Journal of Computer and System Sciences 79 (2013) 1322–1340

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Denials leak information: Simulatable auditing

Krishnaram Kenthapadi [a,*], Nina Mishra [a], Kobbi Nissim [b,1]

[a] Search Labs, Microsoft Research, Mountain View, CA 94043, United States
[b] Department of Computer Science, Ben-Gurion University, P.O. Box 653, Be'er Sheva 84105, Israel

ARTICLE INFO

Article history:
Received 13 October 2008
Received in revised form 15 September 2011
Accepted 18 June 2013
Available online 28 June 2013

Keywords:
Auditing
Privacy-preserving
Simulation paradigm

ABSTRACT

Imagine a data set consisting of private information about individuals. The *online query auditing problem* is: given a sequence of queries that have already been posed about the data, their corresponding answers and given a new query, deny the answer if privacy can be breached or give the true answer otherwise. We investigate the fundamental problem that query denials leak information. This problem was largely overlooked in previous work on auditing. Because of this oversight, some of the previously suggested auditors can be used by an attacker to compromise the privacy of a large fraction of the individuals in the data. To overcome this problem, we introduce a new model called *simulatable auditing* where query denials provably do not leak information. We present a simulatable auditing algorithm for `max` queries under the classical definition of privacy where a breach occurs if a sensitive value is fully compromised. Because of the known limitations of the classical definition of compromise, we describe a probabilistic notion of (partial) compromise, closely related to the notion of semantic security. We demonstrate that `sum` queries can be audited in a simulatable fashion under probabilistic compromise, making some distributional assumptions.

"*Denial ain't just a river in Egypt.*"

[Mark Twain]

1. Introduction

Let $X = \{x_1, \ldots, x_n\}$ be a set of n private values from n individuals, where each x_i is some real value. We consider the *online query auditing problem*: Suppose that the queries $q_1, \ldots, q_{t-1}$ have already been posed about X and the answers $a_1, \ldots, a_{t-1}$ have already been given, where each a_j is either the true answer to the query or "denied". Given a new query q_t, deny the answer if privacy may be breached,[2] and provide the true answer otherwise. The *classical* definition of breach is that there exists an index i such that x_i is uniquely determined. In other words, in all data sets consistent with the queries and answers, there is only one possible value of x_i. The kinds of queries considered in this paper are `sum` and `max` queries. Given a collection of indices $S \subseteq [n]$, $\texttt{sum}(S) = \sum_{i \in S} x_i$ and $\texttt{max}(S) = \max_{i \in S} x_i$.

One of the first auditing results dates back almost 30 years ago to the work of Dobkin, Jones, and Lipton [9]. That work restricts the class of acceptable queries by their size and overlap, and then demonstrates that a data set cannot

* Corresponding author.
E-mail addresses: krisken@microsoft.com (K. Kenthapadi), ninam@microsoft.com (N. Mishra), kobbi@cs.bgu.ac.il (K. Nissim).

[1] Research partially supported by the Israel Science Foundation (grant No. 860/06).

[2] We use the terms *breach* and *compromise* interchangeably to denote disallowed leakage of information.

0022-0000/$ – see front matter
http://dx.doi.org/10.1016/j.jcss.2013.06.004

be compromised provided that the number of queries is appropriately upper bounded. Following that work, many others considered the auditing problem including [25,5,6,4,18] (see also [1] for a survey).

In the work of Chin [4], the online max auditing problem is considered. Given an online sequence of max queries, that paper gives a method of determining whether a value has been uniquely determined using only the queries that were exactly answered. Another example of an online auditing algorithm is due to Kleinberg, Papadimitriou, and Raghavan [18]. In the case that the underlying data set is Boolean valued, a method is given for determining whether a value has been uniquely determined – again using only the queries that were exactly answered.

These online auditing algorithms ignore the queries that were denied and this turns out to be quite problematic since denials can leak information. A simple example illustrates the phenomena. Suppose that the underlying data set is real-valued and that a query is denied only if some value is fully compromised. Suppose that the attacker poses the first query $\mathtt{sum}(x_1, x_2, x_3)$ and the auditor answers 15. Suppose also that the attacker then poses the second query $\mathtt{max}(x_1, x_2, x_3)$ and the auditor denies the answer. The denial tells the attacker that if the true answer to the second query were given then some value could be uniquely determined. Note that $\mathtt{max}(x_1, x_2, x_3) \not< 5$ since then the sum could not be 15. Further, if $\mathtt{max}(x_1, x_2, x_3) > 5$ then the query would not have been denied since no value could be uniquely determined. Consequently, $\mathtt{max}(x_1, x_2, x_3) = 5$ and the attacker learns that $x_1 = x_2 = x_3 = 5$ – a privacy breach of all three entries. The issue here is that query denials reduce the space of possible consistent solutions, and this reduction is not explicitly accounted for in existing online auditing algorithms.

This oversight of previous work on auditing [4,18] is critical: *privacy-preserving auditing algorithms that ignore denials can be used to launch massive privacy attacks*.

1.1. Contributions

Our first contribution is to illustrate how such major privacy violations can ensue from ignoring query denials. In the case of max queries, we illustrate how 1/8 of the data set can be compromised in expectation by allowing the denials of [4] to guide us towards a breach. In the case where the underlying data set is Boolean valued and each query requests the sum of a subset of private values, we demonstrate how the "conservative" approximate online auditing algorithm of [18] (that denies more often than it should) can be used to compromise about 1/2 of the data set in expectation – again using the denials to guide us towards a breach.

How can we overcome the problem of denials leaking information? The simulation paradigm from cryptography offers an elegant solution that we apply to auditing. We say that an auditor is *simulatable* if an attacker, knowing the query-answer history, could make the same decision as to whether or not a newly posed query will be answered. Since the auditor only uses information already known to the attacker when deciding whether to deny, the attacker can mimic or simulate that decision. Hence, the decision to deny a query provably does not leak any information.

We next give a general method for designing simulatable auditors. The idea is that if an auditor has received queries $q_1, \ldots, q_t$ and given answers $a_1, \ldots, a_{t-1}$, it simply considers many possible answers to the query q_t (obliviously of the actual data) and determines how often privacy would be compromised. If privacy is breached one or many times in these answers, then the query is denied, otherwise the query is answered.

We then give an algorithm for simulatable auditing of max queries under classical compromise, where a compromise occurs if a value is uniquely determined. The algorithm runs in time logarithmic in the number of previous queries and linear in the sum of the sizes of the previous queries. Simulatable auditing of sum queries follows from previous work.

Next we revisit the definition of compromise. The classical definition of compromise has been extensively studied in prior work. This definition is conceptually simple, has an appealing combinatorial structure, and serves as a starting point for evaluating solutions for privacy. However, as has been noted by many others, e.g., [3,16,7,19], this definition is inadequate in many real contexts. For example, if an attacker can deduce that a private data element x_i falls in a tiny interval, then the classical definition of privacy is not violated unless x_i can be uniquely determined. While some have proposed a privacy definition where each x_i can only be deduced to lie in a sufficiently large interval [19], note that the distribution of the values in the interval matters. For example, ensuring that age lies in an interval of length 50 when the user can deduce that age is between $[-50, 0]$ does not preserve privacy.

In order to extend the discussion on auditors to more realistic partial compromise notions of privacy, we describe an auditing privacy definition that is similar to the notion of semantic security. Our privacy definition assumes that there exists an underlying probability distribution from which the data is drawn. This is a reasonable assumption since many attributes such as age and salary can have a known probability distribution. The essence of the definition is that for each data element x_i and every interval J with not too small a priori probability mass, the auditor ensures that the prior probability that x_i falls in the interval J is about the same as the posterior probability that x_i falls in J given the queries and answers. This definition overcomes some of the aforementioned problems with classical compromise.

With this notion of privacy, we describe a simulatable auditor for sum queries. The new auditing algorithm computes posterior probabilities by utilizing existing randomized algorithms for sampling from a logconcave distribution, e.g., [20]. To guarantee simulatability, we make sure that the auditing algorithm does not access the data set while deciding whether to allow the newly posed query q_t (in particular, it does not compute the true answer to q_t). Instead, the auditor draws many data sets according to the underlying distribution, conditioned on the previous queries and answers. For each of

the randomly generated data sets, the auditor computes the answer a'_t to the current query and checks whether revealing this answer would breach privacy. If for most answers the data set is not compromised then the query is answered, and otherwise the query is denied.

1.2. Overview of the paper

The rest of this paper is organized as follows. In Section 2 we discuss related work on auditing. In Section 3 we illustrate how denials leak information and show that auditors proposed in previous work [4,18] can be used to launch privacy attacks. We then introduce simulatable auditing in Section 4 and prove that `max` queries can be audited under this definition of auditing and the classical definition of privacy in Section 5. Then in Section 6 we describe a probabilistic definition of privacy. Finally in Section 7 we prove that `sum` queries can be audited under this definition of privacy in a simulatable fashion.

2. Related work

We partition related work into online and offline auditing. In the *offline* auditing problem, one is given a sequence of queries and exact answers and the goal is to determine if a privacy breach has occurred ex post facto. As the initial motivation for work on auditing involves the online auditing problem, we begin with known online auditing results.

2.1. Online auditing

The earliest work is due to Dobkin, Jones, and Lipton [9] and Reiss [25] for the online `sum` auditing problem, where the answer to a query q is $\sum_{i\in q} x_i$. With queries of size at least k elements, each pair overlapping in at most r elements, they showed that any data set can be compromised in $(2k-(\ell+1))/r$ queries by an attacker that knows ℓ values a priori. For fixed k, r and ℓ, if the auditor denies answers to query $(2k-(\ell+1))/r$ and on, then the data set is definitely not compromised. Here the monitor logs all the queries and disallows q_i if $|q_i| < k$, or for some query $t < i$, $|q_i \cap q_t| > r$, or if $i \geqslant (2k-(\ell+1))/r$.[3] These results completely ignore the answers to the queries. On the one hand, we will see later that this is desirable in that the auditor is simulatable – the decision itself cannot leak any information about the data set. On the other hand, we will see that because answers are ignored, sometimes only short query sequences are permitted (that could be longer if previous answers were used).

The online `max` auditing problem was first considered in [4]. Both the online and offline Boolean `sum` auditing were considered in [18]. We describe these online results in more detail in Section 3 and the offline Boolean `sum` auditing work in Section 2.2.

Following the publication of [17], the paper [22] solves the online `max` and `min` simulatable auditing problem under both classical and probabilistic compromise (where both `max` and `min` queries are allowed). An initial study of the utility of auditing is also undertaken.

2.2. Offline auditing

In the *offline auditing* problem, the auditor is given an offline set of queries $q_1,\ldots,q_t$ and true answers $a_1,\ldots,a_t$ and must determine if a breach of privacy has occurred. In most related work, a privacy breach is defined to occur whenever some element in the data set can be uniquely determined. If only `sum` or only `max` queries are posed, then polynomial-time auditing algorithms are known to exist [5]. However, when `sum` and `max` queries are intermingled, then determining whether a specific value can be uniquely determined is known to be NP-hard [4].

Kam and Ullman [16] consider auditing subcube queries which take the form of a sequence of 0s, 1s, and *s where the *s represent "don't cares". For example, the query 10**1* matches all entries with a 1 in the first position, 0 in the second, 1 in the fifth and anything else in the remaining positions. Assuming `sum` queries over the subcubes, they demonstrate when compromise can occur depending on the number of *s in the queries and also depending on the range of input data values.

Kleinberg, Papadimitriou, and Raghavan [18] investigate the offline `sum` auditing problem of Boolean data. They begin by proving that the offline `sum` auditing problem is coNP-hard. Then they give an efficient offline `sum` auditing algorithm in the case that the queries are "one-dimensional", i.e., for some ordering of the elements say $x_1,\ldots,x_n$, each query involves a consecutive sequence of values $x_i, x_{i+1}, x_{i+2},\ldots,x_j$. An offline `max` auditing algorithm is also given. Note that an offline algorithm cannot be used to solve the online problem as illustrated in [17].

In the *offline maximum auditing* problem, the auditor is given a set of queries $q_1,\ldots,q_t$, and must identify a maximum-sized subset of queries such that all can be answered simultaneously without breaching privacy. Chin [4] proved that the offline maximum `sum` query auditing problem is NP-hard, as is the offline maximum `max` query auditing problem.

Miklau and Suciu [21] consider a definition of privacy where the privacy of property A is preserved by the disclosure of property B if $\Pr(A|B) = \Pr(A)$ for all probability distributions over the underlying data set. This definition of privacy is quite

[3] Note that this is a fairly negative result. For example, if $k = n/c$ for some constant c and $r = 1$, then the auditor would have to shut off access to the data after only a constant number of queries, since there are only about c queries where no two overlap in more than one element.

strong and actually no non-trivial pair (A, B) satisfies this privacy definition, unless some assumptions are made about the underlying probability distribution. More recently, Evfimievski et al. [13] consider an asymmetric notion of privacy where privacy is not violated if the disclosure of B causes a loss of confidence in A, but is violated if the disclosure of B helps an attacker gain confidence in A. The relaxation in privacy definition permits more queries than prior offline work.

2.3. Subsequent work

New auditing papers have built upon the work initially published in [17]. For example, Nabar et al. [22] give simulatable auditing algorithms for max and min queries. Also, that paper undertakes a first study of utility, demonstrating that for large databases, many queries will be answered, so that the algorithm does not repeatedly deny. In addition, simulatable algorithms are considered for publishing suppressed contingency tables in [23].

The privacy definition used in this paper has some limitations. Specifically, the setting assumes that the private dataset has entries drawn independently and identically from some underlying distribution $\mathcal{D}$ known to both the attacker and the auditor. While this definition greatly improves on classical definitions where compromise occurs only if an attacker exactly learns a sensitive value, it still carries weaknesses: (1) that private values are independent of each other and (2) that the attacker knows $\mathcal{D}$. These assumptions are susceptible to auxiliary knowledge attacks [10], e.g., the HIV status of a husband and wife are often not independent. In reality, the attacker's prior distribution can be different from the true distribution. However, if the attacker's distribution can deviate arbitrarily from the true data distribution, any release by the auditor will result in a partial disclosure of some private value, since the auditor is required to release exact answers if at all. For example, consider a database in which height is a private attribute, and consider an attacker with the prior belief that all men are less than a foot tall. If by querying the data, the attacker suddenly learns that this is not true and there is substantial change in the attacker's posterior distribution, the privacy breach would be huge. In reality, the attacker's prior beliefs are so far off the mark, that there is no aggregate query about the heights that the auditor can truthfully answer without compromising privacy, not even the average height of all people in the database. Instead the data distribution that we assume the auditor and the attacker share is supposed to represent such common sense facts and it allows for more useful information to be released. There are many scenarios where such an assumption can be realistic. For example, distributions of attributes such as age or salary may be known from previous data releases or even published by the auditor itself.

The understanding of how to formally define privacy in the setting of statistical datasets has significantly evolved since the first publication of our work [17], and a new notion called differential privacy [11] has emerged. One nice property of this definition is that it does not assume the existence of an underlying distribution. The possible downside is that it provably excludes the possibility of giving exact answers, i.e., answers without noise. Consequently, differential privacy may be inadequate in applications where exact answers are required. It is plausible that a privacy guarantee with similar properties exists for the auditing setting – we leave this as an open problem for future work.

3. Examples where denials leak information

We first demonstrate how the $\texttt{max}$ auditor of [4] can be used by an attacker to breach the privacy of 1/8 of the data set. The same attack works also for $\texttt{sum}/\texttt{max}$ auditors. Then we demonstrate how an attacker can use the approximate online auditing problem of [18] to breach the privacy of 1/2 of the data set. Finally, while all the examples assume that the attacker has the ability to pose queries involving arbitrary subsets of the data, we demonstrate that an attacker who only poses SQL queries can still compromise the data.

Neither [4] nor [18] explicitly state what their algorithms do in the event a query is denied. One interpretation is that once a query is denied, every query thereafter will also be denied. Under this interpretation, auditors have almost no utility since a denial of service attack can be mounted with a first singleton query, e.g., $q_1 = \texttt{max}(x_3)$ or $q_1 = \texttt{sum}(x_3)$ – such a query will be denied, and so will every other one henceforward. Another interpretation is that once a query is denied, it is treated as if it was never posed. We use this latter interpretation, although some other interpretation may have been intended.

Finally, we assume that the attacker knows the auditor's algorithm for deciding denials. This is a standard assumption employed in the cryptography community known as *Kerckhoffs' Principle* – despite the fact that the actual privacy-preserving auditing algorithm is public, an attacker should still not be able to breach privacy.

3.1. $\texttt{max}$ auditing breach

Prior to describing the breach, we describe at a high level the $\texttt{max}$ auditing algorithm of [4]. Given a sequence of $\texttt{max}$ queries and corresponding answers, a method is given for compressing the sequence into a short representation that has $O(n)$ size, assuming the private data set consists of distinct values. Each element of the synopsis is a predicate of the form $\texttt{max}(S_i) < M$ or $\texttt{max}(S_i) = M$ where each S_i is a subset of indices. Since there are no duplicates in the data set, the method ensures that query sets of predicates in the synopsis are pairwise disjoint, so that the size of the synopsis is $O(n)$. It suffices to consider just these predicates while checking for privacy breach instead of the entire sequence of past queries. When a new query S_t is posed to the auditor, the real answer to the query M_t is computed and then the synopsis is efficiently updated so that the sets are pairwise disjoint. A breach is shown to occur if and only if there is an S_i such that $|S_i| = 1$

and the synopsis contains a predicate of the form $\max(S_i) = M$. In such a case, the query is denied and we assume that the synopsis goes back to its previous state. Otherwise the query is answered and the synopsis remains the same.

In order to breach privacy, the attacker partitions the data set $X = \{x_1, \ldots, x_n\}$ into $n/4$ disjoint 4-tuples and considers each 4-tuple independently. In each of these 4-tuples, the attacker will use the exact answers and the denials to learn one of the four entries, with success probability $1/2$. Hence, on average the attacker will learn $1/8$ of the entries. Let x_1, x_2, x_3, x_4 be the entries in one 4-tuple. The attacker will first issue the query $\max(x_1, x_2, x_3, x_4)$. This query is never denied since it is disjoint from all previous queries and knowing the answer will not help to uniquely determine one of these 4 values. Let a be the answer to the query. For the second query, the attacker drops at random one of the four entries, and queries the maximum of the other three. If this query is denied, then the dropped entry equals a. Otherwise, let a' $(= a)$ be the answer and drop another random element and query the maximum of the remaining two. If this query is denied, then the second dropped entry equals a'. If the query is allowed, we continue with the next 4-tuple.

Note that whenever the second or third queries are denied, the above procedure succeeds in revealing one of the four entries. If all the elements are distinct, then the probability we succeed in choosing the maximum value in the two random drops is $2/4$. Consequently, on average, the procedure reveals $1/8$ of the data.

3.2. Boolean sum auditing breach

Prior to describing the Boolean auditing attack, we describe the "conservative" approximate auditor of [18] that denies more often than it should. For each Boolean valued element x_i of the data set, the *trace* of x_i is defined to be the set of queries that involve x_i. The claim in [18] is that if for each variable x_i there is a corresponding variable x_j such that $x_i \neq x_j$ and x_i and x_j share the same trace (that is, the set of queries that involves x_i is identical to the set of queries that involves x_j), then no value is uniquely determined. The trace is only updated when a query is exactly answered, and not when a query is denied.

Next we demonstrate how an attacker can compromise $1/2$ of the data in expectation with the conservative approximate auditor of [18]. In this example, we assume that the data is $1/2$ 0s and $1/2$ 1s.[4] The attacker randomly permutes the entries and then partitions the data set $X = \{x_1, \ldots, x_n\}$ into $n/2$ disjoint 2-tuples. The attacker then poses the queries $\mathtt{sum}(x_i, x_{i+1})$ for odd i. If the query is answered then $x_i \neq x_{i+1}$, but the pair is forever ignored. And, if the query is denied, then the attacker can deduce that $x_i = x_{i+1}$ and furthermore since the trace is not updated on a denied query, future queries can be posed about these values. At the end of this process, the attacker has pairs that are equal, but does not know if they are both 0s or both 1s.

Assume without loss of generality that the queries that were denied involved the values $x_1, \ldots, x_m$ where $m = n/2$ in expectation. The attacker now asks queries $\mathtt{sum}(x_i, x_{i+1})$ for even i. If such a query is denied, then $x_{i-1} = x_i = x_{i+1} = x_{i+2}$. Otherwise, if the query is answered, then $x_{i-1} = x_i$ are different from $x_{i+1} = x_{i+2}$. At the end of this process, the private values can be partitioned into two sets, one set having all 0s and the other all 1s, but the attacker will not know which side has which value.

To determine this final bit of information, note that in expectation about $1/2$ of the previous queries were denied. Let x_j be a value that is different from x_1 that was in a denied query. We ask the final query $x_1 + x_j + x_m$. Note that this query will be answered since $x_1 \neq x_j$ and they have the same trace, and similarly for x_m. Suppose the answer to the query is a. If $x_m = x_1$ then we can determine x_1 by solving for x_1 in the equation $x_1 + (1 - x_1) + x_1 = a$. Otherwise, if $x_m \neq x_1$, then we can solve for x_1 in the equation $x_1 + (1 - x_1) + (1 - x_1) = a$. Once the attacker knows the value of x_1, all the other values ($1/2$ of the data set in expectation) can now also be uniquely determined.

3.3. SQL queries

Whereas the previous examples assume that the attacker could pose queries about arbitrary subsets of the data, we next show that even if we weakened the attacker by only allowing queries over attributes of the data, data can still be compromised. For example, consider the three queries, `sum`, `count` and `max` (in that order) on the 'salary' attribute, all conditioned on the same predicate involving other attributes. Since the selection condition is the same, all the three queries act on the same subset of tuples. Suppose that the first two queries are answered. Then the `max` query is denied whenever its value is exactly equal to the ratio of the `sum` and the `count` values (which happens when all the selected tuples have the same salary value). Hence the attacker learns the salary values of all the selected tuples whenever the third query is denied. Even though the attacker may not know the identities of these selected tuples, learning the private values of many individuals can still be considered a significant breach.

4. Simulatable auditing

Intuitively, denials leak because users can ask *why* a query was denied, and the reason is in the data. If the decision to allow or deny a query depends on the actual data, it reduces the set of possible consistent solutions for the underlying data.

[4] The attack can be modified to work for $0 < p < 1$ when there is a p fraction of 0s and $1 - p$ fraction of 1s – and p is not known to the attacker. In such a case, with high probability, the attacker can breach even more data: a $1 - 2p(1 - p)$ $(\geqslant 1/2)$ fraction, in expectation.

A naive solution to the leakage problem is to deny whenever the offline algorithm would, and to also randomly deny queries that would normally be answered. While this solution seems appealing, it has its own problems. Most importantly, although it may be that denials leak less information, leakage is not generally prevented. Furthermore, the auditing algorithm would need to remember which queries were randomly denied, since otherwise an attacker could repeatedly pose the same query until it was answered. A difficulty then arises in determining whether two queries are equivalent. The computational hardness of this problem depends on the query language, and may be intractable, or even undecidable.

To work around the leakage problem, we make use of the simulation paradigm which is used in cryptography (starting with the definition of semantic security [15]). The idea is the following: The reason that denials leak information is because the auditor uses information that is not available to the attacker (the answer to the newly posed query). In particular, this results in a computation the attacker could not perform by himself. A successful attacker capitalizes on this leakage to gain information. We introduce a notion of auditing where the attacker *provably* cannot gain any new information from the auditor's decision. This is formalized by requiring that the attacker is able to simulate or mimic the auditor's decisions. In such a case, because the attacker can equivalently decide if a query would be denied, denials do not leak information.

4.1. A formal definition of simulatable auditing

We begin by formally defining an auditor and a simulatable auditor.

Definition 4.1. An *auditor* is a function that given any data set X and any sequence of queries $q_1, \ldots, q_t$, either gives an exact answer to the query q_t or denies the answer.

Definition 4.2. Let X be any data set, let $\mathcal{Q}_t = \langle q_1, \ldots, q_t \rangle$ be any sequence of queries, and let $\mathcal{A}_t = \langle a_1, \ldots, a_t \rangle$ be the corresponding answers according to data set X. An *auditor B is simulatable* if there exists another auditor B^* that is a function of $\mathcal{Q}_t$ and $\mathcal{A}_{t-1}$, and the outcome of B on $\mathcal{Q}_t$, $\mathcal{A}_t$ and X is equal to that of B^* on $\mathcal{Q}_t$, $\mathcal{A}_{t-1}$.

All the auditors we design are trivially simulatable since the only information they use is $\mathcal{Q}_t$ and $\mathcal{A}_{t-1}$ – information that is readily available to an attacker.

Note that a simulatable auditor need not remember which queries where denied: queries that were denied will continue to be denied. Furthermore, denied queries do not influence which future queries will be answered or denied. To see why, consider the classical definition of compromise where a breach occurs if a sensitive value can be uniquely determined. When the query q_t is posed, the only information on the data set X available to the attacker is encoded in the previous queries and answers, that is, $\{\langle q_1, a_1 \rangle, \ldots, \langle q_{t-1}, a_{t-1} \rangle\}$. Suppose q_t is denied. The decision to deny q_t is based only on q_t and the previous queries and answers, and was made because there is some answer to q_t that could uniquely determine some data element x_l. Note that the query q_t will be denied even if it is posed at a future time. This is because at a later time, even more queries and answers could be provided and if x_l could be uniquely determined before, it will continue to be uniquely determined after. Further, the decision to deny q_t does not reveal any new information to the attacker, and hence the attacker's knowledge about the data set X remains the same even if denied queries are not remembered. Hence denied queries need not be taken into account in the auditor's subsequent decisions. We thus assume without loss of generality that $q_1, \ldots, q_{t-1}$ were all answered.

Simulatable auditors improve upon methods that completely ignore all previous query answers [9] in that longer query sequences can now be answered (an example is given in Section 5.4) and improve upon the use of offline algorithms to solve the online problem since denials do not leak information as shown in [17].

Randomized versions of Definitions 4.2 and 4.1 are also possible, where B and B^* are probabilistic computations, equality in Definition 4.2 is replaced by a statistical closeness guarantee on B and B^* and the auditor can make use of an underlying probability distribution. We discuss such a randomized variant in Section 6.

4.2. A perspective on auditing

We cast related work on auditing based on two important dimensions: utility and privacy (see Fig. 1). It is interesting to note the relationship between the information an auditor uses and its utility – the more information used, the longer query sequences the auditor can answer. That is because an informed auditor need not deny queries that do not actually put privacy at risk. On the other hand, as we saw in Section 3, if the auditor uses too much information, some of this information may be leaked, and privacy may be adversely affected.

The oldest work on auditing includes methods that simply consider the size of queries and the size of the intersection between pairs of queries [9,25] (upper left hand corner of Fig. 1). Subsequently, the contents of queries were considered (such as the elementary row and column matrix operations suggested in [4,5]). We call these *monitoring* methods. Query monitoring only makes requirements about the queries, and is oblivious of the actual data entries. In other words, to decide whether a query q_t is allowed, the monitoring algorithm takes as input the query q_t and the previously allowed queries $q_1, \ldots, q_{t-1}$, but ignores the answers to all these queries. This obliviousness of the query answers immediately implies the safety of the auditing algorithm in the sense that query denials cannot leak information. In fact, a user need not even communicate with the data set to check which queries would be allowed, and hence these auditors are simulatable.

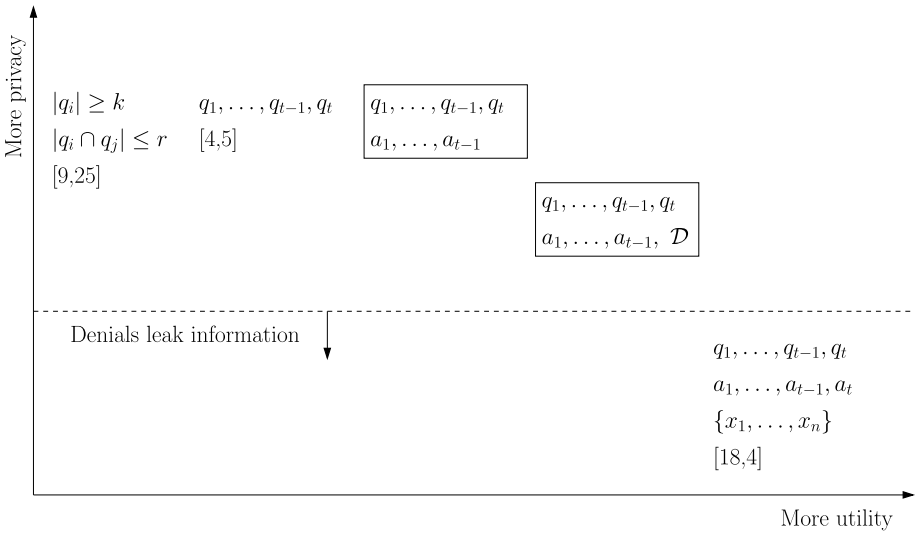

Fig. 1. Online query auditing approaches.

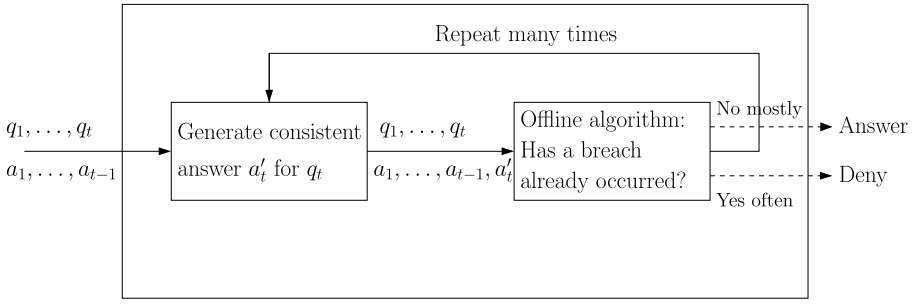

Fig. 2. General approach for designing simulatable auditors.

Other work on online auditing uses the queries $q_1, \ldots, q_t$ and all of their answers $a_1, \ldots, a_t$ [4,18] (bottom right corner of Fig. 1). While this approach yields more utility, we saw in Section 3 that denials leak private information.

Our work on simulatable auditing can be viewed as a 'middle point' (denoted in rectangular boxes in Fig. 1). Simulatable auditors use all the queries $q_1, \ldots, q_t$ and the answers to only the previous queries $a_1, \ldots, a_{t-1}$ to decide whether to answer or deny the newly posed query q_t. We will construct simulatable auditors that guarantee 'classical' privacy. We will also consider a variant of this 'middle point', where the auditing algorithm (as well as the attacker) has access to the underlying probability distribution.[5] With respect to this variant, we construct simulatable auditors that preserve privacy with high probability.

4.3. A general approach for constructing simulatable auditors

We next propose a general approach for constructing simulatable auditors that is useful for understanding our results and may also prove valuable for studying other types of queries.

The general approach (shown in Fig. 2) works as follows: Choose a set of answers to the last query q_t that are consistent with answers to previous queries (that is, consider all data sets satisfying answers to previous queries and compute answers to q_t for these data sets). For each of these answers, check if privacy is compromised. If compromise occurs then the query is denied. Otherwise, it is allowed. In the case of classical compromise for `max` simulatable auditing, we deterministically construct a small set of answers to the last query q_t so that if any one leads to compromise, then we deny the answer and otherwise we give the true answer. In the case of probabilistic compromise for `sum` queries, we randomly generate many consistent answers and if sufficiently many lead to compromise, then we deny the query and otherwise we answer the query.

Following the publication of [17], this general approach was also successfully used in [22,23].

[5] This model was hinted at informally in [8] and the following work [12].

5. Simulatable auditing algorithms, classical compromise

We next construct (tractable) simulatable auditors. We first describe how sum queries can be audited under the classical definition of privacy and then we describe how max queries can be audited under the same definition.

5.1. Simulatable auditing of sum queries

Observe that existing sum auditing algorithms (published in [4]) are already simulatable. In these algorithms, the data set is viewed as a vector of real numbers and each query is expressed as a vector of real numbers, so that the answer is computed as the dot product of the query vector and the data set vector. Consider the matrix whose rows correspond to the queries. Some value has been compromised if and only if this matrix can be reduced to a form where there is a row with one 1 and the rest 0s [4]. Such a transformation of the original matrix can be performed via elementary row and column operations, without looking at the data set or the answers. The reason this auditor is simulatable is that the answers to the queries as well as the data set are ignored when the matrix is transformed.

5.2. Simulatable auditing of max queries

We provide a simulatable auditor for the problem of auditing max queries over real-valued data. The data consists of a set of n values, $X = \{x_1, x_2, \ldots, x_n\}$ and the queries $q_1, q_2, \ldots$ are subsets of $\{1, 2, \ldots, n\}$. The answer corresponding to the query q_i is $a_i = \max\{x_j \mid j \in q_i\}$. Given a set of queries $q_1, \ldots, q_{t-1}$ and the corresponding answers $a_1, \ldots, a_{t-1}$ and the current query q_t, the simulatable auditor denies q_t if and only if there exists an answer a_t, consistent with $a_1, \ldots, a_{t-1}$, such that the answer helps to uniquely determine some element x_j. Since the decision to deny or answer the current query is independent of the real answer a_t, we should decide to answer q_t only if compromise is not possible for all consistent answers to q_t (as the real answer could be any of these). Conversely, if compromise is not possible for all consistent answers to q_t, it is safe to answer q_t.

5.2.1. Revisiting the max auditing breach of Section 3.1

We now return to the max auditing breach example of Section 3.1 and describe how a simulatable auditor would work. The first query $\max(x_1, x_2, x_3, x_4)$ is always answered since there is no answer, a_1 for which a value is uniquely determined. Suppose the second query is $\max(x_1, x_2, x_4)$. This query will always be denied since $x_3 = a_1$ whenever $a_2 < a_1$. In general, under the classical privacy definition, the simulatable auditor has to deny the current query even if only one consistent answer to q_t compromises privacy. Thus, many queries may be denied. This issue is addressed by our probabilistic definition of privacy in Section 6.

5.2.2. max simulatable algorithm

We now discuss how we obtain an algorithm for max simulatable auditing. A naive solution is to determine if for all possible answers a_t in $(-\infty, +\infty)$ whether (a) a_t is consistent and (b) whether some private element would be uniquely determined if a_t were the answer. Of course, such a naive algorithm is computationally expensive. Instead, we show that it is sufficient to test only a finite number of points. Let $q'_1, \ldots, q'_l$ be the previous queries that intersect with the current query q_t, ordered according to the corresponding answers, $a'_1 \leqslant \cdots \leqslant a'_l$. Let $a'_{lb} = a'_1 - 1$ and $a'_{ub} = a'_l + 1$ be the bounding values. Our algorithm checks only $2l + 1$ values: the bounding values, the above l answers, and the mid-points of the intervals determined by them.

Algorithm 1 max simulatable auditor.

1: Input: (allowed) queries and answers q_i and a_i for $i = 1, \ldots, t-1$, a new query q_t.
2: Let $q'_1, \ldots, q'_l$ be the previous queries that intersect with q_t, ordered according to the corresponding answers, $a'_1 \leqslant \cdots \leqslant a'_l$. Let $a'_{lb} = a'_1 - 1$ and $a'_{ub} = a'_l + 1$.
3: **for** $a_t \in \{a'_{lb}, a'_1, \frac{a'_1+a'_2}{2}, a'_2, \frac{a'_2+a'_3}{2}, a'_3, \ldots, a'_{l-1}, \frac{a'_{l-1}+a'_l}{2}, a'_l, a'_{ub}\}$ **do**
4: **if** (a_t is consistent with the previous answers $a_1, \ldots, a_{t-1}$) AND ($\exists 1 \leqslant j \leqslant n$ such that x_j is uniquely determined) **then**
5: Output "Deny" and **return**
6: **end if**
7: **end for**
8: Output "Answer" and **return**

We now prove that the algorithm works as desired.

Theorem 5.1. *Algorithm 1 is a* max *simulatable auditor that runs in time* $O(t \sum_{i=1}^{t} |q_i|)$ *where* t *is the number of queries.*

We begin by describing how to determine if a private element is uniquely determined (from [18]). In order to do so, we need to introduce the notion of an *extreme element*.

Definition 5.1. Given a set of queries and answers, the upper bound, μ_j for an element x_j is defined to be the minimum over the answers to the queries containing x_j, i.e., $\mu_j = \min\{a_k \mid j \in q_k\}$. In other words, μ_j is the best possible upper bound for x_j that can be obtained from the answers to the queries. We say that j is an *extreme element* for the query set q_k if $j \in q_k$ and $\mu_j = a_k$.

This means that the upper bound for x_j is realized by the query set q_k, i.e., the answer to every other query containing x_j is greater than or equal to a_k. The upper bounds of all elements as well as the extreme elements of all the query sets can be computed in $O(\sum_{i=1}^{t} |q_i|)$ time. Since the input includes both the data set and the queries, the time for the above computation is linear in the input size.

Lemma 5.2. *(See [18].) A value x_j is uniquely determined if and only if there exists a query set q_k for which j is the only extreme element.*

Hence, for a given value of a_t, we can check if $\exists 1 \leqslant j \leqslant n$ such that x_j is uniquely determined.

Next we explain how to determine if an answer to the current query is consistent with the previous queries and answers.

Lemma 5.3. *An answer a_t to query q_t is consistent if and only if every query set has at least one extreme element.*

Proof. Suppose that some query set q_k has no extreme element. This means that the upper bound of every element in q_k is less than a_k. This cannot happen since some element has to equal a_k. Formally, $\forall j \in q_k$, $x_j \leqslant \mu_j < a_k$ which is a contradiction.

Conversely, if every query set has at least one extreme element, setting $x_j = \mu_j$ for $1 \leqslant j \leqslant n$ is consistent with all the answers. This is because, for any set q_k with s as an extreme element, $x_s = a_k$ and $\forall j \in q_k$, $x_j \leqslant a_k$. □

In fact, it is enough to check the condition in the above lemma for q_t and the query sets intersecting it (instead of all the query sets).

We next show that Algorithm 1 preserves consistency when considering only the mid-point value, and then prove that checking whether x_j is uniquely determined at the mid-point value (instead of every point in the interval) is sufficient.

Lemma 5.4. *Values of a_t in (a'_s, a'_{s+1}) are either all consistent or all inconsistent with the previous queries and answers.*

Proof. Suppose that some value of a_t in (a'_s, a'_{s+1}) is inconsistent. Then, by Lemma 5.3, some query set q_α has no extreme element. We consider two cases:

- $q_\alpha = q_t$: This means that the upper bound of every element in q_t was $< a_t$ and hence $\leqslant a'_s$. This would be the case even for any value of a_t in (a'_s, a'_{s+1}).
- q_α intersects with q_t: This means that $a_t < a_\alpha$ and the extreme element(s) of q_α became no longer extreme for q_α by obtaining a reduced upper bound due to a_t. This would be the case even for any value of a_t in (a'_s, a'_{s+1}). □

Lemma 5.5. *Suppose that all values of a_t in (a'_s, a'_{s+1}) are consistent with the previous queries and answers. For $1 \leqslant j \leqslant n$, 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 $a_t = \frac{a'_s + a'_{s+1}}{2}$.*

Proof. Observe that revealing a_t can only affect elements in q_t and the queries intersecting it. This is because revealing a_t can possibly lower the upper bounds of elements in q_t, thereby possibly making some element in q_t or the queries intersecting it the only extreme element of that query set. Revealing a_t does not change the upper bound of any element in a query that is disjoint from q_t and hence does not affect elements in such sets.

Hence it is enough to consider $j \in q_t \cup q'_1 \cup \cdots \cup q'_l$. We consider the following cases:

- $q_t = \{j\}$: x_j is breached irrespective of the value of a_t.
- j is the only extreme element of q_t and $|q_t| > 1$: Consider two arbitrary points, β and γ that lie in the interval (a'_s, a'_{s+1}). We will show that if x_j becomes uniquely determined when a_t equals β, then x_j becomes uniquely determined even when a_t equals γ. It then follows that 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 $a_t = \frac{a'_s + a'_{s+1}}{2}$.
  Suppose that x_j becomes uniquely determined when a_t equals β. This means that each element indexed in $q_t \setminus \{j\}$ had an upper bound $< a_t$ and hence $\leqslant a'_s$ (since an upper bound can only be one of the answers given so far). Since this holds even when a_t equals γ, j is still the only extreme element of q_t and hence x_j is still uniquely determined.
- j is the only extreme element of q'_k for some $k \in [1, l]$: Consider two arbitrary points, β and γ that lie in the interval (a'_s, a'_{s+1}). Suppose that x_j becomes uniquely determined when a_t equals β. This means that $a_t < a'_k$ (and hence

$a'_{s+1} \leqslant a'_k$) and revealing a_t reduced the upper bound of some element indexed in $q'_k \setminus \{j\}$. This would be the case even when a_t equals γ. In other words, x_j becomes uniquely determined even when a_t equals γ. As in the previous case, it follows that 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 $a_t = \frac{a'_s + a'_{s+1}}{2}$. □

Thus, it suffices to check for $a_t = \frac{a'_s + a'_{s+1}}{2}$ $\forall 1 \leqslant s < l$ together with $a_t = a'_s$ $\forall 1 \leqslant s \leqslant l$ and also representative points, $(a'_1 - 1)$ in $(-\infty, a'_1)$ and $(a'_l + 1)$ in (a'_l, ∞). Note that a representative point is inconsistent if and only if its corresponding interval is inconsistent.

As noted earlier, the upper bounds of all elements as well as the extreme elements of all the query sets and hence each iteration of the for loop in Algorithm 1 can be computed in $O(\sum_{i=1}^{t} |q_i|)$ time (which is linear in the input size). As the number of iterations is $2l + 1 \leqslant 2t$, the running time of the algorithm is $O(t \sum_{i=1}^{t} |q_i|)$, proving Theorem 5.1. In the above analysis, we ignore the time needed to maintain the sorted list of answers since the insertion to this list takes only $O(\log t)$ time for each query, and this insertion time is dominated by the computation time of the for loop.

5.3. Improving the running time of the `max` simulatable auditor

We have shown that it is sufficient to test a boundable number of answers to the query q_t. A next natural question is whether we can speed up the algorithm by doing binary search through this set of answers. In particular, does an inconsistent answer to a query imply that all values smaller (or larger) are also inconsistent? It turns out that we can do a form of binary search, although not as simply as the previous question implies. In this section, we show how to improve the running time to $O((\log t) \sum_{i=1}^{t} |q_i|)$ where t is the number of queries.

We give four conditions each of which exhibits a monotonicity property. Two of the conditions pin down an interval of consistent answers. The other two conditions pin down an interval of answers where no value can be uniquely determined. If for every consistent answer, no value can be uniquely determined then the query is safe to answer. Consequently we can answer a query if the interval of consistent answers is contained in the interval of answers where no value is uniquely determined.

The four conditions are:

- $\mathcal{A} \equiv$ "$a_t = \theta$ is inconsistent because some query set q_k (different from the current query q_t) has no extreme element",
- $\mathcal{B} \equiv$ "$a_t = \theta$ is inconsistent because the current query set q_t has no extreme element",
- $\mathcal{C} \equiv$ "For $a_t = \theta$, x_j is uniquely determined for some $j \notin q_t$",
- $\mathcal{D} \equiv$ "For $a_t = \theta$, x_j is uniquely determined for some $j \in q_t$".

Suppose that the $2l + 1$ values to be checked for a_t in step 3 of Algorithm 1 are arranged in increasing order in an array. As we go from left to right in this array, we will show that condition $\mathcal{A}$ holds for all array elements below some index and does not hold for elements thereafter. Consequently, we can determine the above index by performing a binary search in the array and checking for condition $\mathcal{A}$ at each fork. Condition $\mathcal{C}$ also exhibits monotonicity in the same direction as $\mathcal{A}$. Thus, for a condition $P \in \{\mathcal{A}, \mathcal{C}\}$, let $binarySearch(Left, P)$ denote the index of the leftmost array element for which the condition P does not hold. Conditions $\mathcal{B}$ and $\mathcal{D}$ also exhibit monotonicity, but in the opposite direction. Again, for a condition $P \in \{\mathcal{B}, \mathcal{D}\}$, we use binary search to obtain $binarySearch(Right, P)$ which is the index of the rightmost array element for which the condition P does not hold.

Algorithm 2 Faster `max` simulatable auditor.

```
1: α ← binarySearch(Left, A)
2: β ← binarySearch(Right, B)
3: γ ← binarySearch(Left, C)
4: δ ← binarySearch(Right, D)
5: if [α, β] ⊆ [γ, δ] then
6:     Output "Answer" and return
7: else
8:     Output "Deny" and return
9: end if
```

Step 5 of Algorithm 2 checks whether for every consistent answer a_t no value x_j is uniquely determined. The array elements with indices in the range $[\alpha, \beta]$ correspond to consistent answers for the current query whereas those outside this range are inconsistent. Further, there is at least one value of a_t that is consistent, i.e., $\alpha \leqslant \beta$. Similarly, the array elements with indices in the range $[\gamma, \delta]$ correspond to answers for which no data value is uniquely determined whereas some x_j is uniquely determined for those outside this range. Hence it is safe to answer the current query if for every consistent answer, no data value is uniquely determined, i.e., the interval $[\alpha, \beta]$ is fully contained in the interval $[\gamma, \delta]$. The running time of the algorithm is $O((\log t) \sum_{i=1}^{t} |q_i|)$, since this is the time taken by each binary search step.

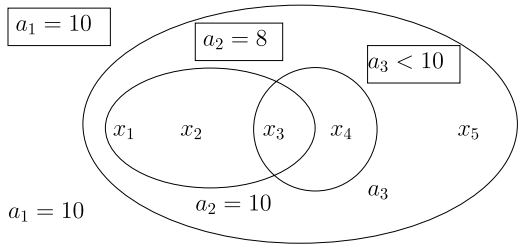

Fig. 3. max simulatable auditor is more useful than max query restriction auditor. The values within the boxes correspond to the second scenario.

We now prove the main lemma.

Lemma 5.6.

(1) *If $\mathcal{A}$ then any value of $a_t < \theta$ is also inconsistent.*
(2) *If $\mathcal{B}$ then any value of $a_t > \theta$ is also inconsistent.*
(3) *If $\mathcal{C}$ then x_j is uniquely determined for any value of $a_t < \theta$.*
(4) *If $\mathcal{D}$ then x_j is uniquely determined for any value of $a_t > \theta$.*

Proof. *Part* 1: Since the sequence of the first $t-1$ queries and answers was consistent and $a_t = \theta$ is inconsistent, it follows from Lemma 5.3 that some query set q_k, which had some extreme element(s) earlier, no longer has any extreme element. This means that the extreme element(s) of q_k is no longer extreme for q_k because $a_t = \theta$ reduces the upper bound. In such a case, any value $a_t < \theta$ is also inconsistent.

Part 2: If $a_t = \theta$ is inconsistent because of the current query q_t then the upper bound of every element in q_t is less than $a_t = \theta$. Thus if $a_t > \theta$, q_t still has no extreme element and the answer would still be inconsistent.

Part 3: Since x_j is uniquely determined, it follows from Lemma 5.2 that there exists a query set q_k (different from q_t) for which j is the only extreme element. This means that $\theta = a_t < a_k$ and revealing a_t reduced the upper bound of some element indexed in $q_k \setminus \{j\}$. This would be the case even for any value of $a_t < \theta$, so that j is still the only extreme element for q_k and hence x_j is still uniquely determined.

Part 4: As in the proof of Lemma 5.5, $q_t = \{j\}$ is a trivial case. Hence assume that $|q_t| > 1$ and j is the only extreme element of q_t. This means that each element indexed in $q_t \setminus \{j\}$ has an upper bound less than $a_t = \theta$. This would be the case even for any value of $a_t > \theta$, so that j is still the only extreme element for q_t and hence x_j is still uniquely determined, by Lemma 5.2. □

Consequently, we have the following theorem.

Theorem 5.7. *Algorithm 2 is a* max *simulatable auditor that runs in time $O((\log t)\sum_{i=1}^{t}|q_i|)$ where t is the number of queries.*

5.4. Utility of max *simulatable auditor vs. monitoring*

While both simulatable auditors and monitoring methods are safe, simulatable auditors potentially have greater utility, as shown by the following example (see Fig. 3).

Consider the problem of auditing max queries on a data set containing 5 elements. We will consider three queries and two possible sets of answers to the queries. We will demonstrate that the simulatable auditor answers the third query in the first case and denies it in the second case while a query monitor (which makes the decisions based only on the query sets) has to deny the third query in both cases. Let the query sets be $q_1 = \{1,2,3,4,5\}$, $q_2 = \{1,2,3\}$, $q_3 = \{3,4\}$ in that order. Suppose that the first query is answered as $a_1 = 10$. We consider two scenarios based on a_2. (1) If $a_2 = 10$, then every query set has at least two extreme elements, irrespective of the value of a_3. Hence the simulatable auditor will answer the third query. (2) Suppose $a_2 = 8$. Whenever $a_3 < 10$, 5 is the only extreme element for S_1 so that $x_5 = 10$ is determined. Hence it is not safe to answer the third query.

While the simulatable auditor provides an answer q_3 in the first scenario, a monitor would have to deny q_3 in both, as its decision is oblivious of the answers to the first two queries.

6. Probabilistic compromise

We next describe a definition of privacy that arises from some of the previously noted limitations of classical compromise. On the one hand, classical compromise is a weak definition since if a private value can be deduced to lie in a tiny interval – or even a large interval where the distribution is heavily skewed towards a particular value – it is not considered

a privacy breach. On the other hand, classical compromise is a strong definition since there are situations where no query would ever be answered. This problem has been previously noted [18]. For example, if the data set contains items known to have a lower bound, e.g., age, then no `sum` or `max` query would ever be answered. For instance, the query $\texttt{sum}(x_1, \ldots, x_n)$ (as well as the query $\texttt{max}(x_1, \ldots, x_n)$) would not be answered since there exists a data set, e.g., $x_i = 0$ for all i where a value, in fact all values, can be uniquely determined.

To work around these issues, we propose a definition of privacy that bounds the change in the ratio of the posterior probability that a value x_i lies in an interval I given the queries and answers to the prior probability that $x_i \in I$. This definition is related to the notion of semantic security [15] and to definitions suggested in the perturbation literature including [14,8,12].

6.1. Privacy definition

Consider an arbitrary data set $X = \{x_1, \ldots, x_n\}$, in which each x_i is chosen independently according to the same distribution $\mathcal{H}$ on $(-\infty, \infty)$. Let $\mathcal{D} = \mathcal{H}^n$ denote the joint distribution. Let the queries be denoted as $q_j = (Q_j, f_j)$, for $j = 1, \ldots, t$ where $Q_j \subseteq [n]$ specifies a subset of the data set entries and f_j specifies a function (such as `sum` or `max`). The answer $a_j = f_j(Q_j)$ is f_j applied to the subset of entries $\{x_i \mid i \in Q_j\}$.

We next define the notion of λ-*safe*. We say that a sequence of queries and answers is λ-*safe* for an entry x_i and an interval I if the attacker's confidence that $x_i \in I$ does not change significantly upon seeing the queries and answers.

Definition 6.1. The sequence of queries and answers, $q_1, \ldots, q_t, a_1, \ldots, a_t$ is said to be λ-*safe* with respect to a data entry x_i and an interval $I \subseteq (-\infty, \infty)$ if the following Boolean predicate evaluates to 1:

$$\text{Safe}_{\lambda,i,I}(q_1, \ldots, q_t, a_1, \ldots, a_t) = \begin{cases} 1 & \text{if } 1/(1+\lambda) \leqslant \frac{\Pr_{\mathcal{D}}(x_i \in I \mid \bigwedge_{j=1}^{t}(f_j(Q_j) = a_j))}{\Pr_{\mathcal{D}}(x_i \in I)} \leqslant (1+\lambda), \\ 0 & \text{otherwise.} \end{cases}$$

We say that an interval J is β-*significant* if for every $i \in [n]$ the (a priori) probability that $x_i \in J$ is at least β. We will only care about probability changes with respect to significant intervals, due to the fact that requiring the a priori and posteriori probabilities of a private value to be close on arbitrarily small intervals would cause no queries to be answered at all. The definition below defines privacy in terms of a predicate that evaluates to 1 if and only if $q_1, \ldots, q_t$, $a_1, \ldots, a_t$ is λ-safe for all entries and all β-significant intervals:

$$\text{AllSafe}_{\lambda,\beta}(q_1, \ldots, q_t, a_1, \ldots, a_t) = \begin{cases} 1 & \text{if } \text{Safe}_{\lambda,i,J}(q_1, \ldots, q_t, a_1, \ldots, a_t) = 1, \text{ for every } i \in [n] \text{ and} \\ & \text{every } \beta\text{-significant interval } J, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

We note that this definition defines privacy with respect to not just all β-significant intervals, but also unions of such intervals. This is due to the fact that if a sequence of queries and answers is λ-safe with respect to two intervals I_1 and I_2, then it will also be λ-safe with respect to the union of the intervals, $I_1 \cup I_2$. (Consider the probability expression in Definition 6.1. For $j \in \{1, 2\}$, let a_j denote the numerator of this expression for interval I_j and b_j the denominator. We use $\frac{1}{1+\lambda} \leqslant \frac{a_1}{b_1}, \frac{a_2}{b_2} \leqslant 1+\lambda \Rightarrow \frac{1}{1+\lambda} \leqslant \frac{a_1+a_2}{b_1+b_2} \leqslant 1+\lambda$.)

We now turn to our privacy definition. Let $X = \{x_1, \ldots, x_n\}$ be the data set, in which each x_i is chosen independently according to the same distribution[6] $\mathcal{H}$ on $(-\infty, \infty)$. Let $\mathcal{D} = \mathcal{H}^n$ denote the joint distribution. Consider the following (λ, β, T)-*privacy game* between an attacker and an auditor, where in each round t (for up to T rounds):

(1) The attacker (adaptively) poses a query $q_t = (Q_t, f_t)$.
(2) The auditor decides whether to allow q_t or not. The auditor replies with $a_t = f_t(Q_t)$ if q_t is allowed and with $a_t =$ "denied" otherwise.
(3) The attacker wins if $\text{AllSafe}_{\lambda,\beta}(q_1, \ldots, q_t, a_1, \ldots, a_t) = 0$.[7]

Note that the predicate AllSafe is computed by both the attacker and the auditor using only the underlying distribution, that is, without looking at the sampled data set.

Definition 6.2. We say that an auditor is $(\lambda, \delta, \beta, T)$-*private* if for any attacker $\mathcal{A}$

$$\Pr\big[\mathcal{A} \text{ wins the } (\lambda, \beta, T)\text{-privacy game}\big] \leqslant \delta.$$

The probability is taken over the randomness in the distribution $\mathcal{D}$ and the coin tosses of the auditor and the attacker.

[6] Our analysis works even if each x_i is chosen independently from a different distribution. However, to simplify the notation, we assume that all values are drawn from the same distribution.

[7] Hereafter, we will refer to the predicates without mentioning the queries and answers for the sake of clarity.

Combining Definitions 4.2 and 6.2, we have our new model of simulatable auditing. In other words, we seek auditors that are simulatable and $(\lambda, \delta, \beta, T)$-private.

Note that, on the one hand, the definition of simulatable auditing prevents the auditor from using a_t in deciding whether to answer or deny the query. On the other hand, the privacy definition requires that regardless of what a_t was, with high probability, each data value x_i is still safe (as defined by $\text{AllSafe}_{\lambda,\beta}$). Consequently, it is important that the current query q_t be used in deciding whether to deny or answer. Because we cannot use the answer a_t, but want to use the query q_t, our auditor ignores the real answer a_t and instead makes guesses about the value of a_t obtained by randomly sampling data sets according to the distribution $\mathcal{D}$ conditioned on previous queries and answers.

6.2. Discussion on the privacy definition

We remark that Definition 6.2 assumes that the distribution from which the data is drawn is known to the attacker. However, in practice, the attacker may not be aware of the underlying distribution. Then the predicate AllSafe has to be evaluated with respect to the attacker's prior distribution, since a privacy breach occurs only if there is a substantial change in her beliefs. However if the true distribution and the attacker's distribution are arbitrarily different, any exact answer released by the auditor would result in partial disclosure of some private value. For example, consider a database that contains salary as a private attribute, and let the attacker's prior belief be that every person's salary exceeds \$10M. By querying the data, the attacker may learn that her belief is not true (causing a substantial change in her posterior distribution) resulting in massive privacy breach. Given that the attacker's prior beliefs are way off the mark, the auditor cannot truthfully answer any aggregate query without violating privacy, not even the average salary. Instead we assume that the auditor and the attacker share a common data distribution, representing common knowledge, thereby allowing useful information to be released. This assumption is realistic in many scenarios, for example, the distributions of attributes such as age or salary may be known from previously published data.

Definition 6.2 directly guarantees privacy only with respect to events that correspond to β-significant intervals. For other types of events, e.g., those corresponding to intervals that are not β-significant, only limited privacy is guaranteed. Consider the case where an extremely sensitive event corresponds to an interval with a priori probability $p \ll \beta$. The posterior probability can grow up to $\beta(1+\lambda)$, and the ratio of posterior to a priori probability of this event $\beta(1+\lambda)/p$ can hence be very large. Still, enough protection may be provided by choosing the parameters β, λ to be sufficiently small. We note that since the presentation of our ideas a stronger privacy standard has emerged – differential privacy [11] – whose guarantee also extends for low probability events.

6.3. Evaluating the predicate $\text{AllSafe}_{\lambda,\beta}$

Eq. (1) requires checking whether the sequence of queries and answers is λ-safe for infinitely many intervals (i.e., for all β-significant intervals). We next show that all such intervals J can be guaranteed to be λ-safe, by making sure that a *finite* number of intervals are safe.

We assume that the distribution $\mathcal{H}$ is specified such that we can obtain the partition $\mathcal{I}$ of $(-\infty, \infty)$ into ℓ intervals each with equal probability mass of $\frac{1}{\ell}$, i.e., $\Pr_{\mathcal{D}}(x_i \in I) = \frac{1}{\ell}$ for every interval $I \in \mathcal{I}$. For example, if $\mathcal{H}$ is specified as a cumulative distribution function, then we can perform binary search to obtain the points in $(-\infty, \infty)$ that define the above partition to the desired level of precision.

We show that if privacy is preserved in each regularly-weighted interval of probability mass $\frac{1}{\ell}$, then the privacy of any β-significant interval is also preserved. In other words, to guarantee that any β-significant interval J is λ-safe, we will ensure that every interval $I \in \mathcal{I}$ is $\tilde{\lambda}$-safe where $\tilde{\lambda}$ is smaller than λ. We provide a smooth trade-off: the finer-grained the partition (i.e., larger ℓ), the weaker the privacy requirements (i.e., larger $\tilde{\lambda}$) for each of the ℓ intervals and vice versa. The intuition is the following. The privacy guarantees of the intervals fully contained in J can be used to imply the privacy of J whereas the guarantees of the two bounding intervals (which are partially contained in J) cannot be used. Hence, if the partition is very fine-grained, J contains more intervals from the partition and hence weaker privacy requirements suffice.

Given λ and β, we can choose the trade-off parameter, c to be any integer greater than $1 + 2/\lambda$. Choosing c determines the parameters, $\ell = \lceil c/\beta \rceil$ and $\tilde{\lambda} = \frac{\lambda(c-1)-2}{c+1}$. We formally state and prove the lemma below.

Lemma 6.1. *If privacy is preserved for each of the ℓ intervals in $\mathcal{I}$ with parameter $\tilde{\lambda}$, then privacy of every β-significant interval is preserved with parameter λ.*

Proof. Note that any β-significant interval J is contained in the union of either $d+1$ or $d+2$ consecutive intervals in $\mathcal{I}$ where $d = \lfloor \ell \Pr_{\mathcal{D}}(x_i \in J) \rfloor$. Consider the case where J is contained in the union of $d+1$ intervals, $I_1, I_2, \ldots, I_{d+1}$, of which J contains the intervals, $I_2, I_3, \ldots, I_d$ (the argument for $d+2$ intervals follows similarly). Denote $\Pr_{\mathcal{D}}(x_i \in I_k)$ by $p_k = \frac{1}{\ell}$ and $\Pr_{\mathcal{D}}(x_i \in I_k \mid \bigwedge_{j=1}^{t} f_j(Q_j) = a_j)$ by q_k.

Since privacy is preserved for each of these $d+1$ intervals, each q_k is lower bounded by $\frac{1}{\ell(1+\tilde{\lambda})}$ and upper bounded by $\frac{1+\tilde{\lambda}}{\ell}$. Further the overlap of J with intervals in $\mathcal{I}$ implies bounds on a priori and posteriori probabilities: $\sum_{k=1}^{d+1} p_k \geqslant$

$\Pr_{\mathcal{D}}(x_i \in J) \geqslant \sum_{k=2}^{d} p_k$ and $\sum_{k=2}^{d} q_k \leqslant \Pr_{\mathcal{D}}(x_i \in J \mid \bigwedge_{j=1}^{t} f_j(Q_j) = a_j) \leqslant \sum_{k=1}^{d+1} q_k$. Combining these relationships, and noting that $d \geqslant c$, we get the desired result. □

7. Simulatable sum auditing, probabilistic compromise

In this section we consider the problem of auditing sum queries (where each query is of the form $\mathtt{sum}(Q_j)$ for a subset of the dimensions, Q_j) under the newly defined probabilistic definition of compromise.

Prior to describing the solution, we give some intuition. Assume for simplicity that each individual can take a value uniformly between $[0, 1]$. Then over n individuals, the data set $\{x_1, \ldots, x_n\}$ can be any point in the unit cube $[0, 1]^n$ with equal probability. A sum query and its corresponding answer induces a hyperplane. The data sets consistent with one sum query/answer are then those points in $[0, 1]^n$ that fall on this hyperplane. Each successive query/answer reduces the space of possible consistent data sets to those in the intersection of the induced hyperplanes that also fall in $[0, 1]^n$, i.e., the consistent data sets lie in a convex polytope. Because the prior distribution is uniform, the posterior distribution (given the queries and answers) inside the convex polytope is also uniform.

How can we audit sum queries? Following the general paradigm suggested in Fig. 2, given a sequence of queries and answers and given a new query q_t, we generate a consistent answer to q_t (without using the underlying data set). We do this by drawing a point uniformly at random from the convex polytope induced by the previous queries and answers. This point is a sample data set and we then compute a candidate answer to q_t based on this sample data set. Once we have an answer we can then determine whether a privacy breach has occurred: Suppose that P is the current convex polytope. To determine if a breach has occurred for a particular individual x_i and a particular interval I, consider the ratio in Definition 6.1: $\frac{\Pr(x_i \in I | \vec{x} \in P)}{\Pr(x_i \in I)} = \frac{\Pr(x_i \in I | \vec{x} \in P)}{|I|}$. We can estimate the probability in the numerator by sampling from the convex polytope P and estimating what fraction of the sample points lie inside I. If the fraction above is greater than $(1 + \lambda)$ or less than $\frac{1}{1+\lambda}$ then the query is unsafe for this sampled data set. To increase our certainty, we repeat the above process with many consistent datasets, i.e., many consistent answers to the query q_t. If many consistent answers lead to a privacy breach, we deny the answer and otherwise we give the exact answer. In fact, our algorithm in [17] for auditing sum queries under probabilistic compromise uses the above technique.

While this intuition was given in the context of the uniform distribution, there is nothing specific about the uniform distribution that this argument utilizes. Indeed, in the rest of this section, we will assume that the underlying data set is generated from a more general logconcave distribution. We use this distribution because there exist algorithms for approximately sampling from it. However, there is nothing about our general approach that limits its applicability to logconcave distributions.

7.1. Properties of logconcave distributions

The class of logconcave distributions forms a common generalization of uniform distributions on convex sets and Gaussian distributions. A distribution over a domain D is said to be logconcave if it has a density function f such that the logarithm of f is concave on its support. That is, the density function $f : D \to \mathcal{R}_+$ is *logconcave* if it satisfies $f(\alpha x + (1-\alpha)y) \geqslant f(x)^{\alpha} f(y)^{1-\alpha}$ for every $x, y \in D$ and $0 \leqslant \alpha \leqslant 1$. These distributions constitute a broad class and play an important role in stochastic optimization and economics of uncertainty and information [20,24,2].

We assume that each element x_i is independently drawn according to the same logconcave distribution H over $\mathcal{R}$. Let $\mathcal{D} = H^n$ denote the joint distribution. Since the product of two logconcave functions is logconcave, the joint distribution $\mathcal{D}$ is also logconcave. Moreover, as the queries and answers impose convex constraints and indicator functions of convex sets are logconcave, the posteriori distribution $\mathcal{D}_{Q,t}$ (which is $\mathcal{D}$ conditioned on $\bigwedge_{j=1}^{t}(\mathtt{sum}(Q_j) = a_j)$) is also logconcave. This is because the density function for the posteriori distribution can be expressed as the product of the density function for the a priori distribution and the indicator function corresponding to the convex constraints (scaled by a constant).[8]

Our algorithms make use of randomized, polynomial-time algorithms for sampling (with a small error) from a logconcave distribution (see for example [20]). We will use the following theorem.

Theorem 7.1. *There exists Algorithm Sample$(\mathcal{G}, \epsilon)$ for sampling from an arbitrary logconcave distribution $\mathcal{G}$ with running time of $O^*(n^k)$ oracle calls*[9] *(for some constant k) such that the sampled output follows a distribution $\mathcal{G}'$ where the total variation distance between $\mathcal{G}$ and $\mathcal{G}'$ is at most ϵ.*

The asterisk in the $O^*()$ notation indicates that (polynomial) dependence on $\log n$, and the error parameter, ϵ are not shown (similar to the notation used in [20]). The *total variation distance*, aka statistical difference between two proba-

[8] The density function for the posteriori distribution, $f_{\mathcal{D}_{Q,t}}$ is proportional to the a priori density, $f_{\mathcal{D}}$ inside the convex region C and is zero outside the region. Denoting the indicator function for the region by δ_C, we get $f_{\mathcal{D}_{Q,t}}(\cdot) = \frac{f_{\mathcal{D}}(\cdot) \times \delta_C(\cdot)}{\int_C f_{\mathcal{D}}(\vec{x})\, d\vec{x}}$.

[9] As in [20], we assume that the density function $f_{\mathcal{G}}$ is given by an oracle, that is, for any $x \in \mathcal{R}^n$, the oracle returns the value $f_{\mathcal{G}}(x)$. In our setting, we assume that the density function $f_{\mathcal{D}}$ corresponding to the a priori joint distribution $\mathcal{D}$ is given by an oracle. It follows that the posteriori density function $f_{\mathcal{D}_{Q,t}}$ can be computed using $O(nT)$ oracle calls. This is because checking whether a point lies inside the convex region defined by at most T hyperplanes takes $O(n)$ time per hyperplane.

bility distributions $\mathcal{G}$ and $\mathcal{G}'$ is the largest possible difference between the probabilities assigned to the same event, i.e., $\sup_E |\Pr_{\mathcal{G}}(E) - \Pr_{\mathcal{G}'}(E)|$. The current best known algorithm [20] for sampling from a logconcave distribution runs in time $O^*(n^5)$.

7.2. Estimating the predicate $\text{AllSafe}_{\lambda,\beta}$ using sampling

We begin by considering the situation when we have the answer to the last query a_t. Our auditors cannot use a_t, and we will show how to get around this assumption. Given a_t and all previous queries and answers, Algorithm 3 checks whether privacy has been breached. As described in Section 6.3, it is enough to make sure that each of the ℓ intervals in $\mathcal{I}$ is safe (for a suitable choice of c and hence ℓ in Lemma 6.1). For each x_i and each interval I in $\mathcal{I}$, the a priori probability that x_i lies in I is equal to $\frac{1}{\ell}$, from the definition of $\mathcal{I}$. To estimate the posteriori probability that x_i lies in I, we sample many data sets according to the posteriori distribution $\mathcal{D}_{Q,t}$ (which is logconcave) and compute the fraction of the data sets satisfying $x_i \in I$. This is done using Algorithm *Sample* from Theorem 7.1.

Algorithm 3 takes as input, the sequence of queries and answers and the parameters, λ (from Definition 6.1), η (probability of error), c (trade-off parameter from Lemma 6.1), β (from the notion of β-significant interval), and n (the size of the data set). $\tilde{\lambda}$ is defined in terms of λ as for Lemma 6.1. However, we check the privacy requirement with respect to a smaller parameter, $\lambda' = \tilde{\lambda}/3$. N denotes the number of data sets sampled and N_e denotes the number of data sets satisfying $x_i \in I$. The ratio, $\frac{N_e}{N}$ is an estimate of the posteriori probability that $x_i \in I$. As the a priori probability is equal to $1/\ell$, we require that the ratio, $\frac{\ell \cdot N_e}{N}$ be very close to 1. In Algorithm 3, let $\ell = \lceil c/\beta \rceil$, $\tilde{\lambda} = \frac{\lambda(c-1)-2}{c+1}$, $\lambda' = \frac{\tilde{\lambda}}{3}$, $N = \frac{9\ell^2 \ln(2/\eta)}{\tilde{\lambda}^2} \cdot (1+\tilde{\lambda}/3)^2 \cdot \max((1+\tilde{\lambda})^2, (3+\tilde{\lambda}/3)^2)$ and $\epsilon = \frac{\eta}{2N}$. The proof will illustrate why this choice of parameters works.

Algorithm 3 Safe.

```
Input: queries and answers q_j and a_j for j = 1, ..., t, the a priori distribution D, and parameters λ, η, c, β, n.
Let safe = true
for each x_i and for each interval I in I do
    Sample N data sets according to D_{Q,t}, using Algorithm Sample(D_{Q,t}, ε)
    Let N_e be the number of data sets satisfying x_i ∈ I
    if ((ℓ·N_e)/N ∉ [1/(1+λ'), 1+λ']) then
        Let safe = false
    end if
end for
Return safe
```

We now prove that Algorithm 3 behaves as desired. Ideally, we would like to prove that if the query is not safe then we deny the query and if the query is safe then we answer the query. However our claims will not be that strong for a few reasons: (a) we do not check all (in fact, infinitely many) β-significant intervals for privacy and instead check only ℓ intervals in the partition (b) we estimate the posteriori probability using sampling and then use Chernoff bounds and (c) we cannot sample exactly from the underlying logconcave distribution. Consequently, with probability close to 1, whenever $\text{AllSafe}_{\lambda,\beta} = 0$ we deny the query and for a smaller privacy parameter, $\tilde{\lambda}/9$ and larger significance parameter, ℓ, whenever $\text{AllSafe}_{\tilde{\lambda}/9,\ell} = 1$ we answer the query. For the region in between, we make no guarantees. The following lemma is proved in Appendix A.

Lemma 7.2.

(1) *If* $\text{AllSafe}_{\lambda,\beta} = 0$ *then Algorithm* Safe *returns* false *with probability at least* $1 - \eta$.
(2) *If* $\text{AllSafe}_{\tilde{\lambda}/9,\ell} = 1$ *then Algorithm* Safe *returns* true *with probability at least* $1 - 2n\ell\eta$.

From now on we assume that Algorithm Safe always works, that is, it always returns *false* when $\text{AllSafe}_{\lambda,\beta} = 0$ and always returns *true* when $\text{AllSafe}_{\lambda'',\lceil c/\beta \rceil} = 1$, for some $\lambda'' < \lambda$. This is because the error guarantees of Lemma 7.2 can be made exponentially small using standard methods (repetition and majority vote) and further the second part of the lemma can be proved with any value $\lambda'' < \tilde{\lambda}$ (instead of $\tilde{\lambda}/9$).

7.3. Constructing the simulatable auditor

Without loss of generality, we assume that the input $q_1, \ldots, q_{t-1}, a_1, \ldots, a_{t-1}$ contains only queries allowed by the auditor. As the auditor is simulatable, denials do not leak any information (and hence do not change the conditional probability on data sets) beyond what the previously allowed queries already leak. Each data set sampled follows a distribution which is within a total variation distance, ϵ from the desired conditional distribution, $\mathcal{D}_{Q,t-1}$. For a data set X and query Q, let $\text{sum}_X(Q) = \sum_{i \in Q} X(i)$.

Algorithm 4 sum simulatable auditor.

1: Input: (allowed) queries and answers q_j and a_j for $j = 1, \ldots, t-1$, a new query q_t, the a priori distribution $\mathcal{D}$, and parameters λ, λ'', η, c, β, n, δ, T.
2: Let $\epsilon = \frac{\delta}{10T}$
3: **for** $O(\frac{T}{\delta}\log\frac{T}{\delta})$ times **do**
4: Sample a data set X' according to $\mathcal{D}_{Q,t-1}$, using Algorithm *Sample*$(\mathcal{D}_{Q,t-1}, \epsilon)$
5: Let $a'_t = \mathtt{sum}_{X'}(Q_t)$
6: Evaluate Algorithm SAFE on input $q_1, \ldots, q_t, a_1, \ldots, a_{t-1}, a'_t, \mathcal{D}$ and parameters λ, λ'', η, c, β, n
7: **end for**
8: **if** the fraction of sampled data sets for which Algorithm SAFE returned *false* is more than $\frac{\delta}{2T}$ **then**
9: Output "Deny" and **return**
10: **else**
11: Output "Answer" and **return**
12: **end if**

Theorem 7.3. *Algorithm 4 is a $(\lambda, \delta, \beta, T)$-private simulatable auditor.*

Proof. By Definition 6.2, the attacker wins the game in round t if he poses a query q_t for which $\text{AllSafe}_{\lambda,\beta}(q_1, \ldots, q_t, a_1, \ldots, a_t) = 0$ and the auditor does not deny q_t.

Consider first an auditor that allows every query. Given answers to the first $t-1$ queries, the true data set is distributed according to the distribution $\mathcal{D}$, conditioned on these answers. Given q_t (but not a_t), the probability the attacker wins hence equals

$$p_t = \Pr_{\mathcal{D}}\left[\text{AllSafe}_{\lambda,\beta}\begin{pmatrix} q_1, \ldots, q_t, \\ a_1, \ldots, a_{t-1}, \\ \mathtt{sum}_{X''}(q_t) \end{pmatrix} = 0 \;\middle|\; q_1, \ldots, q_{t-1}, a_1, \ldots, a_{t-1}\right]$$

where X'' is a data set drawn truly according to $\mathcal{D}_{Q,t-1}$ (i.e., $\mathcal{D}$ conditioned on $q_1, \ldots, q_{t-1}, a_1, \ldots, a_{t-1}$). Let $\tilde{p}_{t,j}$ denote the corresponding probability under the distribution sampled by Algorithm *Sample* in the jth iteration. From Theorem 7.1, $\tilde{p}_{t,j} \geqslant p_t - \epsilon$. Let $\tilde{p}_t = \min_j \tilde{p}_{t,j}$ so that $\tilde{p}_t \geqslant p_t - \epsilon$. Note that, since $a'_t = \mathtt{sum}_{X'}(Q_t)$ is precisely what the algorithm computes, the algorithm essentially estimates $\tilde{p}_t$ via multiple draws of random data sets X' from $\mathcal{D}_{Q,t-1}$.

Our auditor, however, may deny answering q_t. First consider the case when $p_t > \frac{\delta}{T}$ so that $\tilde{p}_t > \frac{9\delta}{10T}$. Then, by the Chernoff bound, the fraction computed in step 8 is expected to be higher than $\frac{\delta}{2T}$, with probability at least $1 - \frac{\delta}{T}$. Hence, if $p_t > \frac{\delta}{T}$, the attacker wins with probability at most $\frac{\delta}{T}$. When $p_t \leqslant \delta$, the attacker wins only if the query is allowed, and even then only with probability p_t. We get that in both cases the attacker wins with probability at most $\frac{\delta}{T}$. By the union bound, the probability that the attacker wins any one of the T rounds is at most δ, as desired. □

7.4. Running time

Denoting the running time of Algorithm *Sample* by TIME(*Sample*, ϵ), the running time of Algorithm SAFE is $O(n\frac{c}{\beta}N \cdot \text{TIME}(\textit{Sample}, \epsilon))$ and hence the running time of Algorithm 4 is $O(n\frac{c}{\beta}N\frac{T}{\delta}\log\frac{T}{\delta} \cdot \text{TIME}(\textit{Sample}, \epsilon))$.

We observe that the above algorithm can be modified to handle the case when the number of rounds T is not known a priori or could be potentially unbounded. Suppose the number of rounds is estimated to be within a constant factor of T_0. We allot an error budget of $\delta/2$ for the first T_0 queries, $\delta/4$ for the next T_0 queries, and so on. In other words, we set $\delta/2$ as the error parameter for the first T_0 rounds, $\delta/4$ as the error parameter for the next T_0 rounds, and so on. Then, we get that the attacker wins the first T_0 rounds with probability at most $\frac{\delta}{2}$, the next T_0 rounds with probability at most $\frac{\delta}{4}$, and so on. By the union bound, the probability that the attacker wins any one of the T rounds is at most $\frac{\delta}{2} + \frac{\delta}{4} + \cdots < \delta$.

Remark. We further remark that our simulatable auditing algorithm for sum queries can be extended to any linear combination queries. This is because, as in the case of sum auditing, $(\bigcap_{j=1}^{t}(\sum_{i=1}^{n} q_{ji}x_i = a_j))$ defines a convex constraint where $q_{j1}, \ldots, q_{jn}$ are the coefficients of the linear combination query q_j.

8. Conclusions and future directions

We investigated the fundamental issue that query denials leak information. While existing online auditing algorithms do not explicitly account for query denials, we believe that future research must account for such leakage if privacy is to be ensured. We suggest one natural way to get around the leakage that is inspired by the simulation paradigm in cryptography – where the decision to deny can be equivalently decided by either the attacker or the auditor.

Next we introduced a new definition of privacy. While we believe that this definition overcomes some of the limitations discussed, there is certainly room for future work. The current definition does not ensure that the privacy of a group of individuals or any function of a group of individuals is kept private.

Our sum simulatable auditing algorithm demonstrates that a polynomial-time solution exists. But the sampling algorithms that we invoke are still not practical – although they have been steadily improving over the years. Simulatable sum

queries over Boolean data is an interesting avenue for further work, as is the study of other classes of queries such as the kth ranked element, variance, clusters and combinations of these.

Acknowledgments

We thank the anonymous reviewers for their insightful comments.

Appendix A. Proof of Lemma 7.2

To simplify the analysis, we will assume that Algorithm *Sample* always returns a sample from the true distribution. We will first take into account the error due to this assumption. For $1 \leqslant j \leqslant N$, let G_j be the distribution followed by the output of Algorithm $Sample(\mathcal{D}_{Q,t}, \epsilon)$ in jth iteration of the algorithm. Then, the variation distance between the distributions, $dist(G_j, \mathcal{D}_{Q,t}) \leqslant \epsilon$. Using a standard argument (simple hybrid argument), it follows that, the variation distance between the two product distributions, $dist(G_1 \times G_2 \times \cdots \times G_N, \mathcal{D}_{Q,t}^N) \leqslant N\epsilon = \eta/2$.

We will use the subscript '*real*' to refer to probabilities under the distribution sampled by *Sample* and '*ideal*' to denote the probabilities under the assumption that *Sample* returns a sample from the true distribution. Then, for any event E, $\Pr_{real}(E) \leqslant \Pr_{ideal}(E) + \eta/2$.

We now prove the two parts of the lemma.

Part 1: Using Lemma 6.1, it follows that $\text{Safe}_{\tilde{\lambda},i,I} = 0$ for some $i \in [n]$ and $I \in \mathcal{I}$. Let $p_e = \Pr_{\mathcal{D}_{Q,t}}(x_i \in I)$ denote the posteriori probability. From the definition of $\mathcal{I}$, the a priori probability $\Pr_{\mathcal{D}}(x_i \in I) = 1/\ell$.

Suppose $1 + \tilde{\lambda} < \frac{\Pr_{\mathcal{D}}(x_i \in I | \bigwedge_{j=1}^{t} (\text{sum}(Q_j) = a_j))}{\Pr_{\mathcal{D}}(x_i \in I)} = p_e \ell$.

We use Chernoff bounds to show that $\frac{\ell \cdot N_e}{N} > 1 + \lambda'$ with probability at least $1 - \eta$. Let $\theta_1 = \frac{p_e\ell - (1+\lambda')}{p_e\ell} \geqslant \frac{\tilde{\lambda} - \lambda'}{p_e\ell} = \frac{2\tilde{\lambda}}{3p_e\ell} > 0$. Then,

$$\begin{aligned}
\Pr_{ideal}\left(\frac{\ell \cdot N_e}{N} \leqslant 1 + \lambda'\right) &= \Pr_{ideal}\left(N_e \leqslant Np_e(1-\theta_1)\right) \\
&\leqslant e^{-\frac{Np_e\theta_1^2}{4}} \\
&\leqslant \eta/2
\end{aligned}$$

where the last step is obtained using $N \geqslant \frac{9\ell^2 \ln(2/\eta)}{\tilde{\lambda}^2}$, so that $\frac{Np_e\theta_1^2}{4} \geqslant \frac{9\ell^2 \ln(2/\eta)}{\tilde{\lambda}^2} \cdot \frac{p_e}{4} \cdot (\frac{2\tilde{\lambda}}{3p_e\ell})^2 \geqslant \ln(2/\eta)$. Hence,

$$\begin{aligned}
\Pr_{real}\left(\frac{\ell \cdot N_e}{N} \leqslant 1 + \lambda'\right) &\leqslant \Pr_{ideal}\left(\frac{\ell \cdot N_e}{N} \leqslant 1 + \lambda'\right) + \eta/2 \\
&\leqslant \eta.
\end{aligned}$$

Thus Algorithm Safe returns *false* with probability at least $1 - \eta$.

Now suppose $p_e\ell < \frac{1}{1+\tilde{\lambda}}$. Let $\theta_2 = \frac{1 - p_e\ell(1+\lambda')}{p_e\ell(1+\lambda')} > 0$. Using a similar argument as above, we get

$$\begin{aligned}
\Pr_{real}\left(\frac{\ell \cdot N_e}{N} \geqslant \frac{1}{1+\lambda'}\right) &\leqslant \Pr_{ideal}\left(\frac{\ell \cdot N_e}{N} \geqslant \frac{1}{1+\lambda'}\right) + \eta/2 \\
&= \Pr_{ideal}\left(N_e \geqslant Np_e(1+\theta_2)\right) + \eta/2 \\
&\leqslant e^{-\frac{Np_e\theta_2^2}{4}} + \eta/2 \\
&\leqslant \eta
\end{aligned}$$

where the last step is obtained using $N \geqslant \frac{9\ell^2 \ln(2/\eta)}{\tilde{\lambda}^2} \cdot (1 + \tilde{\lambda}/3)^2(1 + \tilde{\lambda})^2$ and $\theta_2 = \frac{1 - p_e\ell(1+\lambda')}{p_e\ell(1+\lambda')} \geqslant \frac{1 - \frac{1+\lambda'}{1+\tilde{\lambda}}}{p_e\ell(1+\lambda')} = \frac{\tilde{\lambda} - \lambda'}{p_e\ell(1+\lambda')(1+\tilde{\lambda})} = \frac{2\tilde{\lambda}}{3p_e\ell(1+\tilde{\lambda}/3)(1+\tilde{\lambda})}$, so that $\frac{Np_e\theta_2^2}{4} \geqslant \frac{9\ell^2 \ln(2/\eta)}{\tilde{\lambda}^2} \cdot (1 + \tilde{\lambda}/3)^2(1 + \tilde{\lambda})^2 \cdot \frac{p_e}{4} \cdot (\frac{2\tilde{\lambda}}{3p_e\ell(1+\tilde{\lambda}/3)(1+\tilde{\lambda})})^2 \geqslant \ln(2/\eta)$.

Part 2: We actually prove a stronger claim than what is given in the statement of the lemma, by assuming a weaker condition. The stronger claim was not given as part of the lemma so that the two parts of the lemma are symmetric and easier to understand. By definition, $\text{AllSafe}_{\tilde{\lambda}/9,\ell} = 1$ means that the sequence of queries and answers is $\tilde{\lambda}/9$-safe for all entries and all ℓ-significant intervals. In particular, this holds for the ℓ intervals in the partition $\mathcal{I}$. Our stronger claim, which requires only this weaker assumption, is stated and proved below.

Claim 1. *Whenever* $\text{Safe}_{\tilde{\lambda}/9,i,I} = 1$ *for every* $i \in [n]$ *and each of the* ℓ *intervals* I *in the partition* $\mathcal{I}$*, the following statements hold*: (i) $\text{AllSafe}_{(\frac{\tilde{\lambda}}{9} + \frac{16}{9(c-1)}),\beta} = 1$. (ii) *Algorithm* Safe *returns* true *with probability at least* $1 - 2n\ell\eta$.

Proof. The part (i) of the claim follows directly from the proof of Lemma 6.1, by replacing $\tilde{\lambda}$ with $\tilde{\lambda}/9$. This is because the sequence of queries and answers is $((\frac{d+1}{d-1})(1+\tilde{\lambda}/9)-1)$-safe for the interval J (as defined in that proof). In order to see why $\text{AllSafe}_{(\frac{\lambda}{9}+\frac{16}{9(c-1)}),\beta}=1$, note that:

$$\left(\frac{d+1}{d-1}\right)(1+\tilde{\lambda}/9) \leqslant \left(\frac{c+1}{c-1}\right)(1+\tilde{\lambda}/9) = 1+\left(\frac{\lambda}{9}+\frac{16}{9(c-1)}\right).$$

We next prove part (ii) of the claim. By assumption, for any i and $I \in \mathcal{I}$,

$$\frac{1}{1+\tilde{\lambda}/9} \leqslant p_e \ell \leqslant 1+\tilde{\lambda}/9.$$

We will show that for each i and $I \in \mathcal{I}$, Algorithm SAFE returns *false* with probability at most 2η. Using the union bound on all $i \in [n]$ and $I \in \mathcal{I}$ yields the proof.

Let $\theta_3 = \frac{(1+\lambda')-p_e\ell}{p_e\ell} > 0$ and $\theta_4 = \frac{p_e\ell(1+\lambda')-1}{p_e\ell(1+\lambda')} > 0$.

$$\begin{aligned}
\Pr_{real}\left(\frac{\ell \cdot N_e}{N} > 1+\lambda'\right) &\leqslant \Pr_{ideal}\left(\frac{\ell \cdot N_e}{N} > 1+\lambda'\right) + \eta/2 \\
&= \Pr_{ideal}\left(N_e > Np_e(1+\theta_3)\right) + \eta/2 \\
&\leqslant e^{-\frac{Np_e\theta_3^2}{4}} + \eta/2 \\
&\leqslant \eta.
\end{aligned}$$

The last step is obtained using $N \geqslant \frac{9\ell^2\ln(2/\eta)}{\tilde{\lambda}^2}\cdot(1+\tilde{\lambda}/3)^2(3+\tilde{\lambda}/3)^2 \geqslant \frac{81\ell^2\ln(2/\eta)}{\tilde{\lambda}^2}$ and $\theta_3 = \frac{(1+\lambda')-p_e\ell}{p_e\ell} \geqslant \frac{(1+\lambda')-(1+\tilde{\lambda}/9)}{p_e\ell} = \frac{2\tilde{\lambda}}{9p_e\ell}$, so that $\frac{Np_e\theta_3^2}{4} \geqslant \frac{81\ell^2\ln(2/\eta)}{\tilde{\lambda}^2}\cdot\frac{p_e}{4}\cdot(\frac{2\tilde{\lambda}}{9p_e\ell})^2 \geqslant \ln(2/\eta)$.

Similarly,

$$\begin{aligned}
\Pr_{real}\left(\frac{\ell \cdot N_e}{N} < \frac{1}{1+\lambda'}\right) &\leqslant \Pr_{ideal}\left(\frac{\ell \cdot N_e}{N} < \frac{1}{1+\lambda'}\right) + \eta/2 \\
&= \Pr_{ideal}\left(N_e < Np_e(1-\theta_4)\right) + \eta/2 \\
&\leqslant e^{-\frac{Np_e\theta_4^2}{4}} + \eta/2 \\
&\leqslant \eta.
\end{aligned}$$

The last step is obtained using $N \geqslant \frac{9\ell^2\ln(2/\eta)}{\tilde{\lambda}^2}\cdot(1+\tilde{\lambda}/3)^2(3+\tilde{\lambda}/3)^2$ and $\theta_4 = \frac{p_e\ell(1+\lambda')-1}{p_e\ell(1+\lambda')} \geqslant \frac{\frac{1+\lambda'}{1+\tilde{\lambda}/9}-1}{p_e\ell(1+\lambda')} = \frac{\lambda'-\tilde{\lambda}/9}{p_e\ell(1+\lambda')(1+\tilde{\lambda}/9)} = \frac{2\tilde{\lambda}}{3p_e\ell(1+\tilde{\lambda}/3)(3+\tilde{\lambda}/3)}$, so that $\frac{Np_e\theta_4^2}{4} \geqslant \frac{9\ell^2\ln(2/\eta)}{\tilde{\lambda}^2}\cdot(1+\tilde{\lambda}/3)^2(3+\tilde{\lambda}/3)^2\cdot\frac{p_e}{4}\cdot(\frac{2\tilde{\lambda}}{3p_e\ell(1+\tilde{\lambda}/3)(3+\tilde{\lambda}/3)})^2 \geqslant \ln(2/\eta)$. □

References

[1] N. Adam, J. Wortmann, Security control methods for statistical databases: A comparative study, ACM Computing Surveys 21 (4) (1989) 515–556.
[2] M. An, Log-concave probability distributions: Theory and statistical testing, Technical Report 9611002, Economics Working Paper Archive at WUSTL, 1996, available at http://ideas.repec.org/p/wpa/wuwpga/9611002.html.
[3] L. Beck, A security mechanism for statistical database, ACM Transactions on Database Systems 5 (3) (1980) 316–338.
[4] F. Chin, Security problems on inference control for SUM, MAX, and MIN queries, Journal of the ACM 33 (3) (1986) 451–464.
[5] F. Chin, G. Ozsoyoglu, Auditing for secure statistical databases, in: Proceedings of the ACM '81 Conference, 1981, pp. 53–59.
[6] F. Chin, G. Ozsoyoglu, Statistical database design, ACM Transactions on Database Systems 6 (1) (1981) 113–139.
[7] T. Dalenius, Towards a methodology for statistical disclosure control, Statistisk Tidskrift 15 (1977) 429–444.
[8] I. Dinur, K. Nissim, Revealing information while preserving privacy, in: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2003, pp. 202–210.
[9] D. Dobkin, A. Jones, R. Lipton, Secure databases: protection against user influence, ACM Transactions on Database Systems 4 (1) (1979) 97–106.
[10] C. Dwork, Differential privacy, in: ICALP, 2006, pp. 1–12.
[11] C. Dwork, F. McSherry, K. Nissim, A. Smith, Calibrating noise to sensitivity in private data analysis, in: Proceedings of the 3rd Theory of Cryptography Conference, 2006, pp. 265–284.
[12] C. Dwork, K. Nissim, Privacy-preserving datamining on vertically partitioned databases, in: Advances in Cryptology: Proceedings of Crypto, 2004, pp. 528–544.
[13] A. Evfimievski, R. Fagin, D. Woodruff, Epistemic privacy, in: PODS, ACM, 2008, pp. 171–180.
[14] A. Evfimievski, J. Gehrke, R. Srikant, Limiting privacy breaches in privacy preserving data mining, in: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2003, pp. 211–222.
[15] S. Goldwasser, S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 365–377.

[16] J. Kam, J. Ullman, A model of statistical databases and their security, ACM Transactions on Database Systems 2 (1) (1977) 1–10.
[17] K. Kenthapadi, N. Mishra, K. Nissim, Simulatable auditing, in: Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2005, pp. 118–127.
[18] J. Kleinberg, C. Papadimitriou, P. Raghavan, Auditing boolean attributes, Journal of Computer and System Sciences 66 (1) (2003) 244–253.
[19] Y. Li, L. Wang, X. Wang, S. Jajodia, Auditing interval-based inference, in: Proceedings of the 14th International Conference on Advanced Information Systems Engineering, 2002, pp. 553–567.
[20] L. Lovasz, S. Vempala, Logconcave functions: Geometry and efficient sampling algorithms, in: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 640–649.
[21] G. Miklau, D. Suciu, A formal analysis of information disclosure in data exchange, Journal of Computer and System Sciences 73 (3) (2007) 507–534.
[22] S. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, R. Motwani, Towards robustness in query auditing, in: Proceedings of the 32nd International Conference on Very Large Databases, 2006.
[23] S. Nabar, N. Mishra, Releasing private contingency tables, Journal of Privacy and Confidentiality 2 (1) (2011) 109–140.
[24] A. Prekova, Stochastic Programming, Akadémiai Kiadó/Kluwer, Budapest/Dordrecht, 1995.
[25] S. Reiss, Security in databases: A combinatorial study, Journal of the ACM 26 (1) (1979) 45–57.