The size of the audit trail can be significantly reduced using blackbox $\mathcal{B}$ (see Section 2.2). $\mathcal{B}$ can be used to maintain separate synopses $B_{\max}$ and $B_{\min}$ for past queries $q_1, \ldots, q_{t-1}$ and answers $a_1, \ldots, a_{t-1}$. When a new query is asked, the new query along with a possible answer can be submitted to $\mathcal{B}$ to obtained updated synopses $B'_{\max}$ and $B'_{\min}$. Since $B'_{\max}$ captures all the information contained in the max queries and their answers and $B'_{\min}$ contains all the information contained in the min queries and their answers, we only need to consider these query sets in determining the extreme elements as in Algorithm 2. We thus no longer need to maintain the entire sequence of queries that have been posed and an audit trail of size $O(n)$ suffices. This was not possible in the max auditing algorithm in [11] where duplicates were allowed.

Updates: If we assume that a modification to a data point is such that the "no duplicates" assumption continues to hold, then the only difference from the max auditing scenario is that we need to take care of lower bounds as well. Additionally, we need to maintain information about whether the lower and upper bounds on the modified elements are strict. If an element x_j is updated to $x'_j = x_j + \Delta$ where Δ is known to be positive, then any lower bound on x_j is a strict lower bound on x'_j. Thus any past min query set involving x_j should be updated to include x'_j as well. Similarly, if Δ is known to be negative then any future min query set involving x'_j should be modified to include x_j as well since any lower bound on x'_j is now a strict lower bound on x_j.

4. PROBABILISTIC COMPROMISE

The notion of probabilistic compromise was introduced for the first time in [11] and is described in Section 2.2. The authors in [11] provide an algorithm for auditing sum queries over data points taken uniformly from the range $[0, 1]$, leaving open the problem of auditing other kinds of queries for data taken from other distributions. We now show how max queries and bags of max and min queries can be audited under this notion of compromise.

4.1 Max Auditing

We consider the problem of building a simulatable auditor for max queries that is $(\lambda, \delta, \alpha, T)$-private for any attacker A. We assume that the dataset is taken uniformly from the set of points in the unit cube $[0,1]^n$ that do not have $x_i = x_j$ for any $i \neq j$.

Recall that under the probabilistic notion of compromise, a query is safe to answer if doing so is not likely to cause a significant change in the attacker's confidence that a data point lies in a particular interval. Also, the decision to deny must be simulatable. The idea here is that the auditor generates a random dataset that is consistent with the answers to queries thus far and checks to see if answering the current query on this dataset would bring about a significant change in the attacker's confidence about the value of any x_i. If the answer is no for a sizable fraction of the random datasets generated, the query is safe to answer.

We first consider the problem of determining whether knowing the answer a_t to query q_t brings about a significant change in the attacker's confidence about a particular data value. For this we make use of the blackbox $\mathcal{B}$ described in Section 2.2. Given queries $q_1, \ldots, q_t$ and corresponding answers $a_1, \ldots, a_t$, $\mathcal{B}$ returns a synopsis, $B_{\max}$ that represents all derivable information as predicates of the form $[\max(S) = M]$ or $[\max(S) < M]$ where every data point can occur in at most one such predicate. For any x_i and any interval I, we would thus like to find $Pr\{x_i \in I | B_{\max}\}$.

Posterior distribution of the x_is: The prior probability that an x_i lies in an interval I of length $1/\alpha$ is given by $Pr\{x_i \in I\} = 1/\alpha$. This is because x_i is initially uniformly distributed in $[0,1]$ [2]. The posterior probability distribution of x_i given $B_{\max}$ is part continuous and part discrete and the computations here require measure-theoretic justifications that we omit. But it can be shown that if $x_i \in S$ and $[\max(S) = M] \in B_{\max}$, then x_i is uniformly distributed in $[0, M)$ with probability $1 - 1/|S|$ and $x_i = M$ with probability $1/|S|$. On the other hand, if $x_i \in S$ and $[\max(S) < M] \in B_{\max}$ then x_i is uniformly distributed in $[0, M)$ with probability 1. As an example, if $[\max\{x_a, x_b, x_c\} = 0.75] \in B_{\max}$, then since any one of the data points is equally likely to be the max value, $x_a = 0.75$ with probability 1/3 and with the remaining 2/3 probability, it is uniformly distributed in [0, 0.75). Thus for any interval in $[0,1]$ it is possible to determine the ratio of the posterior to prior probability that any x_i lies in the interval. This gives rise to Algorithm 5 (presented in the appendix) that returns false if $S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_t) = 0$ and returns true otherwise.

Simulatable auditor: We now construct the simulatable auditor by estimating the probability that answering q_t would breach privacy. The probability is taken over the distribution from which the dataset is drawn and is conditioned on the past queries and their answers. To estimate this probability, we draw a random dataset X' that is consistent with the answers already given, compute an answer a'_t according to X' and evaluate Algorithm 5 on $q_1, \ldots, q_t, a_1, \ldots a'_t$ to see if this answer would cause a considerable change in the attacker's confidence about any one data point. The sampling is then repeated to get a good estimate.

[2]The set of points in the n dimensional cube where $x_i = x_j$ for some $i \neq j$ has measure 0

In order to sample X' uniformly drawn from all datasets consistent with the previous answers, the auditor looks at each predicate $[\max(S) = M]$ or $[\max(S) < M]$ in $B_{\max}$. If the predicate has an equality then it sets some $x_i \in S$ to M and assigns to other x_is in S a value drawn uniformly at random from $[0, M)$. If the predicate has a strict inequality it assigns each $x_i \in S$ a value drawn uniformly at random from $[0, M)$. Every other x_i not represented by a predicate is given a value drawn uniformly at random from $[0,1]$. In case $x_i = x_j$ for some $i \neq j$, the sample is thrown out and another is chosen. But this happens with probability zero. Algorithm 3 gives the details for the simulatable auditor.

1: Input: past queries $q_1, \ldots, q_{t-1}$, answers $a_1, \ldots, a_{t-1}$, new query q_t, parameters $\lambda, \alpha, n, \delta, T$.
2: **for** $O(\frac{T}{\delta} log \frac{T}{\delta})$ times **do**
3: Sample a data set X' consistent with previous answers
4: Evaluate Algorithm 5 on input $q_1, \ldots, q_t, a_1, \ldots, a'_t$ and parameters λ, α, n
5: **end for**
6: **if** the fraction of sampled data sets for which Algorithm 5 returned *false* is more than $\delta/2T$ **then**
7: Return "denied"
8: **else**
9: Return $a_t = max_x(Q_t)$
10: **end if**

Algorithm 3: Simulatable Auditor for Max Queries

THEOREM 4. *Algorithm 3 is a $(\lambda, \delta, \alpha, T)$-private simulatable auditor for max queries.*

PROOF. An attacker wins the game in round t if he poses a query q_t for which $S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_t) = 0$ and the auditor does not deny q_t. The probability that the attacker wins in round t given the answers to previous queries equals

$$p_t = Pr\{S_\lambda(q_1, \ldots, q_t, a_1, \ldots, a_{t-1}, \max_{X'}(Q_t)) = 0 \mid q_1, \ldots, q_{t-1}, a_1, \ldots, a_{t-1}\}$$

where X' is a data set drawn uniformly from the set of all data sets consistent with the previous answers. Thus Algorithm 3 essentially estimates p_t via multiple draws of random data sets X'. When $p_t > \delta/T$, by the Chernoff bound, the fraction of sampled data sets for which algorithm safe returns *false* is larger than $\delta/2T$ with probability at least $1 - \delta/T$. Hence if $p_t > \delta/T$ the attacker wins with probability at most δ/T. When $p_t < \delta/T$, the attacker wins only if the query is answered and even then with probability p_t. In both cases the attacker wins with probability at most δ/T. Thus the probability that the attacker wins in any of the T rounds is less than δ by the union bound. □

4.2 Max-and-Min Auditing

Once again we assume that the dataset is taken from the set of points in the unit cube $[0,1]^n$ that do not have $x_i = x_j$ for any $i = j$. We consider the problem of building a $(\lambda, \delta, \alpha, T)$-private simulatable auditor for bags of max and min queries. Due to space constraints we do not give the algorithm in full but give a sketch of the main ideas involved in constructing the auditor.

As previously, we would like to determine if answering the current query would lead to a significant change in the attacker's confidence about the value of a particular data point. We first consider the problem of determining whether

knowing the answer a_t to query q_t brings about a significant change in the attacker's knowledge. We make use of the blackbox $\mathcal{B}$ described in Section 2.2. Given queries $q_1, \ldots, q_t$ and corresponding answers $a_1, \ldots, a_t$, $\mathcal{B}$ returns a synopsis $B = (B_{\max}, B_{\min})$. $B_{\max}$ represents all derivable information from the max queries as predicates of the form $[\max(S) = M]$ or $[\max(S) < M]$ where every data point can occur in at most one such predicate. And $B_{\min}$ represents all derivable information from the min queries as predicates of the form $[\min(S) = m]$ or $[\min(S) > m]$ where every data point can occur in at most one such predicate. We further modify the predicates as follows: if any max predicate and min predicate have the same value, M, then the corresponding query sets - S_1 and S_2 must have exactly one element, x_j, in common due to the assumption of no duplicates. We remove the two predicates and replace them with the predicates $[\max(x_j) = M]$, $[\max(S_1 - x_j) < M]$ and $[\min(S_2 - x_j) > M]$. At the end of this process no two max and min predicates will have the same answer and each x_i can be determined to lie in a range R_i. Let ℓ_i be 1 divided by the length of this interval.

Posterior distribution of the x_is: The prior probability that an x_i lies in an interval I of length $1/\alpha$ is given by $Pr\{x_i \in I\} = 1/\alpha$. This is because x_i is initially uniformly distributed in $[0, 1]$. We would like to determine the posterior probability $Pr\{x_i \in I|B\}$. As a simple example, let's consider two predicates of the form $[\max\{x_a, x_b, x_c\} = 1]$ and $[\min\{x_a, x_b\} = 0.2]$. We know that x_a and x_b must lie in the range $[0.2, 1]$ and x_c must lie in the range $[0, 1]$. Moreover, one of x_a or x_b must be exactly 0.2, and one of x_a, x_b or x_c must be exactly 1. Now if $x_a = 1$, then x_b must be 0.2 and x_c can lie anywhere in the range $[0, 1)$. This determines a line segment of (one-dimensional) volume 0.8. Enumerating over the other three possible cases, we find that the volume covered by all possible points that satisfy the two constraints is 3.6. Now if we consider the probability that $x_a = 1$, the volume covered by points that satisfy the two constraints and the condition that $x_a = 1$ is 1. Thus $Pr\{x_a = 1\} = 1/3.6 = 5/18$.

Equivalent graph coloring problem: The above procedure can be generalized, but the number of cases to sum over may be exponential in the number of queries. So instead, we approximate the required probability by *sampling* from the joint distribution over the entire dataset X. As an intermediate step, consider the following graph coloring problem. Let $\mathcal{Q}$ be the set of predicates in B with a strict equality. We construct a graph $\mathcal{G}$ with a node corresponding to each predicate in $\mathcal{Q}$. Let $S(Q)$ denote the query set of a predicate $Q \in \mathcal{Q}$ and let $A(Q)$ denote the answer of the predicate. $\mathcal{G}$ has an an edge between Q_1 and Q_2 iff $S(Q_1) \cap S(Q_2) \neq \emptyset$. Each element x_i in the dataset corresponds to a color, and the set of colors available at node Q is just the query set $S(Q)$. A *valid* coloring of $\mathcal{G}$ is a mapping c that assigns to each node Q a color $c(Q) \in S(Q)$, such that if there is an edge between Q_1 and Q_2, then $c(Q_1) \neq c(Q_2)$. We define a probability distribution over valid colorings $\tilde{P}(c) = \frac{1}{Z} \prod_{Q \in \mathcal{Q}} \ell_{c(Q)}$, where Z is a constant[3] chosen to make $\tilde{P}$ sum to 1.

The intuition is that the process of generating a dataset according to the posterior distribution can be split into two parts. First, for each max or min query with equality, we choose an item from its query set which will achieve the bound. This is exactly the information contained in the coloring. After doing this for every query, the values of the remaining items may now be chosen uniformly at random. More formally,

THEOREM 5. *The following procedure generates a sample from $P(X|B)$:*

1. *Sample a coloring according to $\tilde{P}(c)$.*
2. *For each Q, set $x_{c(Q)} = A(Q)$.*
3. *For each remaining unassigned item x_i, sample a value uniformly at random from R_i.*

PROOF. Given a dataset X that satisfies the queries, we can associate to it a unique coloring c as follows : for each predicate Q with strict equality, there must be exactly one item x_i in $S(Q)$ such that $x_i = A(Q)$. Set $c(Q) = x_i$. Conversely, a coloring c specifies the values of a subset of the variables as described in step 2 above, and the remaining ones can take any value in their range. We can therefore write $P(X|B) = P(c|B)P(X|c, B)$. Now, given B and c, the remaining unassigned variables are independent and uniformly distributed over the ranges determined by B. Thus, step 3 above does generate samples from $P(X|B, c)$.

It remains to be shown that step 1 above generates samples from $P(c|B)$. To see this, we examine the geometry more closely. Suppose there are n items in the dataset, and k predicates with equality. Which datasets are compatible with the evidence? Once we choose a coloring, this determines the values of k items, and the remaining items may vary across their ranges. The posterior is thus uniform over a union of rectangles of dimension $n - k$. Each rectangle corresponds to a coloring c, and $P(c|B)$ is therefore proportional to the volume of this rectangle, which is the product of $1/\ell_i$ for those x_i that aren't set by c. Equivalently, the probability is proportional to the product of ℓ_i for those x_i that *are* set by c, which is exactly the definition of $\tilde{P}$. □

We now describe a Markov chain $\mathcal{M}$ over valid colorings, which is a slightly modified version of ones found in the Markov Chain Monte Carlo literature (e.g. [9]). The Markov chain is initialized by looking at the actual state of the database and constructing the corresponding coloring. At each successive step, given a valid c, the Markov chain generates c' as follows:

- Pick Q uniformly from $\mathcal{Q}$.
- For each $Q' \neq Q$, set $c'(Q') = c(Q')$.
- Pick a color x_i from $S(Q)$ with probability proportional to ℓ_i.
- Set $c'(Q) = x_i$ if this results in a valid coloring, and set it to $c(Q)$ otherwise.

We prove the following theorem.

THEOREM 6. *The stationary distribution of $\mathcal{M}$ is $\tilde{P}$.*

[3]Note that our algorithm doesn't explicitly compute Z

PROOF. Let $\mathcal{M}_Q$ denote the transition distribution in the case when we pick Q. Since $\mathcal{M}$ is a convex combination of the $\mathcal{M}_Q$s, it suffices to show that each $\mathcal{M}_Q$ preserves $\tilde{P}$, i.e., sampling from $\tilde{P}$ and then applying one step of $\mathcal{M}_Q$ results in a coloring whose distribution is also $\tilde{P}$.

Let c_1 be a coloring sampled from $\tilde{P}$ and let c_2 be the resulting coloring after transitioning according to $\mathcal{M}_Q$. Let c be any fixed coloring. We would like to show that $Pr\{c_2 = c\} = \tilde{P(c)}$. Let p be the total probability of colorings that agree with c on $\mathcal{Q}\backslash Q$ in $\tilde{P}$. Assume without loss of generality that $c(Q) = 1$, the other valid colors for Q given the rest of c are $2, \ldots, r$, and the remaining colors in $S(Q)$ are $r+1, \ldots, s$. To get c after one step of $\mathcal{M}_Q$, either we must have begun with c and chosen $c(Q)$ or an invalid color, or we must have begun with a coloring that differed from c only on Q, and then chosen the color $c(Q)$. Use the notation $\ell_{a:b}$ to denote $\ell_a + \ell_{a+1} + \ldots + \ell_b$. The total probability of these events is then

$$\begin{aligned} p\frac{\ell_1}{\ell_{1:r}}\frac{\ell_1+\ell_{r+1:s}}{\ell_{1:s}} + p\frac{\ell_{2:r}}{\ell_{1:r}}\frac{\ell_1}{\ell_{1:s}} &= p\frac{\ell_1}{\ell_{1:r}}(1-\frac{\ell_{2:r}}{\ell_{1:s}}) + p\frac{\ell_{2:r}}{\ell_{1:r}}\frac{\ell_1}{\ell_{1:s}} \\ &= p\frac{\ell_1}{\ell_{1:r}} \\ &= \tilde{P}(c) \end{aligned}$$

□

We can also show the following:

THEOREM 7. *Let Δ be the maximum degree of $\mathcal{G}$, and p_{max} and p_{min} be, respectively, the maximum and minimum conditional probabilities for the color of some Q given a coloring of the rest of the graph. If $n > \Delta(1 + 2\frac{p_{max}}{p_{min}})$, then $\mathcal{M}$ has mixing time $O(k\log(k))$.*

The (omitted) proof is an adaption of that given, e.g., in [9], the main difference being that in our case the distribution over colorings is non-uniform. Together, these theorems show that by starting with any valid coloring and running $\mathcal{M}$ for $O(k\log(k))$ steps, we can generate a sample from a distribution that is close to $\tilde{P}$. We may thus answer probability questions up to accuracy ϵ by generating a set of such samples and forming a Monte Carlo estimate. For any x_i and interval I, we can thus get arbitrarily close estimates of $Pr\{x_i \in I|B\}$. Even if the condition on n in Theorem 7 is not satisfied, there exist other inference techniques besides Markov Chain Monte Carlo sampling, for example a reduction to probabilistic graphical models which we omit here for lack of space.

Simulatable auditor: As in the case of the max auditor, before answering q_t, the auditor generates random datasets consistent with answers to past queries by generating colorings according to distribution $\tilde{P}$ as described above. For each such dataset, X', he checks whether answering q_t in X' is likely to cause a privacy breach. This is done by generating random datasets consistent with all past answers as well as the answer to q_t in X' and estimating the posterior probability that each x_i lies in each interval. A significant difference in the posterior and prior probabilities that any x_i lies in any interval implies that q_t is not safe to answer in the dataset X'. If for a large fraction of sampled datasets, q_t is deemed unsafe, the query is rejected, otherwise it is answered.

5. UTILITY

In this section we attempt to get a handle on the utility provided by the existing online auditing algorithm for sum queries over real-valued data taken from an unbounded range. While there are several possible dimensions along which utility can be measured (and we discuss these later), the dimension that we consider here is the number of queries posed by a user that are actually answered. In particular we would like to measure the expected number of queries that would be answered in a sequence of random queries [4] posed by the user. The larger this number, the greater the utility derived by the user. We show here a surprisingly positive result for large databases - the number of queries that can be answered is at least a constant fraction of the size of the database.

The sum auditing algorithm can be found in [4, 11] and was briefly described in Section 3.1. Essentially a diagonalized form of the query matrix is efficiently maintained to detect when individual values can be determined via Gaussian elimination. We consider a series of random queries posed to the auditor and compute the expected time to first denial, T_{denial}.

THEOREM 8. $E[T_{denial}] \geq n/5$

PROOF.

$$\begin{aligned} E[T_{\text{denial}}] &= \sum_{m=1}^{\infty} mPr\{T_{\text{denial}} = m\} \\ &= \sum_{m=1}^{\infty} Pr\{T_{\text{denial}} \geq m\} \\ &= 1 + \sum_{m=2}^{\infty} Pr\{T_{\text{denial}} \geq m\} \\ &= 1 + \sum_{m=2}^{\infty} Pr\{\text{No denial in } q_1 \ldots q_{m-1}\} \end{aligned}$$

We would thus like to get a lower bound on the probability that there is no denial in the first m of a sequence of random queries. Let the query vectors of the posed queries be $v_1, \ldots, v_m$ and let A be the matrix of query vectors. Note that $v_1, \ldots, v_m$ are random 0-1 vectors. Compromise occurs if we can find any real-valued λ_is such that $\Sigma_i \lambda_i v_i$ gives us a unit vector (a vector with only one 1 and $n-1$ 0s). Let A^T be the transpose of the matrix A. A^T is thus an $n \times m$ matrix of 0s and 1s. Now if we could find some m-dimensional vector w such that $A^T w$ gives us a unit vector then the elements of w would correspond to the λ_is that give rise to the privacy breach. w must thus be perpendicular to all the rows in A^T except for the one row which gives rise to the 1 in the unit vector. We can now state that there is no denial amongst the first m queries if and only if there is no m-dimensional vector that is perpendicular to all but one of the rows in A^T. We thus need to lower bound the probability that there is no such m-dimensional vector.

Note that if we could divide the rows of A^T in to two disjoint sets, each of which forms a basis for $\mathbb{R}^m$, then there could be no such vector. This is because there can be no m-dimensional vector that is perpendicular to all the vectors

[4] A random query is a query drawn uniformly at random from the set of all sum queries that could be formulated over the data.

that form the basis of $\mathbb{R}^m$. So if we take any row in the matrix and claim that there is some vector w perpendicular to all other rows but this, we know that this is not possible since a subset of the remaining rows must form a basis for $\mathbb{R}^m$. We thus need to lower bound the probability that A^T contains such a double basis. For this we consider the rows of A^T one after the other and calculate the probability that both the first $n/2$ and the last $n/2$ rows span $\mathbb{R}^m$ separately. The following lemma is useful in finding this probability:

LEMMA 1. *A hyperplane H in $\mathbb{R}^m$ can intersect B^m in at most $2^{d(H)}$ points where $d(H)$ is the dimension of the hyperplane.*

Here B is the set $\{0,1\}$ and B^m is the boolean hypercube in m dimensions. Note that the rows of A^T correspond to vertices of B^m. As we consider the rows of A^T in order, we would like to find the probability that a new row raises the rank of the matrix. If i rows have been considered thus far, they can span a hyperplane of dimension at most i. From our lemma, this can intersect B^m in at most 2^i points. Thus there are 2^i points in B^m that are linearly dependent on the i rows and $2^m - 2^i$ points that are linearly independent. So the probability that the $i+1$st row raises the rank of the matrix is simply $1 - 2^i/2^m > 1/2$. We can thus consider a process where we toss a coin $n/2$ times. At each time step the probability of *heads* is $1/2$ and we would like to compute the probability that m of the tosses turn up heads. This would be a lower bound for the probability that the first $n/2$ rows of A^T span $\mathbb{R}^m$. Using Chernoff bounds, we can show that if $m \leq n/5$ then with high probability $(1 - o(1))$ this happens. Similarly, the remaining $n/2$ rows also span $\mathbb{R}^m$ with high probability. Thus

$$\begin{aligned} E[T_{\text{denial}}] &\geq 1 + \sum_{m=2}^{n/5} Pr\{\text{no denial in } q_1 \dots q_{m-1}\} \\ &= 1 + \sum_{m=2}^{n/5} 1 - o(1) \\ &= n/5 \end{aligned}$$

□

Thus the first denial occurs after $\theta(n)$ queries. We now show that it also occurs within $\theta(n)$ queries.

THEOREM 9. $E[T_{denial}] \leq n + \log n + \theta(1)$

PROOF.

$$E[T_{\text{denial}}] = \sum_{m=1}^{\infty} Pr\{X \geq m\}$$

For m ranging from 1 to $m^* = n + \log(n \ln 2)$, we upper-bound $Pr\{x \geq m\}$ by 1. This gives us

$$\begin{aligned} E[T_{\text{denial}}] &\leq n + \log(n \ln 2) - 1 + \sum_{m=m^*}^{\infty} Pr\{X \geq m\} \\ &= n + \log(n \ln 2) - 1 + \\ &\quad \sum_{m=m^*}^{\infty} Pr\{\text{no denial in } q_1 \dots q_m\} \end{aligned}$$

Once again we consider the $n \times m$ matrix A^T and the probability that there is no vector perpendicular to all but one of its rows. Note that if these n rows were linearly independent then there would always be such a vector. So we would like to upper-bound the probability that the n random 0-1 rows of the matrix are not linearly independent. If we call this probability $P_{n,m}$, then we can show that $P_{n,m} \leq n/2^{m-n+1}$. Thus

$$\begin{aligned} E[T_{\text{denial}}] &\leq n + \log(n \ln 2) - 1 + \sum_{m=m^*}^{\infty} \frac{n}{2^{m-n+1}} \\ &= n + \log(n \ln 2) - 1 + \frac{n}{2^{m^*-n}} \\ &= n + \log(n \ln 2) - 1 + \frac{1}{\ln 2} \\ &= n + \log n + \theta(1) \end{aligned}$$

□

Thus we can expect the first denial to occur in $\theta(n)$ queries. Note that our lower bound on the expected time to first denial is a high probability result - with high probability $(1 - o(1))$ there will be no denial in the first $n/5$ queries posed to the database. This is a positive result for very large datasets - since n is large, many queries will be answered before even the first denial occurs.

The next question to ask is what happens after the first denial. Note that once the rank of the query matrix reaches $n-1$, denials will happen with probability at least $1/2$. This because at least half the points in B^m will be linearly independent of the rows of the query matrix and if any of these are answered, the rank of the query matrix will reach n, enabling the attacker to solve the system of equations for all the points in the dataset. The question to answer is thus how quickly does the rank of the query matrix become $n-1$ after the first denial. It seems that this would happen rapidly since we can show, using similar arguments, that the expected time till $n-1$ linearly independent queries are posed is $\theta(n)$. Not all these queries would be answered however, and an exact analysis remains to be done.

In case n is small, the above would indicate a need to relax some of the strict conditions required off an auditor - in particular the requirement that an auditor answer truthfully, if at all. We suggest combining denials with adding noise (as in the output perturbation approaches in [6, 8]) as a solution. The idea is to set a bound on the amount of noise that can be added to any answer and then to add noise (within this bound) to the answer to a query depending on how much private information the query reveals. If the answer to a query would reveal the value of any individual no matter how much noise is added, the query should be denied. Such an approach would probably enable us to answer more queries than in the exact answer approach and at the same time would probably give greater precision in answers that are provided than a pure output perturbation based approach. Another possible approach, if queries are taken from some distribution other than uniform, is to deny certain queries in the present (even if they do not cause a privacy breach) in the hope that more queries can be answered in the future.

Note that in addition to the number of queries that are answered, there are several other dimensions along which utility can be measured. There is the *price of simulatability* - how many queries were denied when they could have been safely answered because we did not look at the true answers to the queries when choosing to deny them. Additionally,

there might be some important, fairly generic queries that the world would always like to have answered. For example the total number of cancer patients in a particular hospital. An auditing scheme that could deny such a query would be considered to provide low utility. A complete analysis of all these different aspects of utility is an avenue for future work.

6. CONCLUSIONS

We considered a wide variety of problems with the goal of increasing the robustness of the notion of query auditing. We considered the realistic scenario of non-static databases and showed how existing algorithms could be modified to work in this case. In the process we introduced a new simulatable algorithm for bags of max and min queries when the database does not contain duplicates. An interesting avenue for future work here is to devise an algorithm that works in the presence of duplicates as well. The next problem we considered was building new simulatable auditors for different kinds of queries under the new notion of probabilistic compromise and here we introduced auditors for max queries and bags of max and min queries. Considering more complex kinds of queries would be the next step. Further, we gave an initial analysis of the utility of simulatable sum auditing showing a positive result for large databases. There is scope for much future work here - the question of utility for other types of queries and other dimensions of utility mentioned in Section 5 remains unexplored.

7. REFERENCES

[1] N. Adam and J. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[2] L. Beck. A security machanism for statistical database. *ACM Trans. Database Syst.*, 5(3):316–3338, 1980.

[3] F. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, 33(3):451–464, 1986.

[4] F. Chin and G. Ozsoyoglu. Auditing for secure statistical databases. In *Proceedings of the ACM '81 conference*, pages 53–59. ACM Press, 1981.

[5] T. Dalenius. Towards a methodology for statistical disclosure control. *Sartryck ur Statistisk tidskrift*, 15:429–444, 1977.

[6] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

[7] D. Dobkin, A. Jones, and R. Lipton. Secure databases: Protection against user influence. *ACM Trans. Database Syst.*, 4(1):97–106, 1979.

[8] C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *CRYPTO*, 2004.

[9] A. Frieze and E. Vigoda. Survey of markov chains for randomly sampling colorings. *To appear in Festschrift.*

[10] J. Kam and J. Ullman. A model of statistical databases and their security. *ACM Trans. Database Syst.*, 2(1):1–10, 1977.

[11] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable Auditing. In *Proc. of ACM PODS*, 2005.

[12] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Auditing boolean attributes. *Journal of Computer and System Sciences*, 6:244–253, 2003.

[13] Y. Li, L. Wang, X. Wang, and S. Jajodia. Auditing interval-based inference. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, pages 553–567, 2002.

[14] S. Reiss. Security in databases: A combinatorial study. *J. ACM*, 26(1):45–57, 1979.

APPENDIX

A. MAX-AND-MIN AUDITING UNDER CLASSICAL COMPROMISE

A.1 Proof of Theorem 1

PROOF. If any query set has only one extreme element, x_i, then we know that the value of x_i has to be the answer to the corresponding query. In addition, if there exist a max query and a min query with the same answer, a, then their corresponding query sets can have only one element in common whose value has to be a.

Now suppose that every query set has at least two extreme elements. We first show how to assign values to each of the x_is so that the answers to the queries on these values will be consistent with the observed answers. Initially mark all query sets as "unvisited" and all data elements as "unchosen". Then carry out the following procedure.

1: **while** there exist unvisited query sets **do**
2: Pick an arbitrary unvisited query set - Q_k.
3: Find all other query sets of the same type (max or min) as Q_k with the same answer. Let the query sets be Q and the answer be a.
4: Let E be the set of extreme elements common to these query sets. Set some unchosen $x_i \in E$ to a.
5: Mark every query set in Q as "visited" and x_i as "chosen".
6: **if** x_i is also extreme for unvisited query set, Q_l of opposite type **then**
7: Go to step 3 with $Q_k = Q_l$.
8: **end if**
9: **end while**
10: Set every unchosen x_i to some arbitrary value between its upper and lower bound.

Algorithm 4: AssignValues

Clearly every unvisited query set has only at most one of its extreme elements set to chosen and thus we can always set the value of another extreme element to be the answer to the query set when visiting it.

Suppose that after this process, an x_i is uniquely determined to be either its upper or lower bound. Without loss of generality, let's assume that x_i is set to its upper bound. Look at all the max sets, $\mathcal{Q}_M$, in which x_i is an extreme element. For each of these sets, there exists at least one other extreme element, x_j. Repeat the above procedure for assigning values to elements starting out by visiting the sets in $\mathcal{Q}_M$ and choosing x_j. Since no other max set outside of $\mathcal{Q}_M$ can contain x_i as an extreme element, x_i will either be set to its lower bound (which is not equal to its upper bound since no pair of max and min queries have the same answer) or to some value between its upper and lower bound by the end of this procedure. Yet the new assignment of values is still consistent with the answers to the queries. Thus x_i is not in fact uniquely determined. □

A.2 Proof of Theorem 2

PROOF. If any of the conditions in the theorem are violated it is easy to see that the answers to the queries are not consistent. It thus remains to show that any set of responses that satisfy the above conditions are consistent. We show this by providing an assignment of values to elements such that the responses to queries on these elements would be the same as the observed responses. We first consider all the query sets, Q_i that have only one extreme element, x_j. We set $x_j = a_i$. Note that if any other max or min query, Q_k had x_j as an extreme element, $a_k = a_i$ and in this way we satisfy all queries with x_i as an extreme element. Now all the remaining query sets have at least two extreme elements that have not had values assigned to them as yet. Applying Algorithm 4, we can get a feasible assignment of values to the elements. □

A.3 Proof of Theorem 3

PROOF. Assume that q_t is a max query (the claim for min queries can be proved similarly). Observe that revealing a_t may affect elements in Q_t and elements in other max query sets intersecting Q_t. This is because revealing a_t can possibly lower the upper bounds of elements in Q_t thereby making some element in Q_t or the max query sets intersecting it the only extreme element of that set. Revealing a_t would only affect other query sets (min query sets or other non-intersecting max query sets) if it did in fact make one of elements of an intersecting max query set a strictly extreme element or if the answer to the query set was a_t. In the former case compromise occurs any way, so it suffices to consider only the impact that a_t has on Q_t and intersecting max query sets - $Q_1*, \ldots, Q_m*$. We consider the following cases:

- $Q_t = \{x_j\}$: Clearly x_j is breached irrespective of a_t.
- x_j is the only extreme element of Q_t and $|Q_t| > 1$: Suppose that x_j is uniquely determined for some value of a_t in (a'_s, a'_{s+1}). This means that each element in $Q_t - x_j$ had an upper bound $< a_t$ and hence less than a'_s (since an upper bound can only be one of the answers so far). Since this holds even for $a_t = \frac{a'_s + a'_{s+1}}{2}$, x_j is still uniquely determined. A similar argument applies for the converse.
- x_j is the only extreme element of Q_k* for some k: Suppose that x_j is uniquely determined for some value of a_t in (a'_s, a'_{s+1}). This means that $a_t < a_k*$ (and hence $a'_{s+1} \leq a_k$) and revealing a_t reduced the upper bound of some element in $Q_k * - x_j$. This would be the case even when $a_t = \frac{a'_s + a'_{s+1}}{2}$. A similar argument applies for the converse.

To complete the proof, we show that values of a_t in (a'_s, a'_{s+1}) are either all consistent or inconsistent. Suppose that some value of a_t in (a'_s, a'_{s+1}) results in inconsistency. This means that either it causes some Q_α to have no extreme elements or it causes the inferable upper bound on some x_i to become smaller than its lower bound.

Suppose that the response a_t causes some Q_α to have no extreme elements. This could either be the result of an immediate effect on Q_t or one of the intersecting max query sets or it could be the result of a trickle effect. In the former case, suppose a_t causes either Q_t or an intersecting max query set to have no extreme element. Then every element in Q_α must have had an upper bound $< a_t$ and hence less than a'_s. This is true for any a_t in (a'_s, a'_{s+1}). If the inconsistency was brought about by a trickle effect that was spawned by either Q_t or an intersecting max query set having a single

extreme element, x_j, then any other a_t in (a'_s, a'_{s+1}) would cause x_j to become strictly extreme thereby spawning the same trickle effect.

Similarly, we can show that if a_t causes the inferable upper bound on some x_i to become smaller than its lower bound, either directly or through a trickle effect, we can achieve the same effect by using any a_t in (a'_s, a'_{s+1}).

□

B. ALGORITHM 5

```
Input: Queries and Answers q_j and a_j for j = 1, ..., t and parameters λ, α, n.
Let safe = true
for each x_i and each interval I_j in 𝓘 do
  if x_i ∈ S and [max(S) = M] ∈ B_max then
    Let y = (1−1/|S|)/(Mα)
    if j < ⌈Mα⌉ then
      if α × y < (1 − λ) OR > 1/(1 − λ) then
        Let safe = false
      end if
    else if j = ⌈Mα⌉ then
      if α((Mα − ⌈Mα⌉ + 1)y + 1/|S|) < (1 − λ) OR > 1/(1 − λ) then
        Let safe = false
      end if
    else
      Let safe = false
    end if
  else if x_i ∈ S and [max(S) < M] ∈ B_max then
    Let y = 1/(Mα)
    if j < ⌈Mα⌉ then
      if α × y < (1 − λ) OR > 1/(1 − λ) then
        Let safe = false
      end if
    else if j = ⌈Mα⌉ then
      if yα(Mα − ⌈Mα⌉ + 1) < (1 − λ) OR > 1/(1 − λ) then
        Let safe = false
      end if
    else
      Let safe = false
    end if
  else
    Let safe = true
  end if
end for
Return safe.
```

Algorithm 5: Safe

C. PROOF OF LEMMA 1

PROOF. Let B be the set $\{0, 1\}$. B^m is then the boolean hypercube in m dimensions. A *h*yperplane H is a set of the form $\{v + \sum_{i=1}^{l} \lambda_i v_i\}$ where $v, v_1, \ldots, v_l$ are points in $\mathbb{R}^m$ and λ_i range over the reals. In our scenario, $v = (0, \ldots, 0)$ since our hyperplane always passes through the origin. The *span* of a hyperplane is the set of points which can be written as $x - x'$ where $x, x' \in H$. Note that the span is a linear subspace of $\mathbb{R}^m$. We define the dimension $d(H)$ of H to be the dimension of its span. Our claim is that a hyperplane H in $\mathbb{R}^m$ can intersect B^m in at most $2^{d(H)}$) points.

The proof is by induction on m. Let $l = d(H)$. Note that $l \leq m$. The case $m = 1$ is trivial.

Suppose $m > 1$. If $l = m$, the statement is true since $|B^m| = 2^m = 2^l$. The remaining case is when $m > l$. Assume for a contradiction that $x_1, \ldots, x_k$ are distinct points in $B^m \cap H$ with $k > 2^l$.

Let P_j denote the projection map from $\mathbb{R}^m$ to $\mathbb{R}^{m-1}$ that simply omits the j^{th} dimension. Consider the points $P_j x_1, \ldots P_j x_k$. If, for some j, these points are all distinct, we then

have k distinct points in $P_j B^m \cap P_j H = B^{m-1} \cap P_j H$. Since projecting a hyperplane cannot increase its dimension, we have a contradiction to the induction hypothesis.

The other possibility is that, for each j, there are two different points x_{i_1} and x_{i_2} with $P_j x_{i_1} = P_j x_{i_2}$. Since $x_{i_1} \neq x_{i_2}$, x_{i_1} and x_{i_2} must differ on the j^{th} coordinate and be the same on all others. It follows that the vector e_j, which has a 1 in the j^{th} position and 0 elsewhere, is in the span of H. But since this happens for each of the m dimensions j, the span of H has dimension $m > l$, which is a contradiction. $\square$