Randomized Generalization for Aggregate Suppression Over Hidden Web Databases

Xin Jin, Nan Zhang [*]
George Washington University
{xjin, nzhang10}@gwu.edu

Aditya Mone, Gautam Das [†]
University of Texas at Arlington
{aditya.mone, gdas}@uta.edu

ABSTRACT

Many web databases are hidden behind restrictive form-like interfaces which allow users to execute search queries over the underlying hidden database. While it is important to support such search queries, many hidden database owners also want to maintain a certain level of privacy for aggregate information over their databases, for reasons including business secrets and homeland security. Existing work on aggregate suppression thwarts the uniform random sampling of a hidden database, but cannot address recently proposed attacks which accurately estimate SUM and COUNT queries without the need to first draw a uniform random sample. In this paper, we consider the problem of suppressing SUM and COUNT queries over a hidden database. In particular, we develop randomized generalization, a novel technique which provides rigid aggregate-suppression guarantee while maintaining the utility of individual search queries. We present theoretical analysis and extensive experiments to illustrate the effectiveness of our approach.

1. INTRODUCTION

1.1 Problem Definition and Motivation

Hidden web databases provide form-like search interfaces that allow users to issue search queries by specifying desired attribute values of the sought-after tuple(s), and the system responds by returning a few (e.g., top-k) tuples that satisfy the selection conditions, sorted by a suitable scoring function. Here are two examples:

Example 1: *Many online auto dealers (e.g.,Yahoo! Autos) offer a web form that lets a user choose values for attributes such as ZIP code, car model, mileage, age, price, etc. The top-k answers, sorted according to a scoring function such as distance or price, are presented to the user, where k is a small constant such as 50.*

Example 2: *Almost all major airline companies (e.g., American Airlines) provide form-like interfaces for a user to search for flights.*

American Airlines' website, for example, lets a user choose values for six attributes: departure city, arrival city, number of stops, departure date, cabin, and carrier. The top-50 answers are returned.

Note that while a hidden web database supports individual search queries, it does not allow aggregate queries, e.g., the total number of used hybrid cars in an online dealer's database, to be executed through the interface[1]. Because of the top-k constraint, such aggregates cannot be inferred from a search query answer either. Thus, it was traditionally believed that aggregate information is shielded from external applications by the restrictive web interface and cannot be efficiently retrieved without first crawling a large number of tuples from the hidden database. Nonetheless, our recent work (e.g., [7, 16]) shows that, through a carefully designed workload of search queries, one can indeed leverage the public web interface to accurately estimate aggregate information over a hidden database.

A recent paper introduced the novel challenge of *privacy preservation of aggregates* in such hidden databases [8]. This paper observed that while it is important for owners of hidden databases to support search queries outlined in the above examples, many of them also want to maintain a certain level of privacy for aggregates over their hidden databases (e.g., as business secrets). In Example 1, the auto dealer would not be willing to disclose aggregate information that enables competitors to infer its inventory, e.g., that a certain popular car is in short supply at the dealership around the country. If the competitors were able to infer such aggregate information, then they would be able to take advantage of the low inventory by a multitude of tactics (e.g., stock that vehicle, make appropriate adjustments to price). Likewise, in Example 2, airline companies, for both security and business reasons, do not want to make public information that allows terrorists to predict which flight on which date is more likely to be relatively empty. In recent hijackings such as 9/11 and Russian aircraft bombing, terrorists' tactics are believed to be hijacking a relatively empty flight so there would be less resistance from the passengers.

This novel notion of aggregates suppression is in sharp contrast to traditional privacy scenarios, where individual tuple needs to be protected but aggregate information that does not lead to inference to an individual's information is considered acceptable disclosure. As argued in [8], traditional tuple-wise privacy preservation techniques such as encryption [1], data perturbation [2] and generalization methods [21] cannot be used for aggregates suppression as obfuscating individual data tuples is not an option since these tuples must be made visible to normal search users.

The defense mechanism COUNTER-SAMPLER proposed in [8] represented an important first step towards addressing this important problem. However, it suffers from two crucial shortcomings:

[1] Note that even interfaces which choose to display certain aggregates (e.g., COUNT) often only offer coarse estimations (e.g., the COUNT returned by Google is notoriously inaccurate).

1. The main objective of COUNTER-SAMPLER is to prevent a *uniform random sampling attack*, i.e., to prevent an adversary from drawing a uniform random sample of the database via executing search queries. However, a uniform random sample is not necessary for inferring important aggregates such as COUNT and SUM. As shown in [7], a recently proposed technique UNBIASED-AGG can infer such aggregates *without the need to draw a uniform random sample*, thereby bypassing the COUNTER-SAMPLER defense mechanism.

2. COUNTER-SAMPLER involves insertion of "dummy" tuples into the database, whose purpose is to delay the execution of any algorithm attempting to draw a uniform random sample. Dummy tuples are associated with specific CAPTCHA images to enable normal search users to identify and discard them from the query results. This increased burden on users has a detrimental effect on the system's usability. Secondly, the construction of semantically meaningful dummy tuples is a nontrivial task (otherwise automated adversaries can easily detect them) which has not been adequately addressed in [8]. Finally, the use of CAPTCHA images is not foolproof – as has been recently shown in the security community [12], CAPTCHA can be broken given enough computing power.

Therefore, the primary objective of our present paper is to develop privacy preservation techniques against adversarial attacks - e.g., [7,16] - for inferring sensitive SUM and COUNT aggregates[2]. More importantly, as a design principle, we *only consider techniques that are only allowed to return actual tuples of the database in response to search queries*. This precludes techniques such as the use of dummy tuples and CAPTCHA images, and their associated disadvantages of reduced utility and vulnerability.

1.2 Highlights of Proposed Approach

We propose *randomized generalization*, a novel framework for suppressing (SUM, COUNT)-aggregates from adversaries while also preserving utility of the database for normal search users. The philosophy behind randomized generalization is simple. Consider a Washington DC-based user searching for a car in an online dealer's database who specifies a ZIP code such as 20052 in the search query. After randomized generalization, the query answer includes not only all cars in 20052, but also a few other randomly selected cars having Washington DC ZIP codes (i.e., close to 20052), with the location of all returned cars displayed as Washington, DC. More generally, randomized generalization achieves the suppression by generalizing the value of one or a few attributes to also include other randomly selected "close enough" values.

From the view of an attacker, the randomized generalization process brings significant challenges to aggregate estimation - e.g., as we shall show in §3.2, in order to accurately estimate the total number of tuples in the database, the attacker has to learn for each returned tuple the number of different ZIP codes which return it. With a careful design of randomized generalization, this identification process requires the attacker to issue a large number of search queries. On the other hand, from the view of a normal search user, randomized generalization brings a change in the semantics of query answers: as a matter of fact, the returned tuples are those that *fuzzy-match* the query. Fuzzy matching is a popular technique already used in query answering in many real-world hidden databases (e.g., Yahoo! Autos), and is often considered as

an enhancement of the user's search experience, rather than as a degradation of utility.

For our randomized generalization framework to be applicable, we assume that the query interface has a *mandatory* attribute, and each search query has to specify a value for at least this attribute, in addition to any other attributes. Mandatory attributes are quite common for real-world interfaces (e.g., ZIP code/City is mandatory for cars.com new/used car search), in order to limit a search user's attention to the tuples he/she is truly interested in. Next, we assume that all possible values of the mandatory attribute have been partitioned into disjoint subsets, which we refer to as *generalizable value sets* (GVS), so that each subset includes values that are considered "close enough" to each other from a search user's perspective[3]. Our technique only generalizes the mandatory attribute value to a random subset of its corresponding GVS, to ensure the interest of a search user on the returned (fuzzy-matched) tuples.

We stress that in our framework that generalization of a value to a *random subset* of its GVS is critical. This is in sharp contrast to *deterministic* generalization techniques extensively used in traditional privacy-preserving data publishing techniques (e.g., [21]) for suppressing access to individual tuples. We found that if one directly applies the traditional (deterministic) generalization technique, an adversary with knowledge of the generalization scheme (i.e., GVS partitions) can easily nullify generalization and reconstruct the original aggregates. Thus, we introduce uncertainty into the process by performing generalization in a randomized fashion - e.g., even when all ZIP codes in Washington DC are in the same GVS, we do not return all cars in the city when a user queries for ZIP = 20052. Instead, we return all tuples with an exact match to 20052, as well as a few other tuples randomly selected from those which match the GVS (i.e., Washington DC).

Exactly how to randomly select the fuzzy-matching tuples is a subtle issue which we shall address in the paper. In particular, we start with a simple design, RG-SIMPLE, which works over a limited type of hidden databases. Recognizing the limit, we enhance the design of randomized generalization and develop RG-SUPPRESS, a general-purpose algorithm which guarantees the suppression of COUNT and SUM aggregates unless an adversary issues an extremely large number of queries. In particular, in the design of RG-SUPPRESS, we discovered an intriguing connection between the design of randomized generalization and a well-known (and difficult) Prouhet-Tarry-Escott (P-T-E) problem in number theory, and found a P-T-E solution which suits our needs. In the paper, we shall derive theoretical aggregate-suppression guarantees provided by RG-SIMPLE and RG-SUPPRESS, and also verify their effectiveness over two real-world hidden databases, NSF Fastlane Award Search and Yahoo! Auto.

1.3 Summary of Contributions

In summary, our paper makes the following contributions:

- We consider the novel problem of suppressing COUNT and SUM aggregates over a hidden database from being revealed.

- We propose *randomized generalization*, a novel framework for suppressing aggregates from adversaries that also preserves the utility of database for normal search users.

- Based on the randomized generalization framework, we develop RG-SIMPLE, a specific approach to suppress COUNT and SUM aggregates no matter how many queries an adversary issues, but only works for a limited type of databases.

---

[2]It is important to note that not all SUM and COUNT queries can be protected – e.g., an extremely narrow one which only returns only one tuple may have to be disclosed for maintaining the utility of the database to normal search users. The sensitive SUM and COUNT queries are relatively broad queries which match a large number of tuples – e.g., number of hybrid vehicles in a used car database.

[3]Clearly, the assumption of existing GVS partitions for a mandatory attribute needs justification. This issue is discussed in detail in Appendix F of the paper, for a variety of real-world attribute types – such as numeric attributes, categorical attributes with concept hierarchies, attributes with distance measures between values, etc.

- We also develop RG-SUPPRESS, a more robust and general-purpose approach which effectively suppresses aggregates unless an adversary issues an extremely large number of queries. Interestingly, the design of RG-SUPPRESS is based on connections with the well-known (and difficult) Prouhet-Tarry-Escott problem in number theory.

- We conduct a comprehensive set of experiments to validate the effectiveness of our proposed aggregate suppression approaches.

The rest of this paper is organized as follows. We discuss preliminaries in §2. In §3, we introduce the basic idea of randomized generalization and develop RG-SIMPLE. We enhance the design to develop RG-SUPPRESS in §4. §5 presents the experimental results, followed by related work in §6 and conclusions in §7.

2. PRELIMINARIES

2.1 Model of Hidden Databases

Consider a hidden database table D with n tuples $t_1, \ldots, t_n$. Let there be m attributes $A_1, \ldots, A_m$ specifiable through the input interface, and Θ_i be the domain (i.e., set of all possible values) of A_i. Without loss of generality, let A_1 be the mandatory attribute which must be specified through the interface[4]. We use $t_i[A_j]$ to denote the value of A_j for t_i. Consider a prototypical interface which allows a user to query D by specifying values for a subset of attributes - i.e., to issue query q of the form:

SELECT * FROM D WHERE $A_1 = v_1$ & $A_{i_1} = v_{i_1}$ & $\cdots$,

where $v_j \in \Theta_j$. Let $Sel(q)$ be the set of tuples in D which satisfy q. Since the interface is restricted to return up to k tuples, a overly broad query (i.e., $|Sel(q)| > k$) will *overflow* and return only the top-k tuples in $Sel(q)$ selected according to a scoring function. At the other extreme, if the query is too specific to return any tuple, we say that an *underflow* occurs. If there is neither overflow nor underflow (i.e., $|Sel(q)| \in [1, k]$), then $Sel(q)$ will be returned in its entirety and we have a *valid* query result.

2.2 Objective of Aggregate Suppression

Our objective is to suppress aggregates of the form Q_A : SELECT AGGR(*) FROM D WHERE *selection_condition*, where AGGR is an aggregate function. In this paper, we focus on COUNT and SUM as the aggregate function. Let $Res(Q_A)$ be the answer to Q_A. As discussed in §1, due to privacy concerns the owner of a hidden database may consider certain Q_A to be sensitive and would not willingly disclose their results. Throughout the paper, we use Q_A: SELECT COUNT(*) FROM D as a running example, while showing that a simple extension exists to other COUNT and SUM aggregates with or without selection conditions.

To quantify the degree of aggregate disclosure, we consider a (ϵ, δ, c)-privacy game similar in spirit to the privacy game notions in [17] and [8]. For a sensitive Q_A, consider three steps:

1. The owner applies its aggregate-suppression technique.

2. The adversary issues at most c search queries and analyzes their answers to try and estimate $Res(Q_A)$.

3. The adversary wins if $\exists x$ such that the adversary has confidence $> \delta$ that $Res(Q_A) \in [x, x + \epsilon]$. Otherwise, the defender wins.

Based on the (ϵ, δ, c)-game notion, we define the aggregate suppression guarantee for a hidden database as follows:

DEFINITION 1. *We say that an aggregate-suppression technique achieves an (ϵ, δ, c, p)-guarantee if and only if for any sensitive ag-*

gregate Q_A *and any adversary P_A,*

$$\Pr\{P_A \text{ wins } (\epsilon, \delta, c)\text{-privacy game for } Q_A\} \leq p. \quad (1)$$

The probability is taken over the (possible) randomness in both the aggregate-suppression scheme and the attacking strategy. One can see that the greater ϵ is or the smaller δ, c and p are, the more protection a (ϵ, δ, c, p)-privacy-guarantee has to provide on sensitive aggregates.

2.3 Objective of Utility Preservation

Since our method for aggregate suppression is to change a query answer to include not only the tuples that are originally returned but also a small number of other tuples in the database, we have an important objective of minimizing the inconvenience such changes cause to the end users. We would like to note that the inclusion of more tuples in a query answer (than the exact-matching ones) is not necessarily inconvenient by itself - after all, many real-world hidden databases, such as realtors.com, already implement fuzzy matching to include tuples that are close to, but do not exactly match, the selection conditions in the query. What may cause loss of utility here for end users is when the tuples returned are not close to the query selection conditions - e.g., when a user searches for houses in 20001, a ZIP code in Washington DC, but instead gets a house in Texas returned in the query answer.

To capture this notion of utility preservation, we define sets of *generalizable values* - i.e., attribute values that are considered *close* to each other - for the mandatory attribute[5] as follows. Consider attribute A_1 and its domain Θ_1. We define the *generalizable value sets* (GVS) for A_1 as mutually exclusive subsets $U_1, \ldots, U_h \subseteq \Theta_1$ which satisfies $U_1 \cup \cdots \cup U_h = \Theta_1$, such that for any U_j ($j \in [1, h]$), a tuple t with $t[A_1] \in U_j$ can be returned in a query with selection condition $A_1 = v$, where v is an arbitrary value in U_j (assuming that all other conjunctive selection conditions are satisfied by t), without causing inconvenience to the end user.

Constructing GVS can be straightforward for certain attributes - e.g., for ZIP code, we can include in each GVS all ZIP codes in a given city. The construction, however, can also be nontrivial for other attributes - e.g., numerical attributes such as price (e.g., are 15,000 and 19,000 close enough?) or categorical attributes with no known concept hierarchy (e.g., car model - are Ford Focus and Toyota Corolla close to each other?). We discuss the construction of GVS from a variety of auxiliary information - e.g., concept hierarchy, pairwise distance, and scoring function used by the hidden database - in Appendix F. For the other part of the paper, we assume the existence of GVS for the mandatory attribute, and design our aggregate suppression techniques based on these GVS. A note here is that GVS are also known by the adversary.

2.4 Running Example

To illustrate our idea of randomized generalization, we consider a simple running example where the hidden database has only one attribute, ZIP code, which is a mandatory attribute and has 42,000 possible values (close to the real number of ZIP codes in the US). The GVS for ZIP are designed such that each set consists of all ZIP codes in a city.

One can see that with this simple example, there are only 42,001 possible queries that an adversary may issue, the additional one being SELECT * FROM D. We consider the value of k (as in top-k interface) to be large enough such that none of the 42,000 queries (other than SELECT *) overflows. This way, each tuple in the database is returned by one query of the form SELECT * FROM D WHERE ZIP = v_i, in which v_i is a possible value of ZIP.

---

[4] We shall extend the notion of mandatory attribute to a combination of multiple attributes in Appendix F.

[5] Recall from §1 that each search query has to specify a value for at least the mandatory attribute, in addition to possibly other attributes.

2.5 Existing Aggregate Estimation Attack

We now describe the state-of-the-art aggregate estimation attack (over a hidden database), UNBIASED-AGG [7], in the context of our running example. The simplicity of our running example enables the following simple description of the attack which is sufficient for our purpose of illustrating the idea of randomized generalization. We refer readers to [7] for further details of the attack.

For the running example, consider a bipartite graph depicted in Figure 1 (a) where there are 42,000 left-hand-side (LHS) vertices, each corresponding to a query with a different ZIP code. There are $n-k$ right-hand-side (RHS) vertices, each corresponding to a tuple not returned by SELECT * FROM D. An edge connects a query with a tuple if and only if the query returns the tuple.

Since our objective in the running example is to suppress the result of COUNT(*), i.e., n, we consider an adversary which aims to estimate the number of RHS vertices - i.e., $n-k$. A key observation behind the UNBIASED-AGG attack is that the number of RHS vertices is the same as the number of edges in the bipartite graph, because all queries are mutually exclusive and therefore each tuple is returned by exactly one query. Based on this observation, the adversary can simply select a query q_i uniformly at random from the LHS, and then estimate

$$n - k \approx |q_i| \cdot 42000, \qquad (2)$$

where $|q_i|$ is the number of tuples returned by (i.e., degree of) q_i, and 42,000 is the number of LHS nodes. For example, in Figure 1 (a), an adversary which issues SELECT * FROM D WHERE ZIP = 20052 would generate an estimation of $42,000 \times 2 = 84,000$. One can see that the estimation in (2) is completely unbiased - i.e., its expected value is equal to $n-k$. The adversary can further improve estimation accuracy (i.e., reduce estimation variance) by repeating the process multiple times and taking the average estimation.

We would like to note that while we use this attack multiple times in the paper as an example to illustrate our idea of randomized generalization, the effectiveness of our proposed algorithms, RG-SIMPLE and RG-SUPPRESS, are by no means restricted to this attack, as indicated in the proof of the generic aggregate-suppression guarantees offered by these algorithms.
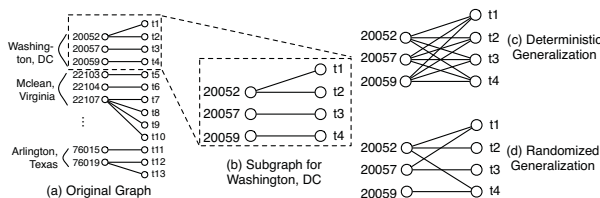

Figure 1: Examples of Bipartite Graphs.

3. RANDOMIZED GENERALIZATION

In this section, we introduce randomized generalization, our main idea for suppressing aggregates while preserving the utility of search queries. The input to randomized generalization is the generalizable value sets (GVS) of the mandatory attribute - the construction of which is discussed in Appendix F. We first discuss the problem of deterministic generalization, then describe our idea of adding randomness to the generalization process, and finally develop RG-SIMPLE and derive its aggregate-suppression guarantees.

3.1 Problem of Deterministic Generalization

A straightforward method to hide COUNT(*) is to (deterministically) *generalize* each tuple in the database based on the input GVS

- in the running example, a tuple with ZIP code 20052 would have its ZIP code attribute generalized to (all zip codes in) Washington, DC. As such, a search query q_i: SELECT * FROM D WHERE ZIP = 20052 will be answered with all[6] tuples in Washington, DC, because all of them have the same generalized ZIP as 20052. From the bipartite-graph view in Figure 1(c), this generalization process inserts a large number of edges into the graph. Note that the ZIP code of each returned tuple must be *displayed* as the city name, because otherwise an adversary can simply filter out all tuples with ZIP $\neq$ 20052, and thereby nullify generalization.

After (deterministic) generalization, a direct application of the UNBIASED-AGG attack for estimating COUNT(*) leads to an estimation much larger than the real value. Consider estimations of the total number of tuples in Washington, DC over Figure 1 (b) and (c). When SELECT * FROM D WHERE ZIP = 20052 is issued, the estimation before deterministic generalization is $2 \times 3 = 6$ (assuming that 20052, 20057 and 20059 are the only ZIP codes in Washington, DC), while the estimation afterwards is $4 \times 3 = 12$. This is because, as shown in §2.5, UNBIASED-AGG estimates COUNT(*) based on the total number of edges in the bipartite graph. While the number of edges and RHS tuples are the same before generalization, the number of edges becomes much larger afterwards, leading to the higher estimations.

Nonetheless, a slight revision of UNBIASED-AGG can easily defeat deterministic generalization. Specifically, in the above example, instead of producing an estimation of $|q_i| \cdot 3$, the adversary instead computes $|q_i| \cdot 3/f(v_i)$, where $f(v_i)$ is the number of ZIP codes in the city to which v_i belongs, because the adversary knows that the total number of edges in the subgraph is exactly $f(v_i)$ times the number of tuples. For example, SELECT * FROM D WHERE ZIP = 20052 over Figure 1 (c) now produces an estimation $4 \cdot 3/3 = 4$ - the exact COUNT of RHS tuples, with no error at all. As such, the aggregate COUNT(*) is fully disclosed.

3.2 Basic Idea of Randomized Generalization

There are two critical observations from the failure of deterministic generalization: (1) The generalization of ZIP code creates a mismatch between the number of edges (observable by the adversary through issuing queries) and the number of RHS tuples, which could potentially mislead the adversary; (2) Nonetheless, if the generalization is done deterministically, then the adversary can leverage knowledge of GVS to infer the degree of RHS tuples and thereby remove the mismatch. Thus, the key idea of randomized generalization is to create the mismatch in a randomized fashion, such that an adversary no longer has the knowledge of RHS degree - i.e., the ability to remove the mismatch.

In particular, for each tuple t, we define its *random-generalized (RG) value set* $V_{RG}(t)$ as a randomly generated subset of the GVS corresponding to t. How the random generation should be performed - in particular, what should be the distribution of the size of $V_{RG}(t)$ - is a subtle yet critical issue for aggregate suppression which we shall address in §3.3 and §4.

Now we use an example to illustrate how $V_{RG}(t)$ is used. Consider tuple t_1 in Figure 1 (c) with ZIP = 20052. If $V_{RG}(t) = \{20052, 20057\}$, then t_1 will be returned by both of the following queries, as shown in Figure 1 (c).

SELECT * FROM D WHERE ZIP = 20052, and
SELECT * FROM D WHERE ZIP = 20057.

But in both query answers, the ZIP code of t_1 will still be displayed as Washington, DC (i.e., its GVS). The key rationale here is that since $V_{RG}(t)$ is a random subset of the GVS, the adversary has no knowledge of the number of ZIP codes in $V_{RG}(t)$ - i.e., the

---

[6]Let us consider a worst-case scenario for aggregate suppression by assuming that k is large enough so no overflow occurs even after generalization.

RHS degree of t_1, and therefore cannot carry out the revision to UNBIASED-AGG discussed in §3.1. In the following subsections, we shall consider a simple design of $V_{\mathrm{RG}}(t)$ and prove the suppression guarantees it provides.

3.3 Algorithm RG-SIMPLE

To illustrate the effectiveness of randomized generalization, we start with a simplistic design of RG-SIMPLE which takes as input an *obfuscation factor* γ ($\gamma > 1$). The larger γ is, the more stringent a guarantee RG-SIMPLE provides on aggregate suppression. In general, γ is much smaller than the size of (the smallest) GVS, with reasons explained in §4.3. Note that to achieve security-by-design, we consider the worst-case scenario where γ is also known by the adversary.

Based on γ, RG-SIMPLE first draws a real-valued *match-factor* μ uniformly at random from $[1, \gamma]$. Then, for each tuple $t \in \mathsf{D}$, we design its RG value set $V_{\mathrm{RG}}(t)$ with one of the following two options: One is to not generalize at all (i.e., $V_{\mathrm{RG}}(t) = t[\mathrm{ZIP}]$). This option is chosen with probability $(\gamma - \mu)/(\gamma - 1)$. Otherwise, we select γ values into $V_{\mathrm{RG}}(t)$ as follows. First, $t[\mathrm{ZIP}]$ is always included. Then, we select a tuple t' uniformly at random from all tuples with ZIP in the GVS of $t[\mathrm{ZIP}]$. If $t'[\mathrm{ZIP}]$ is not in $V_{\mathrm{RG}}(t)$ yet, we add it to $V_{\mathrm{RG}}(t)$, and repeat this process until $V_{\mathrm{RG}}(t)$ contains γ values. Appendix A depicts the pseudo-code of RG-SIMPLE.

Intuitively, RG-SIMPLE aims to "emulate" a database of size μ times as large. To see this, consider using UNBIASED-AGG to estimate COUNT(*) after RG-SIMPLE is applied. The expected value UNBIASED-AGG generates for the COUNT of RHS nodes is $(n - k) \cdot \mu$ instead of the real value $n - k$. Of course, since an attacker knows RG-SIMPLE has been applied, it can infer that the real database may not be as large. Nonetheless, a key obstacle for the attacker is that it has no knowledge of the (random) match factor μ, and therefore cannot make a distinction between an original RHS COUNT of $(n - k) \cdot \mu/\gamma$, $(n - k) \cdot \mu$, or any value in between, because for any of such values there exists a value of μ which leads to an emulation of $(n - k) \cdot \mu$.

One can see that the suppression of COUNT(*) also extends to other COUNT and SUM queries with or without selection conditions, because by emulating a database of size μ times as large, RG-SIMPLE simultaneously "enlarges" all other COUNT and SUM aggregates for the same (expected) ratio. We formally derive the suppression guarantee provided by RG-SIMPLE as follows.

3.4 Privacy Guarantees of RG-SIMPLE

In this subsection, we first prove that RG-SIMPLE achieves an effective privacy guarantee for the running example *no matter* how many queries an adversary issues. Then, we explain why the guarantee does not extend to generic databases - a motivation for our study of enhancing randomized generalization in the next section.

THEOREM 3.1. *For a database where non-mandatory attributes $A_2, \ldots, A_m$ in combination does not form a (super) key, RG-SIMPLE achieves $\langle \epsilon, \delta, \infty, p \rangle$-guarantee for a COUNT or SUM aggregate $Res(Q_A)$ if*

$$\epsilon \leq ((\gamma - 1) \cdot p + 1) \cdot Res(Q_A) \cdot \left(1 - \frac{1}{\gamma}\right) \cdot \delta, \qquad (3)$$

and the database size n is sufficiently large.

Please refer to Appendix B for the proof. A sample guarantee proved by the theorem is $\langle Res(Q_A) \cdot (\gamma^2 - 1)/(4\gamma), 50\%, \infty, 50\% \rangle$, meaning that no adversary can have a 50% confidence interval of width $Res(Q_A) \cdot (\gamma^2 - 1)/(4\gamma)$ or shorter - e.g., $Res(Q_A) \cdot 6/5$ when $\gamma = 5$ - with probability more than 50%. One can see that the larger γ is, the better protection RG-SIMPLE provides.

The theorem has a key limitation: all non-mandatory attributes cannot form a (super-)key of the table. While some hidden databases satisfy this constraint - e.g., a case status search website[7] which requires the only candidate key, case number, to be included in any query - most databases do not. In the following, we first explain the reason behind this limitation, and then derive a weaker guarantee provided by RG-SIMPLE for databases which do not satisfy it.

For Generic Databases: We now explain why Theorem 3.1 does not hold for many real-world hidden databases with a key attribute that is not often specified in a query, e.g., VIN number for Yahoo! Autos or MLS ID for realtors.com. For these databases, an adversary may learn certain information about μ (i.e., the match-factor used by RG-SIMPLE) from certain query answers. For example, if the adversary finds that two tuples with the same key are returned in answers to two different queries (with different ZIP specified within), then the adversary can immediately infer that $\mu > 1$. Then, based on the number of "duplicate" tuples it sees, the adversary can further refine its estimation of μ, and eventually reaches an accurate estimation of the sensitive aggregate.

In particular, since the same μ is used for the randomized generalization of all tuples, an adversary should start by issuing queries from the smallest GVS in order to quickly find duplicates in the query answers. Let $u_{\min}$ be the number of values in the smallest GVS. Due to the birthday paradox [19], an adversary only needs to issue an expected number of $\Omega(\sqrt{u_{\min}})$ queries in order to find a duplicate (and to infer $\mu > 1$). The privacy guarantee provided by RG-SIMPLE therefore becomes much weaker, as shown in the following theorem.

COROLLARY 3.1.1. *For any database with a sufficiently large size n, RG-SIMPLE achieves $\langle \epsilon, \delta, \sqrt{u_{\min}}, p \rangle$ privacy guarantee for a COUNT or SUM aggregate $Res(Q_A)$ if*

$$\epsilon \leq ((\gamma - 1) \cdot p + 1) \cdot Res(Q_A) \cdot \left(1 - \frac{1}{\gamma}\right) \cdot \delta. \qquad (4)$$

where $u_{\min}$ is the size of the smallest GVS.

For our running example, even a ZIP-heavy city like Washington, DC only has 564 ZIP codes. As such, a query cost of $\sqrt{u_{\min}}$ is too small to deter a real-world adversary. In the next section, we shall describe an enhanced design of randomized generalization to significantly increase this adversarial query cost.

4. RG-SUPPRESS

We now focus on enhancing randomized generalization to achieve aggregate suppression when a super-key exists outside the mandatory attribute. Our idea of enhancement is two-fold: One is to improve the design of randomized-generalized value set $V_{\mathrm{RG}}(t)$ for a given GVS, in order to maintain suppression even when the attacker finds the same key from two different queries. The other is to differentiate the design of $V_{\mathrm{RG}}(t)$ for different GVS, in order to ensure that the queries an attacker issues for one GVS cannot be used to predict the enlargement ratio over another GVS. In the following, we shall first describe the two ideas respectively, and then combine them to develop RG-SUPPRESS, our general-purpose aggregate suppression algorithm for hidden databases.

4.1 Enhanced Design for a Single GVS

In this subsection, we focus on the design of V_{RG} for (each tuple in) a single GVS, such as a city for the running example. To illustrate the main idea of our design, we start with the following simple problem, followed by an extension at the end of this subsection:

---

[7]e.g., *https://egov.uscis.gov/cris/Dashboard.do*

Consider two databases D_1 and D_2, with n and $\gamma \cdot n$ tuples, respectively, where γ is the same obfuscation factor as in RG-SIMPLE. Let there be two attributes, ZIP code and a primary key (e.g., VIN number), with all tuples in the two databases having ZIP code in Washington, DC. Our objective is to design V_{RG} for both databases such that they are indistinguishable from the view of an adversary unless it issues a large number of queries. In particular, recall that with RG-SIMPLE, an adversary can distinguish D_1 from D_2 once it retrieves the same tuple in two different query answers. We aim to make the distinction more difficult - in particular, to ensure that an adversary cannot distinguish D_1 from D_2 until finding the same tuple from at least h ($h > 2$) different query answers. Note that when h is small, even a small increase on h leads to a significant increase on the adversarial query cost.

As is in RG-SIMPLE, the critical problem here is how to determine $|V_{\text{RG}}(t)|$. Once $|V_{\text{RG}}(t)|$ has been determined, the actual values in $V_{\text{RG}}(t)$ can be (randomly) selected in the same way as RG-SIMPLE. The following theorem reduces the design of $|V_{\text{RG}}(t)|$ to a classical, yet notoriously difficult, number theory problem called Prouhet-Tarry-Escott (P-T-E) [14] shown in (5).

THEOREM 4.1. *A sufficient and necessary condition for D_1 and D_2 to remain indistinguishable until an adversary receives the same tuple in at least h different query answers is*

$$\sum_{t \in D_1} |V_{\text{RG}}(t)|^i = \sum_{t \in D_2} |V_{\text{RG}}(t)|^i. \quad (5)$$

for all $i = 1, \ldots, h - 1$.

Please refer to Appendix C for the proof of this theorem and a detailed discussion of the connection with the P-T-E problem. For our purpose of building a randomized generalization design which preserves the utility of normal search users, we note the following three desirable property of a solution to (5).

P1. obfuscation factor γ - i.e., the ratio between number of elements on each side of (5) - must be flexible. Ideally we would like to construct a solution for an arbitrary γ.

P2. The solution should minimize $|V_{\text{RG}}(t)|$ for all tuples, in order to maintain the utility of search query answers - Note that the larger $V_{\text{RG}}(t)$ is, the more tuples returned in the generalized query answer do not appear in the original answer.

P3. The solution should also maximize h, so that the adversarial query cost (i.e., $\Omega(u^{(h-1)/h})$ where u is the number of ZIP codes in Washington, DC) can be maximized.

We identified a P-T-E solution according to the above three properties. It is shown in the following two equations, with an example of $\gamma = 2$ depicted in Figure 2.

$$\underbrace{(\gamma+1) + \cdots + (\gamma+1)}_{\gamma^2 + \gamma - 1 \text{ items}} = \underbrace{1 + \cdots + 1}_{\gamma^3 + \gamma^2 - \gamma - 1 \text{ items}} + (\gamma^2 + \gamma)$$
$$\underbrace{(\gamma+1)^2 + \cdots + (\gamma+1)^2}_{\gamma^2 + \gamma - 1 \text{ items}} = \underbrace{1^2 + \cdots + 1^2}_{\gamma^3 + \gamma^2 - \gamma - 1 \text{ items}} + (\gamma^2 + \gamma)^2$$

This solution can be understood as follows: First, we (arbitrarily) classify all tuples in D_1 into groups with $\gamma^2 + \gamma - 1$ tuples each, and all tuples in D_2 into groups with $\gamma^3 + \gamma^2 - \gamma$ tuples each[8]. One can see that we have the same number of groups for both databases. Then, for each group in D_1, we assign $|V_{\text{RG}}(t)| = \gamma + 1$ to all of its tuples - i.e., according to the LHS of the above P-T-E solution. For each group in D_2, we assign $|V_{\text{RG}}(t)| = 1$ for all but one tuples, and $|V_{\text{RG}}(t)| = \gamma^2 + \gamma$ for the remaining one - i.e., according to the RHS of the P-T-E solution. The above two equations ensure

[8]If $\gamma^2 + \gamma - 1$ cannot exactly divide n, we can simply ignore the remaining tuples as they form only a small percentage of the database (given $\gamma \ll n$).

that the sum of $|V_{\text{RG}}(t)|$ and $|V_{\text{RG}}(t)|^2$ are the same for D_1 and D_2, achieving $h = 3$ according to Theorem 4.1.

One can see that our solutions satisfy property P1 because it supports an arbitrary obfuscation factor γ. In addition, the values of $|V_{\text{RG}}(t)|$ in our solution are close to, if not exactly, the minimum values possible to achieve $h \geq 2$ - an obvious reason being that the mean of $|V_{\text{RG}}(t)|$ for D_1 is at least γ even for $h = 1$. This satisfies P2. The solution is not perfect on P3, as it achieves only $h = 3$. Nonetheless, even $h = 3$ offers significant increase on adversarial query cost than $h = 2$ (as we shall show in the experiments), because substantially more queries are needed to return the same tuple thrice than twice. Given the known difficulty of finding P-T-E solutions for large h with small values on each side, and the fact that the focus of this paper is not on number theory, we decide to leave as an open problem finding solutions for (5) when $h > 3$.

Figure 2: Demonstration of Our P-T-E Solution.

We focused on the indistinguishability of D_1 and D_2 in the above discussion. It is straightforward to extend the solution to solve the generic problem - i.e., for any single GVS U, design V_{RG} so as to maintain an *indistinguishable segment* $[n_0, \gamma \cdot n_0]$ for the number of tuples belonging to U, such that as long as two databases D_1 and D_2 both fall into this size range, they are indistinguishable from the view of an adversary issuing queries corresponding to U unless the adversarial query cost reaches $\Omega(|U|^{2/3})$. We describe the detailed extension in Appendix C.3.

4.2 Randomized Generalization for All GVS

We now consider the design of randomized generalization for different GVS. Intuitively, the main requirement here is for the design over each GVS to be somehow "separated" from each other, so that an adversary cannot use queries issued over one GVS to infer aggregates over another. A straightforward solution appears to be an extension of the match factor μ in RG-SIMPLE - i.e., to determine μ for each GVS independently. Nonetheless, we show in Appendix D that this design is vulnerable to a sampling-based attack which can accurately estimate the sensitive aggregates.

Our method for addressing the problem is to construct for each GVS a number of *indistinguishable segments* based on the obfuscation factor[9] γ

$$[\gamma^2 + \gamma - 1, \gamma^3 + \gamma^2 - \gamma), [\gamma^3 + \gamma^2 - \gamma, \gamma^4 + \gamma^3 - \gamma^2],$$
$$\ldots, [\gamma^{i+1} + \gamma^i - \gamma^{i-1}, \gamma^{i+2} + \gamma^{i+1} - \gamma^i], \ldots \quad (6)$$

For each GVS U, let $n(U)$ be the number of tuples in the database with ZIP belonging to U. With the indistinguishable-segment design, we first determine which segment $n(U)$ falls into, say the i-th segment $[\gamma^{i+1} + \gamma^i - \gamma^{i-1}, \gamma^{i+2} + \gamma^{i+1} - \gamma^i]$. Then, we design V_{RG} (according to the above-described P-T-E solution) to guarantee that no adversary, unless issuing $\Omega(|U|^{2/3})$ queries in U, can distinguish among cases where $n(U)$ takes any possible values in the i-th segment.

[9]Note that here we assume each GVS to have at least $\gamma^2 + \gamma - 1$ tuples. When γ is small (e.g., 2 or 4), the vast majority of GVS in a practical database satisfy this condition, and we can simply ignore a small number of GVS which do not. Please refer to §4.3 for reasons why γ should not be set to be too large for RG-SUPPRESS anyway.
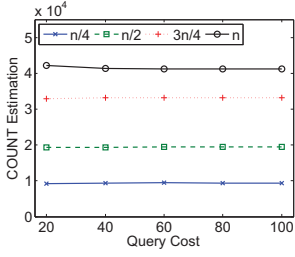
Figure 3: UNBIASED-AGG.
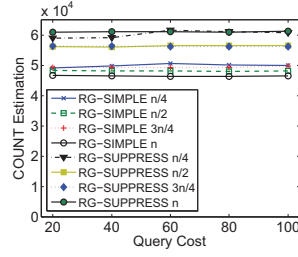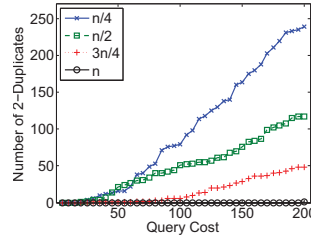
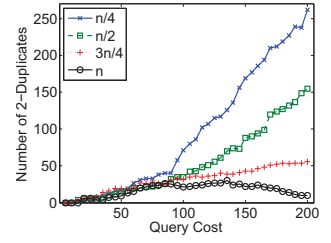

Figure 4: After Suppression.

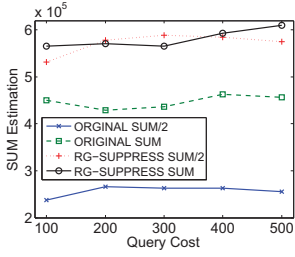Figure 5: Vulnerability.

Figure 6: Remedy.

Figure 7: SUM Suppression.
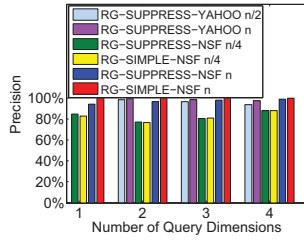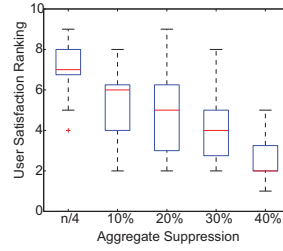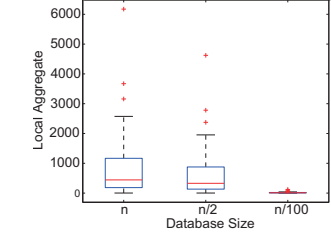

Figure 8: Precision.

Figure 9: User Studies.

Figure 10: Setting of γ.

4.3 Algorithm RG-SUPPRESS and its Privacy Guarantees

We now combine the above two enhancements to develop RG-SUPPRESS, our general-purpose algorithm. RG-SUPPRESS first determines the indistinguishable segment for each GVS based on the indistinguishable segment design in (6), and then designs $V_{\text{RG}}(t)$ based on our P-T-E solution for each GVS to achieve its indistinguishable segment. The pseudo-code of RG-SUPPRESS is presented in Appendix A. The following theorem shows the aggregate-suppression guarantee achieved by RG-SUPPRESS.

THEOREM 4.2. *For any database with a sufficiently large size* n, *RG-SUPPRESS achieves* $\langle \epsilon, \delta, w \cdot u_{\min}^{2/3}, 50\% \rangle$-*guarantee for a COUNT or SUM aggregate* $Res(Q_A)$ *if*

$$\epsilon \leq 2\text{erf}^{-1}(\delta) \cdot \min\left(\sqrt{\frac{\text{var}(q_A)}{w}} \cdot n_{\text{GVS}}, \right.$$

$$\left. \sqrt{\frac{1}{48w \cdot n_{\text{GVS}}}} \cdot \frac{\gamma \cdot Res(q_A)}{u_{\min}^{1/3}} \right), \qquad (7)$$

where $u_{\min}$ *is the size of the smallest GVS,* n_{GVS} *is the number of different GVS,* $\text{var}(q_A)$ *is the variance of* q_A *over (all tuples in) each GVS, and* $\text{erf}^{-1}(\cdot)$ *is the inverse error function.*

Please refer to Appendix E for the proof. We make two important observations from the theorem. One is that, compared with Corollary 3.1.1 for RG-SIMPLE, the adversarial query cost mandated by RG-SUPPRESS is significantly larger, thanks to (1) an increase from $\sqrt{u_{\min}}$ to $u_{\min}^{2/3}$, which is enabled by our P-T-E solution in §4.1; and (2) the multiplicative factor w enabled by our isolated generalization across different GVS, as described in §4.2.

The other observation is that the obfuscation factor γ should not be set too large, because then the MIN function in (7) would be determined by the first item which is independent of γ. Intuitively, the reason can be stated as follows. No matter how large γ is, an adversary can always first randomly select a GVS U, then crawl all tuples belonging to U, and finally use the aggregate over crawled tuples as a sample to estimate the overall value. While this process

in general requires a very large query cost, it is independent of γ and is always a fall-back option for the attacker when γ is large - as we shall show in the experimental results. For practical purposes, γ should be set to be a small value such as 3 or 5 - after all, with an aggregate-suppression technique which only returns tuples in the original database, it would be unreasonable to assume that an attacker can be misled into believing in the database being 100 times larger than its real size.

5. EXPERIMENTAL RESULTS

We conducted experiments on two datasets crawled from real-world hidden databases: NSF Fastlane Awards and Yahoo! Auto. The detailed experimental setup is described in Appendix G.

Aggregate Suppression: We tested both RG-SIMPLE and RG-SUPPRESS for protecting COUNT(*) over Fastlane. in order to evaluate the effectiveness of RG-SIMPLE on "emulating" a larger database and RG-SUPPRESS on maintaining indistinguishability, we generated four databases of size $n/4$, $n/2$, $3n/4$, n through sampling. For RG-SIMPLE, we set $\mu = 4$, 2, 4/3, and 1 for the four databases, respectively - i.e., all emulating a database of size n. For RG-SUPPRESS, we set $\gamma = 4$ to include all four databases in the same indistinguishable segment. Figures 3 and 4 depict the estimations produced by UNBIASED-AGG before and after aggregate suppression. One can see that, once RG-SIMPLE or RG-SUPPRESS is applied, the predictions of UNBIASED-AGG are almost equal for all 4 databases, demonstrating their effectiveness.

We tested the vulnerability of RG-SIMPLE in the same setting. Since Award Number, a candidate key, is not mandatory, RG-SIMPLE violates the condition in Theorem 3.1. We repeatedly sample a ZIP code (without replacement) from a GVS and queries it to identify duplicate keys. Figure 5 depicts the number of 2-duplicates, i.e., keys returned by two different queries. One can see that, after only 20 queries, the attacker can start distinguishing the 4 databases. This verifies the vulnerability discussed in §3. The same test on RG-SUPPRESS, as depicted in Figure 6, shows that an adversary now needs many more queries (about 100) to make the distinction. This shows the effectiveness of our P-T-E solution.

We also tested RG-SUPPRESS for protecting a SUM aggregate,

this time over the Yahoo! Auto data which has two mandatory contributes, as described in Appendix G. One can see from Figure 7 that, after RG-SUPPRESS is applied, the estimations generated by UNBIASED-AGG becomes the same over two databases where the SUM aggregate differs by two times.

Utility: We tested the effectiveness of our two algorithms on maintaining the utility of search queries in two ways: a theoretical measure of *precision*, i.e., the percentage of tuples in the (generalized) query answer which also appear in the original answer, and real-world user studies. Figure 8 depicts the precision offered by RG-SIMPLE and RG-SUPPRESS in the above-mentioned tests. The generation of search-query workload is described in Appendix G. One can see from the figure that our algorithms offer high precisions (about 90%) over both datasets.

Figure 9 depicts the user study results in which we ask respondents to rate their satisfiability (10-best, 1-worst) towards query answers generated by RG-SUPPRESS (left-most column) and the previous dummy-&-CAPTCHA based COUNTER-SAMPLER [8] when 10%-40% of dummy tuples are inserted, respectively. Please refer to Appendix G for details about the user study. One can see from the box plots of user ratings shown in Figure 9 that survey respondents are significantly more satisfied with RG-SUPPRESS than with the previous COUNTER-SAMPLER, even when the latter inserts a very small number (10%) of dummy tuples (note that $> 25\%$ dummies are used in all experiments in [8]).

Parameter Setting for γ: Finally, we verified our argument in §4.3 that γ should not be set too large. Recall from §4.3 that if γ is too large, an adversary can crawl (all tuples in) a GVS, and then use the local aggregate (over this GVS) to estimate the global aggregate. Figure 10 shows the box plots for local aggregates over databases of sizes $n/100$, $n/2$, and n, respectively. One can see that after crawling just one GVS, an adversary is very likely to make a distinction between $n/100$ and the other two databases - i.e., $\mu = 50$ is unsustainable with even a moderate adversarial query cost. On the other hand, note that an adversary cannot do the same when $\gamma = 2$, because of the similarity between box plots for $n/2$ and n.

6. RELATED WORK

Data Analytics over Hidden Databases: There has been prior work on crawling, sampling, and aggregate estimation over the hidden web, specifically over text [5, 6] and structured [20] hidden databases and search engines [4, 18]. Particularly related work includes [7] and references therein, which considered sampling and aggregate estimation over structured hidden databases.

Protection of Individual Tuple Privacy: Much research on privacy/security issues in databases and data mining focused on the protection of individual tuples which is complementary to our proposed research - e.g., access control [15], query auditing [17], tuple encryption/perturbation [1, 2, 21], query answer perturbation [9], etc. More closely related to this paper is the existing work on protecting aggregation information, in particular sensitive association rules, in frequent pattern mining [3, 11, 22].

7. CONCLUSION

In this paper, we have studied the problem of suppressing sensitive aggregates, in particular SUM and COUNT queries, over a hidden database while maintaining the utility of individual search queries to normal users. We developed randomized generalization, a novel technique which provides rigid aggregate-suppression guarantees while ensuring that all tuples returned in response to a search query are actual tuples in the hidden database. We instantiated randomized generalization with two algorithms, RG-SIMPLE, which works for a limited type of databases, and RG-SUPPRESS,

a general-purpose algorithm. We provided theoretical analysis and extensive experiments to show the effectiveness of our approaches.

8. REFERENCES

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD*, pages 86–97, 2003.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439–450, 2000.

[3] M. Atallah, E. Bertino, A. K. Elmagarmid, M. Ibrahim, and V. S. Verykios. Disclose limitation of sensitive rules. In *Knowledge and Data Exchange Workshop*, pages 45–52, 1999.

[4] Z. Bar-Yossef and M. Gurevich. Efficient search engine measurements. In *WWW*, pages 401–410, 2007.

[5] Z. Bar-Yossef and M. Gurevich. Mining search engine query logs via suggestion sampling. In *VLDB*, pages 54–65, 2008.

[6] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW*, pages 379–388, 1998.

[7] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *SIGMOD*, pages 855–866, 2010.

[8] A. Dasgupta, N. Zhang, G. Das, and S. Chaudhuri. Privacy preservation of aggregates in hidden databases: Why and how? In *SIGMOD*, pages 153–164, 2009.

[9] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[10] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: an approach based on dynamical systems. *The VLDB Journal*, 8(3-4):222–236, 2000.

[11] A. Gkoulalas-Divanis and V. S. Verykios. An integer programming approach for frequent itemset hiding. In *CIKM*, pages 748–757, 2006.

[12] P. Golle. Machine learning attacks against the asirra captcha. In *CCS*, pages 535–542, 2008.

[13] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[14] G. H. Hardy and E. M. Wright. *Introduction to the Theory of Numbers, 5th edition*. Clarendon Press, Oxford, 1979.

[15] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *TODS*, 26(2):214–260, 2001.

[16] X. Jin, N. Zhang, and G. Das. Attribute domain discovery for hidden web databases. In *SIGMOD*, pages 553–564, 2011.

[17] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. *PODS*, pages 118–127, 2005.

[18] K.-L. Liu, C. Yu, and W. Meng. Discovering the representative of a search engine. In *CIKM*, pages 652–654, 2002.

[19] E. H. McKinney. Generalized birthday problem. *American Mathematical Monthly*, 73:385–387, 1966.

[20] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, pages 129–138, 2001.

[21] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[22] V. S. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association rule hiding. *TKDE*, 16(4):434–447, 2004.

APPENDIX

A. PSEUDO-CODE OF RG-SIMPLE AND RG-SUPPRESS

Algorithm 1 RG-SIMPLE

1: Generate μ uniformly at random from $[1, \gamma]$.
2: **for** each tuple $t \in D$, with probability $(\mu - 1)/(\gamma - 1)$ **do**
3:     Assign $V_{\mathrm{RG}}(t)$ to be the mandatory attribute(s) value of t.
4:     **repeat**
5:         Choose t' uniformly at random from all tuples with mandatory attribute(s) value in the same GVS as t. Let v be the mandatory attribute(s) value of t'.
6:         $V_{\mathrm{RG}}(t) \leftarrow V_{\mathrm{RG}}(t) \cup \{v\}$.
7:     **until** $|V_{\mathrm{RG}}(t)| = \gamma$.
8: **end for**
9: In query answers, replace the mandatory attribute(s) value of each returned tuple with the corresponding GVS.

Algorithm 2 RG-SUPPRESS

1: **for** each GVS U **do**
2:     Find the indistinguishable segment for $n(U)$, the number of tuples in the database with mandatory attribute(s) value in U. Let it be $[\gamma^{i+1} + \gamma^i - \gamma^{i-1}, \gamma^{i+2} + \gamma^{i+1} - \gamma^i]$. Compute $\epsilon = (n(U) - \gamma^{i+1} - \gamma^i + \gamma^{i-1})/(\gamma^{i+2} - 2\gamma^i + \gamma^{i-1})$.
3:     Randomly distribute the $n(U)$ tuples into $\lfloor \gamma^{i-1} \cdot \epsilon \rfloor$ groups with $\gamma^2 + \gamma - 1$ tuples each and $\lfloor \gamma^{i-1} \cdot (1-\epsilon) \rfloor$ groups with $\gamma^3 + \gamma^2 - \gamma$ tuples each. Ignore the remaining tuples.
4:     **for** each $(\gamma^2 + \gamma - 1)$-tuple group **do**
5:         Assign $|V_{\mathrm{RG}}(t)| = \gamma + 1$ to each tuple t in the group.
6:     **end for**
7:     **for** each $(\gamma^3 + \gamma^2 - \gamma)$-tuple group **do**
8:         Randomly choose a tuple t_0 from the group and assign $|V_{\mathrm{RG}}(t_0)| = \gamma^2 + \gamma$.
9:         Assign $|V_{\mathrm{RG}}(t)| = 1$ to all other tuples in the group.
10:    **end for**
11:    **for** each tuple in U **do**
12:       Based on $|V_{\mathrm{RG}}(t)|$, randomly generate $V_{\mathrm{RG}}(t)$ in the same way as RG-SIMPLE.
13:    **end for**
14: **end for**
15: In query answers, replace the mandatory attribute(s) value of each returned tuple with the corresponding GVS.

B. PROOF OF THEOREM 3.1

THEOREM 3.1. *For a database in which non-mandatory attributes $A_2, \ldots, A_m$ in combination does not form a (super-) key, RG-SIMPLE achieves $\langle \epsilon, \delta, \infty, p \rangle$-guarantee for a COUNT or SUM aggregate $Res(Q_A)$ if*

$$\epsilon \le ((\gamma - 1) \cdot p + 1) \cdot Res(Q_A) \cdot \left(1 - \frac{1}{\gamma}\right) \cdot \delta, \qquad (8)$$

and the database size n is sufficiently large.

PROOF. We prove the case where Q_A is SELECT COUNT(*) FROM D. As discussed in §3.2, extensions to other SUM and COUNT queries follow in analogy. Since all non-mandatory attributes in combination does not form a superkey, for any given workload of search queries $\langle q_1, \ldots, q_r \rangle$, an adversary cannot distinguish between two sets of query answers as long as $\forall i \in [1, r]$,

both sets have the same $|q_i|$ - i.e., include the same number of tuples in the answer to q_i. Consider the value of mandatory attribute A_1 for each tuple in the database to be randomly generated from an underlying distribution. One can see that for any given q, after RG-SIMPLE is applied, $|q|$ follows Poisson distribution when the database size n is sufficiently large. Since a Poisson distribution is uniquely determined by its mean, it is sufficient to prove that no adversary can violate the guarantee based on the expected number (i.e., mean of size) of tuples returned for all search queries.

We start by considering an arbitrary search query q (other than SELECT * FROM D) and the posterior belief of an adversary on the database size after learning $E(|q|)$, i.e., the expected number of queries returned by q. Let μ be the match factor used by RG-SIMPLE, $r = n - k$, and $p(q)$ be the probability for a tuple to satisfy q according to the underlying distribution. We shall show that for all $\omega \in [\mu/\gamma, \mu]$, the posterior probability of the database size being equal to $k + \omega \cdot r$ is the same. This is proved by the following equation.

$$\Pr\{|D| = k + \omega \cdot r \,|E(|q|) = s\}$$
$$= \frac{\Pr\{E(|q|) = s||D| = k + \omega \cdot r\} \cdot \Pr\{|D| = k + \omega \cdot r\}}{\Pr\{|q| = s\}} \quad (9)$$
$$= \frac{\Pr\{\mu = (s/(p(q) \cdot \omega \cdot r))\} \cdot \Pr\{|D| = k + \omega \cdot r\}}{\Pr\{|q| = s\}} \quad (10)$$

According to the design of RG-SIMPLE, there is always $s/(p(q) \cdot \omega \cdot r) \in [1, \gamma]$ for $\omega \in [\mu/\gamma, \mu]$. Thus, $\Pr\{\mu = (s/(p(q) \cdot \omega \cdot r))\}$ remains the same for all $\omega \in [\mu/\gamma, \mu]$. On the other hand, since the adversary has no prior knowledge of the database size, $\Pr\{|D| = k + \omega \cdot r\}$ is also constant for all ω. As a result, $\Pr\{|D| = k + \omega \cdot r|E(|q|) = s\}$ is constant for all $\omega \in [\mu/\gamma, \mu]$.

One can see from (10) that an adversary cannot learn any information for identifying $\omega \in [\mu/\gamma, \mu]$ from any arbitrary query answer q. According to the design of RG-SIMPLE, each query answer q is independently generated. Thus, no matter how many queries an adversary issue, it cannot make a distinction between databases with sizes in $[k + \mu \cdot (n - k)/\gamma, k + \mu \cdot (n - k)]$, a range of width $\mu \cdot (n - k) \cdot (1 - 1/\gamma)$. Thus, no adversary can win an $\langle \epsilon, \delta, \infty \rangle$-privacy game if $\epsilon \le \mu \cdot (n - k) \cdot (1 - 1/\gamma) \cdot \delta$. Since the probability for $\mu \ge (\gamma - 1)p + 1$ is $1 - p$, no adversary can win an $\langle \epsilon, \delta, \infty \rangle$-privacy game with probability greater than p if $\epsilon \le ((\gamma - 1) \cdot p + 1) \cdot (n - k) \cdot (1 - 1/\gamma) \cdot \delta$. □

C. OUR P-T-E SOLUTION

C.1 Proof of Theorem 4.1

THEOREM 4.1. *A sufficient and necessary condition for two databases D_1 and D_2 to remain indistinguishable until an adversary receives the same tuple from at least h different queries is*

$$\sum_{t \in D_1} |V_{\mathrm{RG}}(t)|^i = \sum_{t \in D_2} |V_{\mathrm{RG}}(t)|^i. \qquad (11)$$

for all $i = 1, \ldots, h - 1$.

PROOF. Consider the information learned by an adversary after issuing a number of queries with predicates being the mandatory attributes equal to various values in its domain (i.e., $A_1 = v$ where $v \in \Theta_1$). We can summarize the adversary's view as

$$R_1, R_2, \ldots, \text{ with } R_1 \supseteq R_2 \supseteq \cdots \qquad (12)$$

where R_i ($i \ge 1$) is the set of tuples which appear in at least i different query answer(s). Since an adversary has no prior knowledge of the database instance, one can see that the two databases, D_1 and D_2 are indistinguishable from the view of the adversary if and only

if they have the same expected values[10] on R_1, R_2, etc. Let $r_j(i)$ be the expected value of R_i for D_j.

If an adversary issues s queries, then $\forall x \in \{1, 2\}$, we have

$$r_x(1) = s \times \sum_{t \in \mathsf{D}_x} \frac{|V_{\mathrm{RG}}(t)|}{|U|} \tag{13}$$

For $i > 1$, we have

$$r_x(i) = r_x(i-1) - \sum_{t \in \mathsf{D}_x} \left[\binom{s}{i-1} \cdot \left(\frac{|V_{\mathrm{RG}}(t)|}{|U|} \right)^{i-1} \cdot \right.$$
$$\left. \left(1 - \frac{|V_{\mathrm{RG}}(t)|}{|U|} \right)^{s-i+1} \right], \tag{14}$$

which can be approximated as

$$r_x(i) \approx r_x(i-1) - \sum_{t \in \mathsf{D}_x} \left[\frac{\binom{s}{i-1}}{|U|^{i-1}} \cdot (|V_{\mathrm{RG}}(t)|^{i-1} - (s-i+1) \cdot |V_{\mathrm{RG}}(t)|^i) \right]. \tag{15}$$

According to (13) and the above approximation, we can maintain $r_1(i) = r_2(i)$ for $i = 1, \ldots, h-1$ if and only if

$$\sum_{t \in \mathsf{D}_1} |V_{\mathrm{RG}}(t)|^i = \sum_{t \in \mathsf{D}_2} |V_{\mathrm{RG}}(t)|^i. \tag{16}$$

for all $i = 1, \ldots, h-1$. $\square$

C.2 Connection with Prouhet-Tarry-Escott

Consider (11). Let $a_1, \ldots, a_n$ be the value of $|V_{\mathrm{RG}}(t)|$ for the n tuples in D_1, and $b_1, \ldots, b_{\gamma n}$ be the value of $|V_{\mathrm{RG}}(t)|$ for the γn tuples in D_2. Then, (11) is reduced to finding positive integer solutions for $a_1, \ldots, a_n, b_1, \ldots, b_{\gamma n}$ such that $\forall i \in [1, h-1]$,

$$\sum_{j=1}^{n} a_j^i = \sum_{j=1}^{\gamma n} b_j^i. \tag{17}$$

One can easily observe the connection between (17) and the Prouhet-Tarry-Escott (P-T-E) problem [14] which requires

$$\sum_{j=1}^{\alpha} c_j^i = \sum_{j=1}^{\alpha} d_j^i. \tag{18}$$

for all $j \in [1, \beta]$. In particular, any solution for (17) forms a solution for the P-T-E problem with $\alpha = \gamma n$ and $\beta = h - 1$.

On the other hand, a solution to P-T-E may also be used to construct a solution for our problem. For example, a known solution [14] for P-T-E is $[c_1, \ldots, c_5] = [0, 4, 8, 16, 17]$ and $[d_1, \ldots, d_5] = [1, 2, 10, 14, 18]$ which satisfy (18) for $i \in [1, 4]$. This solution can serve as the basis for the following design of $|V_{\mathrm{RG}}|$ when $\gamma = 5/4$: for each $j \in [2, 5]$, one fourth of all tuples in D_1 have $|V_{\mathrm{RG}}| = c_i$. Meanwhile, for each $j \in [1, 5]$, one fifth of all tuples in D_2 have $|V_{\mathrm{RG}}| = d_i$. One can see that no adversary can distinguish D_1 from D_2 until observing the same tuple in at least 5 different query answers. Nonetheless, this solution can only support an obfuscation factor of $5/4$. This is insufficient for practical privacy requirements - e.g., making databases of size n and $4n$ indistinguishable.

---

[10]Note that only expected values are of importance here because, according to our assumptions discussed in §2, the adversary has no prior knowledge to distinguish among different ZIP codes in the same city, and therefore can only issue these ZIP codes in a uniform random fashion.

C.3 Generic Extension of Our P-T-E Solution

To address the generic problem, we apply our P-T-E solution as follows. For a given database, let $n(U)$ be the number of tuples which belong to U, and $t_1, \ldots, t_{n(U)}$ be these tuples. We aim to design V_{RG} such that $\forall n(U) \in [n_0, \gamma \cdot n_0]$, there is

$$\sum_{i=1}^{n(U)} |V_{\mathrm{RG}}(t_i)| = (\gamma + 1) \cdot n_0, \sum_{i=1}^{n(U)} |V_{\mathrm{RG}}(t_i)|^2 = (\gamma + 1)^2 \cdot n_0.$$

Note that the solution in §4.3 satisfies two special cases where $n(U) = n_0$ and $n(U) = \gamma \cdot n_0$. To extend this solution to other values of $n(U) \in (n_0, \gamma \cdot n_0)$, we first partition the $n(U)$ tuples into $\lfloor \frac{n_0 \cdot \epsilon}{\gamma^2 + \gamma - 1} \rfloor$ groups with $\gamma^2 + \gamma - 1$ tuples (each) and $\lfloor \frac{n_0 \cdot (1-\epsilon)}{\gamma^2 + \gamma - 1} \rfloor$ groups with $\gamma^3 + \gamma^2 - \gamma$ tuples each, where

$$\epsilon = \frac{\gamma \cdot n_0 - n(U)}{(\gamma - 1) \cdot n_0}. \tag{19}$$

Then, for each $(\gamma^2 + \gamma - 1)$-tuple group, we assign $|V_{\mathrm{RG}}| = \gamma + 1$ to all tuple in the group. Again, the actual generation of V_{RG} is a random selection of values in U according to the original distribution in the database. For each $(\gamma^3 + \gamma^2 - \gamma)$-tuple group, we assign $|V_{\mathrm{RG}}| = \gamma^2 + \gamma$ to one (randomly chosen) tuple in the group, and a size of 1 to all other tuples. It is easy to verify, based on the correctness of our P-T-E solution, that the indistinguishability is guaranteed unless the adversary issues $\Omega(|U|^{2/3})$ queries.

D. PROBLEM OF SIMPLY EXTENDING μ

A straightforward solution seems to be an extension of the match factor μ in RG-SIMPLE - in particular, instead of using the same μ over all tuples as in RG-SIMPLE, one instead generates μ i.i.d. (say again uniformly at random from $[1, \gamma]$) for each GVS, ensuring independence. Nonetheless, this method has a fatal flaw when there is a large number of GVS - instead of estimating μ for each value, the adversary can simply observe that the expected value of μ over all tuples is close to $(1 + \gamma)/2$, and then adjust the result of UNBIASED-AGG based on this knowledge to accurately estimate COUNT(*).

The fundamental problem of reusing match factor μ is that the distribution of μ is part of the algorithm design, and thus must be treated as public information to the adversary. RG-SIMPLE does not have this problem because it generates μ once and for all (tuples), preserving the uncertainty of μ which forces an adversary to estimate it through issuing queries. The extension, however, reduces such uncertainty by generating μ i.i.d. for each GVS.

E. PROOF OF THEOREM 4.2

THEOREM 4.2. *For any database with a sufficiently large size n, RG-SUPPRESS achieves $\langle \epsilon, \delta, w \cdot u_{\min}^{2/3}, 50\% \rangle$-guarantee for a COUNT or SUM aggregate $Res(Q_A)$ if*

$$\epsilon \leq 2\mathrm{erf}^{-1}(\delta) \cdot \min\left(\sqrt{\frac{\mathrm{var}(q_A)}{w}} \cdot n_{\mathrm{GVS}}, \right.$$
$$\left. \sqrt{\frac{1}{48w \cdot n_{\mathrm{GVS}}}} \cdot \frac{\gamma \cdot Res(q_A)}{u_{\min}^{1/3}} \right), \tag{20}$$

where $u_{\min}$ is the size of the smallest GVS, n_{GVS} is the number of different GVS, $\mathrm{var}(q_A)$ is the variance of q_A over (all tuples in) each GVS, and $\mathrm{erf}^{-1}(\cdot)$ is the inverse error function.

PROOF. (Sketch) Similar to the proof of Theorem 4.1, we present the proof for COUNT(*), with other SUM and COUNT queries follow in analogy. Let $\Gamma : \{q_1, \ldots, q_c\}$ be the set of search queries

issued by the adversary. Let A_1 be the mandatory attribute. If there are multiple mandatory attributes, then we can always combine them into A_1 with domain Θ_1 being the Cartesian product of the domains of all mandatory attributes. Consider the worst-case scenario where k is sufficiently large such that any query of the form SELECT * FROM D WHERE $A_1 = v$ where $v \in \Theta_1$. One can see that all q_i ($i \in [1, c]$) must be of this form because the answer to any other search query, with the only exception of SELECT * FROM D, can be inferred from the answer to a corresponding query of this form.

Consider any value $v \in \Theta_1$ such that there exists q_i ($i \in [1, c]$) which has predicate $A_1 = v$. Let U be the GVS of v. There are two possibilities for the "coverage" of Γ on U: One is that an expected number[11] of $|U|^{2/3}$ or more queries in Γ are issued with value in U. In this case, we again consider the worst-case scenario where the adversary learns the exact value of $n(U)$, i.e., the number of tuples with value of A_1 in U. The other possibility is that the expected number of queries in Γ with value in U is smaller than $|U|^{2/3}$. In this case, the probability for the adversary to find the same key 3 times is smaller than 50%. Thus, with probability of at least 50%, no adversary can make a distinction between any values in the indistinguishable segment corresponding to U. Note that the width of such an indistinguishable segment is at least $\gamma \cdot n(U)/2$.

Given the two types of coverage, the adversarial design of a search query workload is essentially reduced to a linear programming problem, with a constraint being a total query cost of $w \cdot u_{\min}^{2/3}$ or less. Note that for the two types discussed above, Type-1 coverage requires a query cost of $|U|^{2/3} \geq u_{\min}^{2/3}$ but produces an intra-GVS variance of $\sigma_{\text{intra}}^2 = 0$, while Type-2 coverage requires a query cost of 1 but produces an intra-GVS variance of

$$\sigma_{\text{intra}}^2 \geq \gamma^2 \cdot \frac{n(U)^2}{48} \tag{21}$$

because continuous uniform distribution over a width of x has variance $x^2/12$. The optimization in linear programming is to minimize the final estimation variance σ^2. Note that given an estimation variance σ^2, no adversary can win a $\langle \epsilon, \delta, w \cdot u_{\min}^{2/3} \rangle$-game if

$$\epsilon \leq 2\text{erf}^{-1}(\delta) \cdot \sigma. \tag{22}$$

After mathematical manipulation, one can see that the optimization function is concave - i.e., the minimum value is taken either when all coverages are Type-I, or when all coverages are Type-II. If all coverages are Type-I, the overall variance satisfies

$$\sigma^2 \geq \frac{\text{var}(n_U)}{w} \cdot n_{\text{GVS}}^2, \tag{23}$$

where $\text{var}(n_U)$ is the variance of all GVS COUNTs. If all coverages are Type-II, then

$$\sigma^2 \geq \frac{\sum_U (\gamma^2 \cdot n(U)^2)}{48 \cdot w \cdot u_{\min}^{2/3}} \geq \frac{\gamma^2 \cdot n^2}{n_{\text{GVS}} \cdot 48 \cdot w \cdot u_{\min}^{2/3}} \tag{24}$$

From (22), (23) and (24), we have

$$\epsilon \leq 2\text{erf}^{-1}(\delta) \cdot \min \left(\sqrt{\frac{\text{var}(n_U)}{w}} \cdot n_{\text{GVS}}, \right.$$

$$\left. \sqrt{\frac{1}{48w \cdot n_{\text{GVS}}}} \cdot \frac{\gamma \cdot n}{u_{\min}^{1/3}} \right), \tag{25}$$

which is a special case of (20) when q_A is SELECT COUNT(*) FROM D. The extension to other SUM and COUNT queries follow in analogy. □

---

[11]The expected value here is taken over the randomness of the design of adversarial query workload .
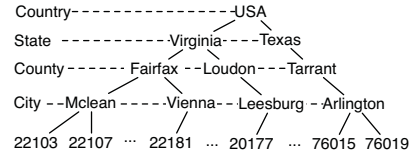

Figure 11: Concept Hierarchy of ZIP Code.

F. CONSTRUCTION OF GVS

We now consider the GVS construction for a mandatory attribute, and then discuss the case of multiple attributes being mandatory.

Recall that each GVS is supposed to include attribute values that, when being fuzzy-matched with each other, cause little inconvenience to a normal search user. Since GVS is by definition a subjective measure, a natural way to build it is to call upon a human expert. Nonetheless, manual construction can be expensive for a mandatory attribute with a large domain, e.g., ZIP code. Thus, we focus on the automated construction of GVS based on various auxiliary information indicating the "closeness" of two attribute values from a search user's perspective.

The automated construction of GVS is closely related to many data clustering algorithms studied in the data mining literature - e.g., hierarchical clustering algorithms such as ROCK [13], and attribute value clustering algorithms such as STIRR [10]. We would like to note that it is not our intention to reinvestigate these problems in this paper. Instead, we shall discuss as follows a few simple methods with which one can leverage the existing clustering algorithms to construct GVS. In particular, we consider two cases: (1) a concept hierarchy - either pre-defined or constructed through hierarchical clustering - is available for the mandatory attribute, (2) a pairwise distance function is available for the attribute values (either numerical or categorical).

For an attribute with a pre-defined concept hierarchy, the "closeness" of two attribute values can be measured by the distance between their corresponding two nodes in the hierarchy tree. As such, for a given maximum tolerable distance d, we can define each generalized value set as the set of all values under a node at level $g - \lfloor d/2 \rfloor$ of the tree, where g is the tree depth. In the example of ZIP code shown in Figure 11, our prior example of generalizing to the city level is corresponding to $d = 2$, while $d = 4$ would produce larger GVS, each containing all ZIP codes in a county. We would like to note that, in order to properly set the maximum tolerable distance d, the hidden database owner may need to perform user studies before hand and/or adjust the setting based on clickthrough rates observed after applying the aggregate suppression algorithm.

For an attribute without a concept hierarchy but with a pairwise distance function defined over all attribute values, either categorical or numerical, the problem of constructing GVS is in effect reduced to the problem of clustering.

We now consider the case where multiple attributes are mandatory on the interface. There are two scenarios where it may happen: (1) when the "semantics" of the hidden database requires multiple mandatory attributes - e.g., a local listing database like craiglist.com only returns interesting results when a user specifies both location and category, and (2) when no single attribute has a domain large enough to support the aggregate-suppression guarantee desired by the hidden database owner - note that when both $u_{\min}$ and n_{GVS} is small, Theorem 4.2 cannot support a strong guarantee on aggregate suppression. In the second case, the database owner may want to make more attributes mandatory to enable aggregate suppression, but at the expense of search query utility because a normal user faces more limitations on issuing search queries.

The construction of GVS and the design of our aggregate suppression algorithms can be easily extended to interfaces with more than one mandatory attributes. A simple way to understand the extension is to consider the combination of all mandatory attributes as one (composite) attribute, with domain equal to the Cartesian product of the domains of all mandatory attribute. One can see that RG-SIMPLE and RG-SUPPRESS can then be directly applied without change. For the purpose of constructing GVS, we can consider two values of the composite attribute to be "close" to each other if they have "close" values on every component (i.e., every mandatory attribute). Again, the aforementioned constructions based on concept hierarchies and pairwise distance functions can be readily applied.

G. EXPERIMENTAL SETUP

Hardware: All our experiments were conducted on a 2.6GHz Intel Core 2 Duo machine with 2GB RAM and Windows XP OS. All algorithms were implemented using C++.

Dataset: We conducted our experiments by using two crawled real-world datasets, NSF Fastlane Award Search (*http://www.nsf.gov/awardsearch/*) and Yahoo! Auto (*http://autos.yahoo.com/*).

Our Fastlane dataset contains 47,816 tuples with 10 attributes such as Zip Code, Award Amount, PI Organization, NSF Organization, etc. Their attribute domains range from 5 (for Award Amount) to 29,042 (for PI Name). Note that to study the case discussed in §3.4 where a key attribute appears in query answers but not in query selection conditions, we include the primary key attribute Award Number in the displayed results although it cannot be specified through the input interface. Consistent with the examples used in the paper, we chose Zip Code, which has 1,995 different values in the database, as the mandatory attribute. We constructed the GVS of ZIP Code according to its real-world concept hierarchy. In particular, all ZIP codes in the same state/territory are clustered into the same GVS, leading to a total of 58 GVS, with sizes ranging from 1 to 233.

We also used a larger dataset: Yahoo Auto!, which has 188,790 tuples and 38 attributes, with domain sizes ranging from 2 to 38. This dataset has been pre-processed to hide the real semantics of attribute values. To demonstrate the generality of RG-SIMPLE and RG-SUPPRESS to databases with multiple mandatory attributes, we arbitrarily chose two attributes, with domain sizes 16 and 27, respectively, as the mandatory attributes. We clustered all value combinations of the two attributes into 38 GVS, with sizes ranging from 13 to 219.

As discussed in §5, to evaluate the effectiveness of RG-SIMPLE on "emulating" a larger database and RG-SUPPRESS on maintaining indistinguishability, we sampled without replacement 25%, 50%, 75% of all tuples from the NSF Fastlane database to form databases of size $n/4 = 11954$, $n/2 = 23908$, $3n/4 = 35862$, respectively. Similarly, for the Yahoo! Auto dataset, we sampled without replacement 50% of all tuples to form a database of size 94,395.

Algorithms: For aggregate estimation attack, we tested the state-of-the-art HD-UNBIASED algorithm proposed in [7]. This algorithm is built upon a concept of query tree and has two parameters, r, the number of drill-downs performed over each subtree, and D_{UB}, the maximum subdomain size for each subtree. Following the parameter settings discussed in [7], we set $r = 4$ for both NSF Fastlane and Yahoo! Auto datasets. For D_{UB}, we set it to be the domain size of the top attribute in the query tree - i.e., a manda-

tory in our problem - which leads to a setting of $D_{UB} = 1995$ for NSF Fastlane data and $D_{UB} = 16$ for Yahoo! Autos.

For aggregate suppression, we tested both RG-SIMPLE and RG-SUPPRESS algorithms proposed in this paper. Besides the GVS construction, the only input to the algorithms is γ, the obfuscation factor. As discussed in §5, we used a default value of $\gamma = 4$ and 2 for NSF Fastlane and Yahoo! Auto, respectively, while also testing a case where $\gamma = 100$, in order to demonstrate that γ should not be set too large.

2-Duplicates Test: We now describe the detailed setup of our 2-duplicates test over the NSF Fastlane dataset which was used to illustrate the vulnerability of RG-SIMPLE. We first chose the largest GVS for ZIP code. Then, we sampled without replacement a ZIP code v from the GVS, and issued a search query SELECT * FROM D WHERE ZIP = v. We repeated the previous sampling process and recorded the number of 2-duplicates - i.e., tuples (with a unique value of Award Number) that were returned by two (different) queries. We also recorded the number of tuples returned by 3, 4, 5, and even larger number of queries. Our finding was that the number of 2-duplicates is the most effective way for an adversary to make a distinction among the four databases we tested, for both RG-SIMPLE and RG-SUPPRESS. Thus, we only presented the number of 2-duplicates in §5.

Search Query Workload for Utility Tests: Our search query workload for evaluating precision, our theoretical utility measure, is generated as follows. We first sampled with replacement 10,000 tuples from the dataset to form the basis of our workload. The rational is that, in general, a ZIP code with more cars (or any attribute value which matches more tuples) is also more likely to be queried by a normal search user. To generate a m-predicate workload, we produced one search query for each of the 10,000 sampled tuples. In particular, for each search query, in order to determine which predicates to specify in the query, we first include each of the i mandatory attributes as a predicate, and then choose the other $m - i$ predicates uniformly at random from all non-mandatory attributes. For each of the m predicates, the corresponding attribute value is the value of the sample tuple corresponding to the query.

User Study: To investigate the user-friendliness of our generalization strategy, we performed a user study to compare the utility of RG-SUPPRESS with the dummy insertion based technique used by COUNTER-SAMPLER [8]. It is important to note that since the objective of these two methods are different (i.e., ours for suppressing aggregate estimation while COUNTER-SAMPLER for countering uniform sampling), the degree of privacy protection provided by the two techniques are not comparable. Thus, we include user satisfaction ranking as the only metric during our study. The ranking is scaled from 1 to 10, where rank 10 means that a user has the most satisfaction. Our study was performed with participation from 13 students at the University of Texas at Arlington. First, each subject was given the same workload of 20 different queries from the Fastlane dataset. These queries were generated in the same way as we discussed above. After that, each subject was required to search these queries over the Fastlane dataset in 5 different scenarios: Scenario 1 contains the dataset produced by applying RG-SUPPRESS on the aforementioned $n/4$ dataset; Scenario 2-5 have datasets with 10%, 20%, 30%, 40% dummy tuples inserted, respectively by applying COUNTER-SAMPLER on the original n dataset. Finally, we collected each subject's satisfaction ranking.