QuerySnout: Automating the Discovery of Attribute Inference Attacks against Query-Based Systems

Ana-Maria Cretu*
Imperial College London
London, United Kingdom
a.cretu@imperial.ac.uk

Florimond Houssiau*
The Alan Turing Institute
London, United Kingdom
fhoussiau@turing.ac.uk

Antoine Cully
Imperial College London
London, United Kingdom
a.cully@imperial.ac.uk

Yves-Alexandre de Montjoye
Imperial College London
London, United Kingdom
deMontjoye@imperial.ac.uk

ABSTRACT

Although query-based systems (QBS) have become one of the main solutions to share data anonymously, building QBSes that robustly protect the privacy of individuals contributing to the dataset is a hard problem. Theoretical solutions relying on differential privacy guarantees are difficult to implement correctly with reasonable accuracy, while ad-hoc solutions might contain unknown vulnerabilities. Evaluating the privacy provided by QBSes must thus be done by evaluating the accuracy of a wide range of privacy attacks. However, existing attacks against QBSes require time and expertise to develop, need to be manually tailored to the specific systems attacked, and are limited in scope. In this paper, we develop QuerySnout, the first method to automatically discover vulnerabilities in query-based systems. QuerySnout takes as input a target record and the QBS as a black box, analyzes its behavior on one or more datasets, and outputs a multiset of queries together with a rule to combine answers to them in order to reveal the sensitive attribute of the target record. QuerySnout uses evolutionary search techniques based on a novel mutation operator to find a multiset of queries susceptible to lead to an attack, and a machine learning classifier to infer the sensitive attribute from answers to the queries selected. We showcase the versatility of QuerySnout by applying it to two attack scenarios (assuming access to either the private dataset or to a different dataset from the same distribution), three real-world datasets, and a variety of protection mechanisms. We show the attacks found by QuerySnout to consistently equate or outperform, sometimes by a large margin, the best attacks from the literature. We finally show how QuerySnout can be extended to QBSes that require a budget, and apply QuerySnout to a simple QBS based on the Laplace mechanism. Taken together, our results show how powerful and accurate attacks against QBSes can already be found by an automated system, allowing for highly complex QBSes to be automatically tested "at the pressing of a button". We believe this line of research to be crucial to improve the robustness of systems providing privacy-preserving access to personal data in theory and in practice.[1] [2]

CCS CONCEPTS

• **Security and privacy → Data anonymization and sanitization**; **Privacy-preserving protocols**.

KEYWORDS

anonymization; query-based systems; privacy attacks

ACM Reference Format:
Ana-Maria Cretu, Florimond Houssiau, Antoine Cully, and Yves-Alexandre de Montjoye. 2022. QuerySnout: Automating the Discovery of Attribute Inference Attacks against Query-Based Systems. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560581

1 INTRODUCTION

Our ability to collect and store data has exploded in the last decade. Coupled with the development of AI and new computational tools, this data has the potential to drive scientific advancements in healthcare [62] and the social sciences [39], and promises to revolutionize the way businesses and governments function.

However, most of this data is either personal or linked to individuals in one way or another. This raises serious privacy concerns, and as such this data falls under the scope of data protection laws such as the European Union's General Data Protection Regulation [1, 31]. Finding solutions to use data for good while preserving our fundamental right to privacy is a timely and crucial question.

Query-based systems (QBS), controlled interfaces through which analysts can query the data, have the potential to enable privacy-preserving anonymous data analysis at scale. As the curator keeps control over the data, they can audit queries sent by analysts and ensure that the answers returned do not reveal individual-level information. Typical queries include histograms, counts, correlations between attributes, and other aggregates over individual

*Equal contribution.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560581

[1] An extended version of the paper, including the Appendix, is available on arXiv and at https://cpg.doc.ic.ac.uk/querysnout.
[2] Our code is available at https://github.com/computationalprivacy/querysnout.

records. QBS interfaces can range from an online interface and API [14, 48], to languages such as SQL [27, 42] and the submission of scripts [47, 49].

It has however long been known that only releasing aggregate information is not sufficient to protect privacy. As early as 1979, Denning et al. [19] theorized difference attacks ("trackers") against databases accessible through user-specified counts. Researchers have shown that releasing marginals or simple counts can reveal the presence of a target individual in the private dataset [24, 32]. Famously, Dinur and Nissim proved in 2003 that a database can be reconstructed with good accuracy from a large number of aggregate statistics [20]. This issue is particularly acute for QBSes, where attackers can design queries to infer information about specific people, including by exploiting vulnerabilities or implementation bugs of the system.

In response to these risks, increasingly sophisticated defense mechanisms have been put in place. These include combining query set size restriction with noise addition mechanisms [27, 48], the use of unbounded static noise [22, 27], the introduction of limits on the number of queries (e.g., as a privacy budget in Differential Privacy [42]), and even online evaluation [44] and rewriting of queries [35]. Computer security tools such as access control, logging queries, code verification, and AI-based anomaly detection mechanisms are typically then deployed on top of QBS-specific mechanisms.

While these measures have helped prevent and mitigate privacy risks [27], the risk of unknown strong "zero-day" attacks has stalled the development and deployment of QBSes. Manually designing and implementing attacks against complex and expressive QBSes is a difficult and painstaking process. It typically requires careful analysis of the system by experts and can take months. Furthermore, existing attacks have only exploited a small subset of the syntax of modern expressive QBSes. Limiting the risk of existence of strong unmitigated attacks is thus essential to unlock the potential of QBSes and make individual-level data available to researchers and companies while strongly preserving privacy in practice.

Contributions. We here propose QuerySnout, the first method to automatically discover privacy vulnerabilities in query-based systems. We frame the discovery of attacks against a QBS as a black-box optimization problem. Specifically, we formalize an attack as a multiset of queries jointly with a mathematical rule (e.g., a machine learning model) to combine answers to the queries in order to reveal a particular secret. Given a threat model and black-box access to a QBS, we optimize the attacks with respect to their performance at inferring the secret.

At a high level, QuerySnout analyzes the query answering behavior of the QBS for patterns that can be used to infer the secret. QuerySnout is fully automated, combining (1) evolutionary search techniques to discover the right set of queries to ask with (2) a machine learning model ("rule") trained to infer the secret from query answers. To efficiently explore the search space of possible attacks, we design a novel mutation operator tailored to this problem.

We instantiate our approach on *attribute inference attacks*, where the secret is the sensitive attribute of a target record. We study two attack scenarios. The first scenario (AUXILIARY) assumes that an attacker has access to an auxiliary dataset similar to the private dataset, e.g., drawn from the same distribution. This is a common assumption across the broader literature [50, 58]. The second scenario (EXACT-BUT-ONE) assumes that an attacker has access to the entire private dataset protected by the QBS except for the sensitive attribute. Attack models like this one, relying on a very strong attacker, are used in the literature to evaluate privacy protections in the "worst-case" scenario [34]. They can help uncover flaws in the system design or implementation and audit systems.

We use QuerySnout to attack two real-world query-based systems, Diffix [27] and TableBuilder [48], and a generic query-based system. We show our attack to be highly successful against all three systems when protecting three real-world datasets, matching or outperforming previous expert-designed attacks [11, 30, 56]. A post-hoc analysis of the attacks found by QuerySnout in the AUXILIARY scenario suggests that they exploit the same vulnerability as the manual attacks, but more effectively. We propose a heuristic to extend our method to budget-based QBSes (such as those guaranteeing differential privacy [22]), and use QuerySnout to attack a simple QBS based on the Laplace mechanism. We show that our attack achieves near-optimal accuracy for $\varepsilon \in \{5, 10\}$.

Finally, we discuss how our method can be extended to other attack models – e.g., to perform membership inference attacks – and larger query syntaxes. Our results suggest that QuerySnout can be used to evaluate the privacy protection offered by QBSes and help improve their design by identifying vulnerabilities.

2 BACKGROUND

We consider a *data curator* entity, such as a business or a government agency, holding data about a set of users denoted by $U \subset \mathcal{U}$, with $\mathcal{U}$ a population of users. The data consists of values for an ordered set of attributes $\mathcal{A} = (a_1, \ldots, a_n)$, e.g., age and nationality. Each attribute $a \in \mathcal{A}$ has a set $\mathcal{V}_a$ of acceptable values. We denote by *individual record* r^u the data available about a user u, consisting of the corresponding values for attributes in $\mathcal{A}$: $r^u \in \mathcal{V}_{a_1} \times \cdots \times \mathcal{V}_{a_n}$. We denote by $r^u_{\mathcal{A}'}$ the restriction of a record to a subset of attributes $\mathcal{A}' \subset \mathcal{A}$ and by r^u_a the restriction to one attribute $a \in \mathcal{A}$. For instance, if the attributes collected are $\mathcal{A} = (age, nationality)$, a person's record could be (19, Bulgaria). Given a user set U, we call *dataset* the multiset of individual records D for users in U. This means that the same record can appear more than once if different users in U have the same values for these attributes.

We assume that the data curator allows data analysts to access information about the dataset D through a query-based system (QBS). The QBS allows the data analyst to retrieve answers to queries about the dataset without directly accessing the individual records [17]. The interface can uses a query language such as SQL or a GUI to define the semantics of queries that can be asked. Formally, we denote by *query space* $\mathcal{Q}$ the set of queries that can be asked to the system. We denote by $T : \mathcal{D} \times \mathcal{Q} \to \mathbb{R}$ the function that gives the *true answer* to a query over a dataset.

Formally, we consider a query-based system to be a randomized function $R : \mathcal{D} \times \mathcal{Q} \to \mathbb{R}$ assigning to a dataset and query pair $(D, q) \in \mathcal{D} \times \mathcal{Q}$ a real-valued random variable $R(D, q)$. The answers provided by the QBS are usually designed to be *similar* to the true answer ($R(D, q) \approx T(D, q)$) but not equal, in order to protect user privacy. We use the more general notion of a random

variable because to preserve the privacy of users, query-based systems commonly implement mechanisms for post-processing that involve randomization (e.g. noise addition). When the answer is fixed, $R(D, q)$ is a deterministic random variable.

Multiple building blocks are typically combined to preserve privacy. These include noise addition [18] with a range of distributions (e.g. Laplace [22], Gaussian [27], uniform [29]), suppression of answers below a threshold (called *query set size restriction*) [19], and restrictions on the set of allowed queries $Q \subset Q'$ or the number of queries that can be meaningfully answered[3] [22].

3 ATTACK MODEL

We propose a general targeted attack against query-based systems, which we call *automated query discovery attack*. The attack is composed of a search mechanism for a multiset of queries and a rule to combine them. While we focus on attribute inference attacks, our approach can be extended to other attacks, e.g., membership inference attacks.

3.1 Attacker access to the query-based system

We assume that the attacker has access to the query-based system protecting the dataset of interest (*target QBS*), in the sense that they can send queries to it. We furthermore assume that the number of queries that can be performed on the target QBS is limited, e.g. to a few hundreds, as queries are typically logged and rate limited.

We also assume that the attacker has black-box access to the QBS software. The QBS software would typically be available freely or for a fee, potentially in compiled form. Alternatively, the attacker might replicate the protection deployed by the target QBS based on public information. Formally, this means that the attacker can retrieve samples from $R(D', q)$ for any query $q \in Q$ and dataset $D' \in \mathcal{D}$. Since the attacker chooses D', they also know the true query answer $T(D', q)$ and can leverage this signal to devise attacks. Note that QBSes are typically initiated with a *seed* for pseudo-random noise generation: we assume that each QBS is initiated with a different seed, and that the attacker does not know the seed used in the target QBS.

3.2 Attribute inference attack

The attacker's goal is to infer the target user's value for one of the attributes (the *sensitive* attribute) $s = r^u_{a^*},\ a^* \in \mathcal{A} \setminus \mathcal{A}'$. We assume that the attacker knows part of the record of a *target user* $u \in U$ consisting of the values for a subset of attributes $\mathcal{A}' \subset \mathcal{A}$. For simplicity, we also assume that the attacker knows (1) that the target is in the dataset ($u \in U$), and (2) that the target's record is unique in the dataset, given all known attributes ($\forall v \in U, v \neq u :\ r^u_{\mathcal{A}'} \neq r^v_{\mathcal{A}'}$).

Formally, the attacker's goal is to devise both a multiset of queries $(q_1, \ldots, q_k) \in Q^k$ and a rule $G : \mathbb{R}^k \to \{0, 1\}$ to combine the answers to the queries $R(D, q_1), \ldots, R(D, q_k)$ to retrieve the sensitive attribute:

$$\hat{s} = G(R(D, q_1), \ldots, R(D, q_k)).$$

The attack is considered successful if the predicted value $\hat{s}$ for the sensitive attribute matches the correct value, i.e., $\hat{s} = s$.

[3]Although this mechanism cannot be expressed formally as $R(D, Q)$, since it requires memory of the number of queries performed. We use this notation for simplicity.

We focus on users that are unique in the dataset D to simplify the evaluation of the privacy gain provided by the QBS alone. An attacker would indeed be able to perfectly infer the sensitive attribute a^* of unique users if the dataset were to be released as-is, allowing us to evaluate the privacy gain provided by the QBS. [4]

3.3 Auxiliary knowledge on the target dataset

We consider two specific attack scenarios, with different assumptions on the attacker's knowledge about the private dataset:

- **AUXILIARY**: the attacker has access to a dataset D' similar to the private dataset D (e.g., from the same distribution).
- **EXACT-BUT-ONE**: the attacker has perfect knowledge of the private dataset D, except for the target record's sensitive attribute $r^u_{a_n}$.

The latter (EXACT-BUT-ONE) is the typical assumption made to evaluate "worst-case" attacks [6, 34], evaluating how much information a very strong attacker would be able to infer. The former (AUXILIARY), on the other hand, is typically used to evaluate privacy risks in practice, i.e., as a more realistic setup [50, 58, 59].

The attacker's knowledge allows them to define a training distribution π_{train} and a validation distribution π_{val} (that can be identical), from which to sample datasets. Datasets generated from the former are used to train the rule G, while datasets from the latter are used to estimate the fitness (test accuracy) of a multiset of queries.

In the AUXILIARY scenario, the attacker generates auxiliary datasets by sampling records uniformly at random without replacement from D' and appending the target record $r^u_{\mathcal{A}'}$ with a random value for the sensitive attribute $r^u_{a_n}$, which defines π_D. In practice, the attacker divides D' in a *training* and a *validation* dataset, which define π_{train} and π_{val}.

In the EXACT-BUT-ONE scenario, auxiliary datasets are obtained by choosing a random value for the target user's sensitive attribute, which defines π_D. In this setup, the training and validation distributions are identical.

Note that in both scenarios, we randomize the target user's sensitive attribute. This breaks possible correlations with known attributes, which could be used to infer the value of the sensitive attribute even if the target user does not contribute their data. Randomization implies that the baseline success rate of an attack without access to data is 50% when the sensitive attribute is binary.

4 ATTACK METHODOLOGY

4.1 Overview of QuerySnout

In this section, we present QuerySnout, our method for automating the discovery of attribute inference attacks against query-based systems. The goal of QuerySnout is to find both a multiset of m queries $q_1, \ldots, q_m \in Q$ and a rule G to combine the answers to these queries to retrieve $r^u_{a_n}$, the target record's value for the sensitive attribute a_n. We here assume, without loss of generality, that the last attribute $a^* = a_n$ is sensitive and that the remaining attributes are known auxiliary information about the target $\mathcal{A}' = (a_1, \ldots, a_{n-1})$. We frame the discovery of attacks against a QBS as an optimization

[4]Note that technically, this property is true for a larger class of users, who are so-called *value-unique* [30]: all records who share the same known attributes have the same sensitive attribute. We focus on unique users for simplicity (who are all value-unique), although the method in this paper applies similarly to value-unique users.

problem over a search space of solutions consisting of all multisets of m queries of a specific structure (Sec. 4.2). QuerySnout optimizes solutions in this space with regards to the accuracy of the attribute inference attack obtained by applying an automatically trained rule G on the answers to this multiset of queries (Sec. 4.3). To optimize for attacks, QuerySnout uses an evolutionary algorithm based on a novel mutation algorithm (Sec. 4.4).

4.2 Search space of solutions

Given a restricted *query search space* $Q_s \subset Q$, we define a *solution* as an unordered list of m queries $q_1 \ldots, q_m \in Q_s$. Formally, solutions are multisets of queries, meaning that the same queries can be repeated multiple times. We denote by $\mathcal{S}^m$ the set of all *solutions* that is explored by the evolutionary algorithm. We describe it formally as follows:

$$\mathcal{S}^m := \Big\{ \{(q^1, m_1), \ldots, (q^k, m_k)\} : q^1, \ldots, q^k \in Q_s^n, \\ i \neq j \implies q_i \neq q_j, \forall i, m_i \in \mathbb{N}, \sum_{i=1}^{k} m_i = m \Big\} \tag{1}$$

where m_i denotes the multiplicity of the i-th unique query.

Note that we here assume that the order of the queries does not impact the way the QBS answers them. All the systems we consider satisfy this property. In the discussion, we explain how our method could be adapted to systems for which query order matters.

Query search space. For general query syntaxes (e.g., SQL), the query space Q is extremely large, even possibly unbounded. In this work, we restrict – as a starting point – the search to a simple, yet still very large *query space* $Q_s \subset Q$ consisting of counting queries that select records via a conjunction of up to n simple conditions, with no more than one condition per attribute.

Given a set of condition operators C_s (such as "equal to" or "different from"), we write $a_i\ c_i\ v_i$ to denote a condition on the i-th attribute of operator $c_i \in C_s$, with value $v_i \in \mathcal{V}_{a_i}$ belonging to set of acceptable values for the i-th attribute. To simplify the search and exploit the information available to the attacker about the target record, we restrict conditions relating to a known attribute to use the target record's value $v_i = r^u_{a_i}, i \in \mathcal{A}'$. A condition on the unknown, sensitive attribute may use any of the acceptable values $v_n \in \mathcal{V}_{a_n}$. Formally, the query search space we consider, Q_s, consists of queries of the form:

$$\begin{aligned} &\text{SELECT COUNT(*)} \\ &\text{WHERE } a_1\ c_1\ r^u_{a_1} \text{ AND } \ldots \text{ AND } a_{n-1}\ c_{n-1}\ r^u_{n-1} \\ &\qquad \text{AND } a_n\ c_n\ v_n \end{aligned} \tag{2}$$

In this work, we set the set of condition operators to $C_s = \{\neq, =, \perp\}$, meaning that conditions can be of operator = (equal to), $\neq$ (different from) or $\perp$ indicating that there is no condition on the i-th attribute. In other words, a condition relating to the i-th attribute can be: (1) $a_i = v_i$, (2) $a_i \neq v_i$ or (3) no condition, which we write as $a_i \perp v_i$ as a convention (the value is ignored); with $v_i \in \mathcal{V}_{a_i}$.

Finally, we assume – for simplicity – the sensitive attribute to be binary: $\mathcal{V}_{a_n} = \{0, 1\}$. Under this assumption, the conditions $a_n = 0$ and $a_n \neq 1$ are equivalent, and there is a one-to-one mapping between the query search space Q_s and ${C_s}^n$, as one choice of condition operators $(c_1, \ldots, c_n) \in C_s^n$ corresponds to exactly one query with operator (2) and conditions $a_1\ c_1\ r^u_{a_1},\ a_2\ c_2\ r^u_{a_2},\ \ldots,\ a_{n-1}\ c_{n-1}\ r^u_{a_{n-1}}$ and $a_n\ c_n\ 0$, respectively. The search algorithm can thus represent queries as strings of k operators.

As an example, let attributes a_1, a_2 and a_3 denote a person's *age*, *nationality* and *diagnosis* for a disease. The query with condition operators $(c_1, c_2, c_3) = (=, \perp, \neq)$ applied to a target record of known age and nationality $(v_1, v_2, v_3) = (19, \textit{Bulgaria}, 0)$ yields the query: "How many people in the database have an age of 19 and a positive diagnosis?" Note that there is no condition on the *nationality* attribute and that the $\neq$ operator on the sensitive attribute is equivalent to selecting users with a positive diagnosis.

We restrict the query space for two main reasons. First, a large number of attacks from previous work can be performed using only queries expressed this way, and we can thus compare our results with manual attacks. For instance, averaging attacks [18], difference attacks [19] and the differential noise-exploitation attack against Diffix [30] can all be written with queries in Q_s. Second, the problem is already computationally challenging, as the size of the attack search space is very large. Indeed, when the sensitive attribute is binary, the cardinality of the query space is $|Q_s| = |C^n| = 3^n$, which increases exponentially with the number of known attributes n. The size of the attack search space $\mathcal{S}^m$ is therefore equal to the number of multisubsets with m elements[5] from a set of size 3^n [26]: $\binom{3^n+m-1}{m} \approx \frac{3^{n(m-1)}}{m!}$, where the approximation holds if $m \ll 3^n$. For typical values such as those used in this paper, e.g., $m = 100$ and $n = 6$, an exact computation yields an extremely large attack search space size of $\approx 1.33 \times 10^{131}$. This emphasizes the importance of being able to search for solutions efficiently.

4.3 Automated learning of a rule G

In order to evaluate whether a solution can be used to perform an attack, we propose a method to automatically learn a rule G to combine the answers to the queries of a solution $\{q_1, \ldots, q_m\} \in \mathcal{S}^m$ to perform an attribute inference attack (Fig. 1). Our method uses the training distribution π_{train} to sample d auxiliary datasets $D'_1, \ldots, D'_d \sim \pi_{\text{train}}$. Each dataset is protected by a QBS initialized with different seeds for their random number generator. We perform the queries on each QBS to obtain the answers $(R(D'_i, q_1), \ldots, R(D'_i, q_m))$, $i = 1, \ldots, d$. We denote by X_{train} the dataset of d samples of m features each (a feature is a query answer) obtained in this way. We also denote by Y_{train} the d values for the target record's sensitive attribute in the corresponding auxiliary datasets. Finally, we train a binary classification model G on $(X_{\text{train}}, y_{\text{train}})$ to infer the value of the sensitive attribute from the answer to queries. The model will be used as the rule G to combine the answers to queries in the solution.

Our approach assumes that an optimal rule G can be effectively approximated using machine learning models. For instance, the averaging attack described in Sec. 5.4 computes a linear combination of the query answers, which can be represented by a logistic regression. Furthermore, multilayer perceptrons are known to be universal approximators [33], which makes them even more apt to model an arbitrary rule. Our approach to learn the rule G automatically given a set of queries is similar to the one proposed by

[5] Note that the size of $\mathcal{S}^m$ is *not* $|Q_s|^m$, as the order does not matter.

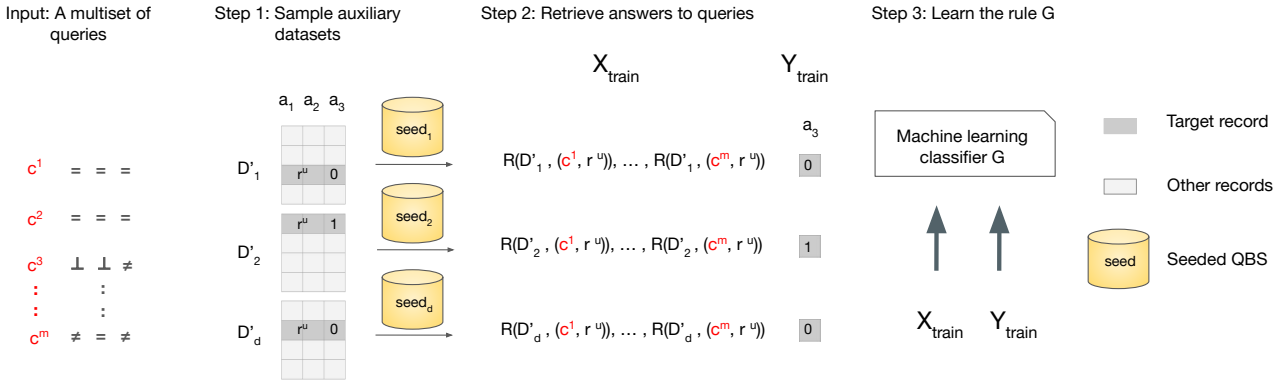

Figure 1: Automated learning of a rule G. We illustrate our approach to automatically combine the answers to a multiset of queries, via a rule G learned using a machine learning model. In our example, partial records r^u consist of values for attributes a_1 and a_2, while a_3 is the sensitive attribute. The attacker samples d auxiliary datasets (randomizing the value of a_3 for the target record), then retrieves $d \times m$ answers to the queries (in red) from (differently seeded) QBSes protecting the datasets. The answers together with the target's sensitive attributes from the corresponding datasets are used to train a classifier G.

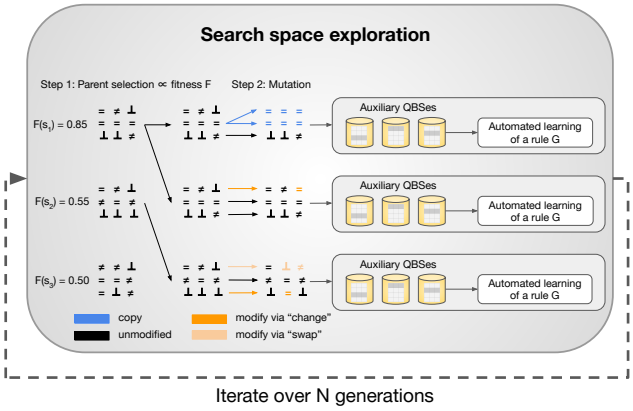

Figure 2: Illustration of how QuerySnout explores the search space using evolutionary algorithms. In this example, the search space consists of multisets of $m = 3$ queries relating to $n = 3$ attributes. Solutions with higher fitness are more likely to be selected. We use different colors to illustrate the changes to a query and black for unmodified query operators.

Pyrgelis et al. to perform membership inference attacks on aggregate statistics [50], except we apply it to attribute inference attacks and use a broader and more flexible range of queries.

4.4 Search space exploration

We explore the search space for queries that can be combined to perform highly accurate attacks using evolutionary algorithms. Evolutionary algorithms are a family of optimization algorithms particularly well suited when the structure of the problem is discrete, unknown or too complex to be described mathematically, and the search space is very large [43]. We refer the reader to the Appendix for a brief introduction to the topic. Our approach (illustrated in Fig. 2) improves a *population* of solutions over time by applying small random changes, called *mutations*, to the solutions. We propose a mutation operator tailored to the task that allows our procedure to efficiently explore the search space. The solutions are optimized with regards to their *fitness*: their ability to infer the target record's sensitive attribute.

Algorithm 1 details our procedure to explore the space of attacks using evolutionary algorithms. We start from a population of P solutions $s_1, \ldots, s_P$. Each solution consists of m queries $s_j = (c^{j1}, \ldots, c^{jm})$, which are initialized uniformly at random: $c^{ji} \leftarrow \mathcal{U}(C^n), i = 1, \ldots, m$. The algorithm runs for N generations. In each generation, we first evaluate the fitness of each of the P solutions. The fitness function, which we describe in the next paragraph, estimates the accuracy of the attribute inference attack obtained with the queries in the solution. Second, we sort the population decreasingly according to the fitness of each solution. Third, we create a new population consisting of (1) a number P_e of *elites* [16], i.e., the P_e solutions in the current population with the highest fitness, and (2) a number $P - P_e$ of *offsprings*. Each offspring is generated by applying a mutation to a *parent* solution.

The parents are selected from the current population by sampling with replacement using the *biased roulette wheel* [57]: the probability to sample a solution is equal to its fitness divided by the sum of fitnesses of solutions in the population. This ensures that solutions with higher fitness are more likely to generate offsprings.

Fitness evaluation. Algorithm 2 in the Appendix describes our procedure to evaluate the fitness. The procedure uses training and validation auxiliary datasets $D'_{\text{aux}} = (D'_{\text{train}}, D'_{\text{val}})$. The datasets in D'_{train} and D'_{val} are (1) sampled upon initialization from π_{train} and π_{val}, respectively, then (2) protected by individual QBS instances with different seeds initializing their random number generator. To evaluate the fitness, we train a rule G to combine the answer of queries to predict the sensitive attribute of the target record, using auxiliary datasets D'_{train}, as described in Sec. 4.3. We measure the

Algorithm 1 QuerySnoutEvolutionarySearch

Inputs:
P: Population size (number of solutions).
m: Number of queries in a solution.
P_e: Number of elites.
N: Number of generations.
D'_{aux}: Auxiliary datasets used to evaluate the fitness.
p_{mut}: Mutation parameters.

Output:
population: A population of P solutions in the search space of solutions $\mathcal{S}^m$.

Initialize:
$population \leftarrow [random\_solution(m)$ for $i = 1$ to $P]$
for $g = 1$ to N **do**
  $fitnesses \leftarrow [\text{EvaluateFitness}(solution, D'_{\text{aux}})$ for $solution$ in $population]$
  // Sort the population by fitness.
  $sort\_descending(population, fitnesses)$
  // Pass the elites unchanged to the next generation.
  $new\_population \leftarrow population[: P_e]$
  for $i = 1$ to $P - P_e$ **do**
    $parent \leftarrow select\_parent(population, fitnesses)$
    $offspring \leftarrow \text{ApplyMutation}(parent, p_{\text{mut}})$
    $new\_population.append(offspring)$
  end for
  $population \leftarrow new\_population$
end for

accuracy of this prediction on training and validation data, a_{train} and a_{val} respectively. We use $\min(a_{\text{train}}, a_{\text{val}})$ as fitness in order to minimize the effect of randomness in the evaluation. Indeed, the accuracy estimates are noisy, and we found empirically that using a_{val} as fitness leads to the algorithm selecting solutions that were *lucky* when estimating the fitness, rather than really superior. The effect was however minor, occurring mostly when the algorithm had converged.

Mutation operator. The mutation operator aims to explore the local space around known good solutions. The mutation operator (we refer the reader to Algorithm 3 in Appendix A.3 for the pseudocode), takes as input a *parent* solution and makes small changes to its queries in order to generate an *offspring* solution. Each query in the parent solution is treated separately, and is either copied with probability p_{copy}, modified in place with probability p_{modify}, or left unmodified with probability $1 - p_{\text{copy}} - p_{\text{modify}}$. Note that copying a query means that we first add the query – as is – to the offspring and then perform the following on a copy of it. If the system is deterministic, we modify the copy; otherwise, with equal probability we either keep the unmodified copy or replace it with a modified version. We add the resulting query to the offspring. We distinguish between the two cases because asking a query repeatedly provides no additional information when the QBS is deterministic. Finally, as the offspring may now contain more than m queries (due to copying), we select a random subset of m queries. This mutation operator is inspired by the fact that many attacks on QBSes [18, 30] use pairs of similar queries, or repeat the same queries multiple times.

Modifying a query. We now describe our procedure to modify a query consisting of n operators $c = (c_1, \ldots, c_n) \in C^n$ (and refer the reader to Algorithm 4 in Appendix A.3 for the pseudocode). For each attribute a_i, we either (1) "change" the corresponding operator c_i with probability p_{change} by replacing it with a different value in the operator set which we sample uniformly at random, (2) swap the operators between the i-th attribute and another attribute that has not been swapped yet with probability p_{swap}, or (3) leave the operator unchanged. To ensure that no attribute pair is more likely to be swapped compared to the others, we randomly permute the order in which this procedure considers the entries of c. We introduce the "swap" operation to exploit symmetry between attributes: if a given attack is successful, it is likely that swapping the conditions of two attributes will lead to a similarly successful attack. Note that although "swap" and "change" can produce the same outcome, they do so with different probabilities and are thus not equivalent.

Choosing the mutation parameters. The mutation parameters are set such that, on average, we copy (modify) a fraction p_{copy} (p_{modify}) of the m queries in a solution, a common heuristic in evolutionary search. The same heuristic also applies to parameters p_{swap} and p_{change}. In practice, the ranges of mutation parameters should be informed by the exploration-exploitation trade-off: making more changes to a solution allows to explore the space more, but making too many changes at once might hinder the improvement of solutions over time.

5 EXPERIMENTAL SETUP

In this section, we first present the datasets we use to evaluate our automated attacks. Second, we present how we instantiate the auxiliary knowledge in the AUXILIARY and EXACT-BUT-ONE scenarios. Third, we describe the query-based systems against which we evaluate our automated attacks. Finally, we describe the system-specific manual attacks we compare against.

5.1 Datasets

We use three publicly available datasets: Adult [52], Census [53] and Insurance [54]. Adult contains 48482 individual records of 14 socio-demographic attributes each. Census contains 299285 records of 41 socio-demographic attributes each. Demographic attributes are, for instance, "age", "education" and "occupation". We assign "income" as the binary sensitive attribute for both Adult and Census and randomize it (50:50) as described below. Insurance contains 9822 records of 86 attributes about customers of a car insurance company, aggregated at the zipcode level. We only use the 43 socio-demographic attributes and add a randomized (50:50) sensitive attribute.

5.2 Instantiating the auxiliary knowledge

For each dataset, in each repetition of the experiments, we sample $n - 1 = 5$ attributes uniformly at random without replacement and discard the others. These are the attributes $\mathcal{A}'$ whose values for a target record we assume to be known to the attacker. We then randomly partition the dataset between a training auxiliary dataset D_{train}, a validation auxiliary dataset D_{val}, and a testing dataset

D_{test} of equal sizes. We select 100 unique *target records* from D_{test} uniformly at random without replacement. For each target record, we instantiate the training and validation distributions π_{train} and π_{val}. We also instantiate a test distribution π_{test} from which *private* datasets D containing the target record will be sampled in two steps.

- In the AUXILIARY scenario, for split $\in$ {train, val, test}, we instantiate π_{split} on the corresponding dataset D_{split} as described in Sec. 3.3. We also add a randomized sensitive attribute (50:50) to datasets sampled from π_{split}, and remove duplicates of the target record from all datasets generated to ensure that the it is unique.
- In the EXACT-BUT-ONE scenario, we sample one dataset D from D_{test} and add a randomized sensitive attribute (50:50). We then instantiate π_{train}, π_{val} and π_{test} on D as described in Sec. 3.3.

To ensure that information from the test distribution does not unintendedly leak into the train and validation distributions, the seeds used to instantiate the QBSes on datasets sampled from π_{train}, π_{val} and π_{test} are *all distinct*. To evaluate the actual privacy leakage of the query-based system and in line with previous work [30], we choose to break the correlations between the sensitive attribute and the other attributes by randomizing the binary sensitive attribute with a 50:50 distribution.

5.3 Query-based systems

We describe our implementation of three query-based systems. The first two are based on real-world privacy protection mechanisms used by Diffix [27] and TableBuilder [29], while the third one is a non-deterministic system combining two QBS building blocks: query set size restriction and (Gaussian) noise addition.

These systems all rely on *query set size restriction* (QSSR), where the QBS refuses to answer a query if it concerns less than T users, for some threshold T. QSSR aims at preventing queries that reveal information about a small number of users, such as counting the number of record identical to the target user and either value of s. We here assume that a QBS doing bucket suppression returns 0 instead of an error message, as this makes the attack harder by giving less information to the attacker. For simplicity, we introduce the following notation for bucket suppression: $\beta_\tau(D,Q) = I\{T(D,Q) > \tau\}$, where I is the indicator function. We also introduce the notion of *query set* $U(D,Q)$, the set of all users who satisfy the conditions of the query Q in the dataset D. Note that for counting queries, $T(D,Q) = |U(D,Q)|$.

Rounding. For realism, since we focus on counting queries, we further assume that the answers obtained from all mechanisms are then rounded to the nearest integer, and that if the mechanism would return a negative answer it returns 0 instead. This again makes the attack harder by giving less information to the attacker. When applying our attack on a mechanism, we thus use R_{qbs} defined as:

$$R_{\text{qbs}}(Q,D) = \lfloor \max(R_{\text{mechanism}}(Q,D), 0) \rceil$$

Randomness. Both Diffix and TableBuilder rely on *seeded* noise, i.e., noise produced by a pseudo-random number generator (PRNG) which outputs the same noise given the same input seed. Hence, these two systems are deterministic: for the same input query, they always output the same result. On the contrary, SimpleQBS is non-deterministic, and sample fresh random variables for each query.

Diffix. Diffix is a commercial QBS developed by the startup Aircloak [27]. It uses bucket suppression with a noisy threshold and two layers of so-called *static* and *dynamic* noise addition. All noises are seeded with different elements of the query, in order to prevent specific attacks. Specifically, we attack Diffix-Birch [28], which adds noise to counting queries as:

$$R_{\text{diffix}}(D,Q) = \beta_{\min(2,\tau)}(D,Q)\left(T(D,Q) + \sum_{i=1}^{n_{\text{cond}}(Q)} N_i^S + N_i^D\right)$$

where $n_{\text{cond}}(Q)$ is the number of non-empty conditions in Q (i.e., $\sum_{i=1}^{n} I\{o_i \neq \perp\}$), $\tau \sim \mathcal{N}(4, 0.5)$ is seeded with the query set, $N_i^S \sim \mathcal{N}(0,1)$ is seeded with the text of condition i of Q, and $N_i^D \sim \mathcal{N}(0,1)$ is seeded with the text of condition i of Q and the query set $U(D,Q)$.

TableBuilder. TableBuilder is a QBS developed by the Australian Bureau of Statistics for census contingency tables [29]. We here focus on the privacy mechanism used on individual cells (similarly to [3]), and hence attack a simplified subset of the syntax of the real system. We define this mechanism as

$$R_{\text{tablebuilder}}(D,Q) = \beta_4(D,Q) \cdot (T(D,Q) + U),$$

where $U \sim \mathcal{U}\{-2,\ldots,2\}$ is seeded with the query set $U(D,Q)$.

SimpleQBS. We study a simple QBS that combines two common building blocks: Gaussian noise addition (with variance σ) and query set size restriction. Formally, the mechanism SimpleQBS(τ,σ^2) answers queries as:

$$R_{\text{simple}}(D,Q) = \beta_\tau(D,Q) \cdot (T(D,Q) + N),$$

with $N \sim \mathcal{N}(0,\sigma^2)$. A fresh noise sample is drawn every time a query is performed, even if the query is repeated. We instantiate this QBS with all pairs (τ,σ), $\tau = 0,\ldots,4$ and $\sigma = 0,\ldots,4$.

Note that SimpleQBS$(\tau, \sigma > 0)$ can be seen as an application of the Gaussian mechanism with sensitivity $\max(1,\tau)$ [23], and thus provides $(\varepsilon,\delta)-$differential privacy for some values of ε,δ. However, for our choice of parameters, and since we do not restrict the number of queries, the corresponding values of ε,δ are very large and do not represent meaningful privacy guarantees. In Sec. 7, we propose an extension of our method to handle budget-based mechanisms.

5.4 Manual attribute inference attacks

We here describe manual attribute inference attacks from prior work against Diffix, TableBuilder, and SimpleQBS. We compare QuerySnout with these attacks both quantitatively (i.e., the accuracy of the inference) and qualitatively (i.e., what is the attack exploiting). We first introduce the notion of *difference attack*, which underlies most of the attacks we consider here.

Difference attacks. A common class of attacks against QBSes is *difference attacks*, which consist of a pair of queries (q_1, q_2) of the form

$$q_1 = \text{COUNT WHERE } \textstyle\bigwedge_{i\in A'}(a_i = r_{a_i}^u) \wedge a_n = s$$
$$q_2 = q_1 \wedge a_{i'} \neq r_{a_{i'}}^u$$

where $A' \subset \{1, \dots, n-1\}$ is a subset of attributes, $i' \notin A', i' \neq n$ is another attribute, and $s \in \{0, 1\}$ is a possible value for the sensitive attribute. The insight behind such attacks is that if the target user is uniquely identified by (A', i'), then the true counts of q_1 and q_2 differ by 1 if and only if $r^u_{a_n} = s$, and 0 otherwise. The rule to combine the answers to q_1 and q_2, as well as how to select the parameters (A', i', s) depend on the system.

Diffix. Gadotti et al. [30] proposed the only known attribute inference attack against Diffix. This attack uses difference queries as described above, and exploits the structure of Diffix's noise to combine results. Specifically, if the target record is unique for $A' \cup \{i'\}$, then the distribution of $R(q_1, D) - R(q_2, D)$ is either $\mathcal{N}(0, 2)$ if $r^u_{a_n} = 1$ or $\mathcal{N}(1, 2|A'| + 2)$ otherwise. The attack uses a likelihood ratio test to distinguish between the cases. In order to find values of (A', i', s) such that the target record is unique and the queries are not suppressed, the attack performs a search over subsets of attributes using access to the target QBS (protecting the private dataset D) and heuristics to determine whether the assumptions are verified. We implement their attack with the ValueUnique [30] heuristic in the AUXILIARY setup, and the exact uniqueness oracle in the EXACT-BUT-ONE setup to reflect the additional knowledge of the attacker. Note that this attack is *interactive*, as it uses iterative access to the target QBS to choose which queries to perform.

TableBuilder. Chipperfield et al.[11] propose a simple difference attack against TableBuilder. For each known attribute j and value $s \in \{0, 1\}$, they perform the difference queries $(q_1^{j,s}, q_2^{j,s})$ for $A' = \{1, \dots, n-1\} \setminus \{j\}$ and $i' = j$. They then compute the difference $r^{j,s} = R(q_1^{j,s}, D) - R(q_2^{j,s}, D)$, and observe that if $r^{j,s} \geq 5$, the true difference must be 1 (because the noise is bounded), and the attack predicts that the user's target value is $1 - s$. If $r^{j,s} < 5$, $\forall j, s$, the attacks computes the averages $\mu_s = \frac{1}{n-1} \sum_{j=1}^{n-1} r_{j,s}$ and predicts 0 iff $\mu_0 > \mu_1$ (because the noise is centered). Rinott et al. [56] propose another attack, based on the same queries but using a different combination rule. This second attack exploits the fact the noise added to a query only depends on the userset, and the same noise is thus added to all the queries selecting the same userset. Then, if the answer to both queries in a pair are equal (i.e., $r^{j,s} = 0$), the attack predicts $1 - s$ (since the target user is not in the user set of either query, $r^u_{a_n} \neq s$). Our work is, to the best of our knowledge, the first to empirically evaluate this attack, since the original paper only presented it as a theoretical vulnerability. We implement both attacks with a small modification to only take into account non-suppressed queries. Additionally, we adapt the attacks to the EXACT-BUT-ONE scenario to reflect the additional knowledge of the attacker, by selecting attribute subsets for which the target record is unique (since they have perfect knowledge of D).

SimpleQBS. Non-deterministic noise addition and simple bucket suppression have long been known to be vulnerable to respectively averaging and difference attacks [18, 19].

When $\tau = 0$, SimpleQBS is vulnerable to a simple averaging attack, where the query $q_{\text{direct}} = \bigwedge_{i=1}^{n-1}(a_i = r^u_{a_i}) \wedge a_n = 0$ is repeated m times to obtain the results $(r_1, \dots, r_m)$. Since the noise is centered, $\mathbb{E}[R_i] = 1$ (resp. 0) if and only if $r^u_{a_n} = 0$ (resp. 1). The attacker predicts 0 iff $\frac{1}{m} \sum_{i=1}^{m} r_i < \frac{1}{2}$, and 1 otherwise.

When $\tau > 0$, the query q_{direct} is suppressed by bucket suppression. We thus combine a difference and an averaging attack. We use the attacker's auxiliary information to find values of (A', i', s) such that the user is unique for (A', i') and the queries bypass bucket suppression in two steps. First, we generate auxiliary datasets (from the auxiliary knowledge available in each scenario). Second, we select the pairs (q_1, q_2) for which both assumptions are satisfied for the largest fraction of datasets. When $\sigma > 0$, we repeat the best pair of queries $m/2$ times, averaging the results to obtain μ_{q_1} and μ_{q_2}. We then output s iff $\mu_{q_1} - \mu_{q_2} > \frac{1}{2}$, and $1 - s$ otherwise. When $\sigma = 0$, we select the $m/2$ best pairs and perform them in decreasing order until the result (r_1, r_2) is such that $r_1 > 0, r_2 > 0$ and $r_1 - r_2 \in \{0, 1\}$. We output s iff $r_1 = r_2 + 1$, and $1 - s$ otherwise.

6 EMPIRICAL RESULTS

The goal of our empirical evaluation is fourfold. First, we want to show that it is possible to automate the discovery of attribute inference attacks against a QBS by "pressing a button". Second, we want to understand quantitatively how well the attacks discovered by QuerySnout perform when compared to manual attacks. Third, we want to understand qualitatively what vulnerabilities are exploited by QuerySnout, for instance if they are similar to known attacks. Finally, we want to showcase the versatility of QuerySnout by deploying it on a variety of attack scenarios to derive new insights.

6.1 Attack parameters

For each target record, we run the evolutionary search and extract after N generations the solution s^* having the highest fitness. For computational efficiency and as the attacks against different records are independent, we parallelize the attack over the target records. We use 2000 training datasets (sampled from π_{train}), 1000 validation datasets (sampled from π_{val}), and 500 test datasets (sampled from π_{test}). The datasets are of size 8000 for Adults and Census and 1000 for Insurance.

Evolutionary search parameters. Each evolutionary search uses solutions of $m = 100$ queries, a population size of $P = 100$ and a maximum number of $N = 200$ generations with a stopping criteria of having 10 generations of fitness superior to 99.99%. To evaluate the fitness of a solution, we use a Logistic Regression as the binary classifier G. We also experimented with a multilayer perceptron and found that it did not improve the performance, while significantly increasing the training time. We set the mutation parameters for modifying operators in a query to $p_{\text{change}} = \frac{1}{n} = \frac{1}{6}$, so that on average we change one operator. Similarly, we use $p_{\text{swap}} = \frac{1}{6}$, $p_{\text{copy}} = 0.025$ and $p_{\text{modify}} = 0.025$.

Attack success metric. For each target record, we compute the accuracy of the attribute inference attack defined by the best solution s^* and the trained classifier G on 500 datasets sampled from the corresponding test distribution $D \sim \pi_{\text{test}}$. Our metric for the attack success is the average accuracy over the 100 target users, which we report averaged over 5 repetitions. Note that we randomize the private attribute a_n, so that the random guess baseline has an accuracy of 50%.

6.2 Real-world systems

Our results show that the attacks discovered automatically by QuerySnout match and often outperform by a large margin the manual attacks from previous work.

AUXILIARY scenario. Table 1a and Table 2a show how, across all datasets and systems, QuerySnout matches or outperforms manual attacks. More specifically, it matches the accuracy of manual attack by Gadotti et al. [30] against Diffix on the Adult and Census datasets and strongly outperforms it on the Insurance dataset. Similarly, QuerySnout strongly outperforms Chipperfield et al. [11] on all datasets and matches Rinott et al. [56] on Adult and Census, while strongly outperforming it on Insurance. **Our results show how automated attacks, and QuerySnout in particular, can today not only replicate but also outperform existing manual attacks.** We identify two reasons for the good performances of QuerySnout.

First, a manual inspection of queries found by QuerySnout on Diffix shows that difference queries account for $\geq$ 97.0% of the accuracy, while constituting less than half of the queries in solutions (see Appendix A.4 for a detailed analysis). Similarly, the difference queries found by QuerySnout on TableBuilder also account for $\geq$ 97.5% of the accuracy. QuerySnout is able to outperform the manual attacks because it is able to (1) find more attribute subsets for which the target record is likely unique, and (2) combine the results from all subsets taken together for the attack. We report examples of the solutions found by QuerySnout for each QBS in Appendix A.11.

Second, the gap between QuerySnout and manual attacks is particularly strong on the Insurance dataset, for both QBSes. This is due to previous attacks performing worse on Insurance than on Adult and Census and to QuerySnout performing better. We hypothesize that this difference is due to the smaller size of the Insurance dataset ($|D| = 1000$ for Insurance and $|D| = 8000$ for Adult and Census), making queries more likely to be suppressed by the query set size restriction (QSSR). While our automated attacks also exploit difference attacks, we believe its flexibility allows it to find more specific queries that bypass QSSR. In Appendix A.7, we show that QuerySnout performs slightly better in general on smaller datasets.

EXACT-BUT-ONE scenario. Table 1a and Table 2a show how the accuracy of the best attack found by QuerySnout increases by at least 10% when moving from the AUXILIARY to the EXACT-BUT-ONE scenarios across datasets and QBSes. More specifically, the accuracy of QuerySnout increases by 12.4% (resp. 10.1% and 11.5%) for Adults (resp. Census and Insurance) compared to AUXILIARY against Diffix and by 13.7% (resp. 11.2% and 13.4%) against TableBuilder. It also strongly outperforms the baselines across all datasets and QBSes, with the exception of Rinott et al. on Census where it matches them. **These results show that much stronger attacks than previously believed can be devised against both Diffix and TableBuilder by a strong ("worst-case") attacker. They also show how QuerySnout is able to automatically discover new and better attacks across scenarios and datasets.**

For instance, in the case of Diffix, an attacker could perform the query $a_1 = r^u_{a_1} \wedge a_n = 0$. Denote by C the true count of this query when $r^u_{a_n} \neq 0$. The answer returned by Diffix is distributed as $\mathcal{N}(C + I\{r^u_{a_n} = 0\}, 4)$, assuming the query is thus not bucket suppressed (which will occur only if $r^u_{a_1}$ is very rare). Repeating this query for different attributes $a_2, \ldots, a_{n-1}$ thus gives n samples to distinguish between (in effect) $\mathcal{N}(1, 4)$ and $\mathcal{N}(0, 4)$. For $n = 5$, a likelihood ratio test distinguishing between these distributions can already achieve an accuracy of $\approx$ 73%.

Table 1: Accuracy against Diffix in the (a) AUXILIARY and (b) EXACT-BUT-ONE scenarios. We report the mean and standard deviation over 5 repetitions.

(a) AUXILIARY	Adult	Census	Insurance
QuerySnout (automated)	**77.8** (0.5)	**78.3** (1.4)	**80.1** (0.6)
Gadotti et al. [30] (manual)	76.3 (0.8)	76.9 (1.4)	73.0 (1.2)
(b) EXACT-BUT-ONE	Adult	Census	Insurance
QuerySnout (automated)	**90.2** (0.6)	**88.3** (0.9)	**91.6** (1.2)
Gadotti et al. [30] (manual)	77.1 (0.9)	77.5 (2.0)	74.4 (0.7)

Table 2: Accuracy against TableBuilder in the (a) AUXILIARY and (b) EXACT-BUT-ONE scenarios. We report the mean and standard deviation over 5 repetitions.

(a) AUXILIARY	Adult	Census	Insurance
QuerySnout (automated)	**84.5** (0.6)	**85.5** (1.4)	**85.4** (0.6)
Rinott et al.[56] (manual)	76.1 (7.5)	78.1 (7.0)	56.9 (4.6)
Chip. et al.[11] (manual)	61.2 (3.5)	62.4 (3.1)	52.8 (1.8)
(b) EXACT-BUT-ONE	Adults	Census	Insurance
QuerySnout (automated)	**98.1** (0.7)	**96.6** (0.9)	**98.8** (0.7)
Rinott et al.[56] (manual)	83.1 (8.7)	72.1 (13.4)	76.5 (2.3)
Chip. et al.[11] (manual)	72.3 (6.4)	64.4 (7.2)	67.2 (1.4)

6.3 Non-deterministic systems: SimpleQBS

Most real-world systems are deterministic, either through seeded noise or caching, ensuring that the same answer is returned every time the same query is sent. This is desirable from both a privacy perspective (preventing averaging attacks) and a utility perspective (always returning the same answer). In some cases though, systems might be non-deterministic. This can be by design, e.g., lack of awareness of averaging attacks; because of implementation or operational issues, e.g., non-functional cache or answer retrieving mechanism; or in cases where the same dataset is made available from different instances of the QBS seeded differently, e.g., distributed computing.

We here show how QuerySnout is able to find efficient attacks against non-deterministic systems. While we here show and discuss results against the Adult dataset, our conclusions hold for Census and Insurance (cf. Fig. 9 in Appendix A.5). We also compare our results to the baseline inspired by Denning et al. [19] and using a simple heuristic to choose difference queries from the auxiliary information available to the attacker (see Sec. 5.4 for details).

Fig. 3 shows how QuerySnout manages to find effective attacks against SimpleQBS across a range of σ (noise) and τ (threshold) outperforming, often strongly, manual attacks. More specifically, QuerySnout outperforms –on average– manual attacks in the AUXILIARY scenario by 3.8%, particularly when the noise added is small

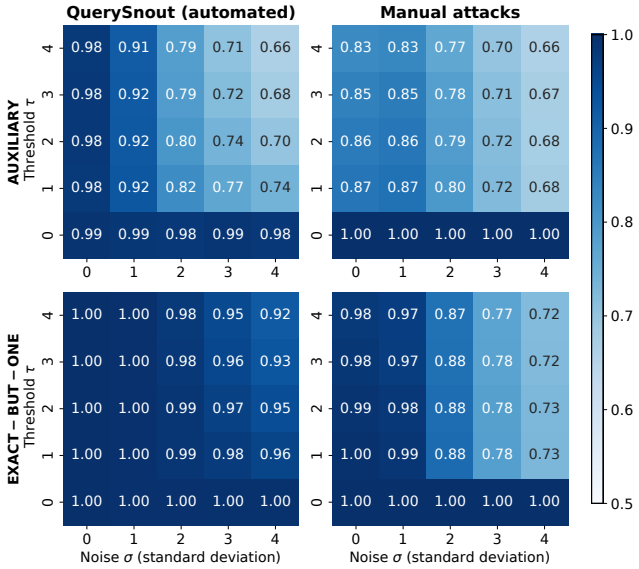

Figure 3: Comparison between automated and manual attacks against SimpleQBS(τ, σ) on the Adult dataset. We report the accuracy of attribute inference attacks discovered by QuerySnout (left) and of manual attacks tailored to each system (right) in the AUXILIARY (top) and EXACT-BUT-ONE (bottom) scenarios. The accuracy is averaged over 5 repetitions.

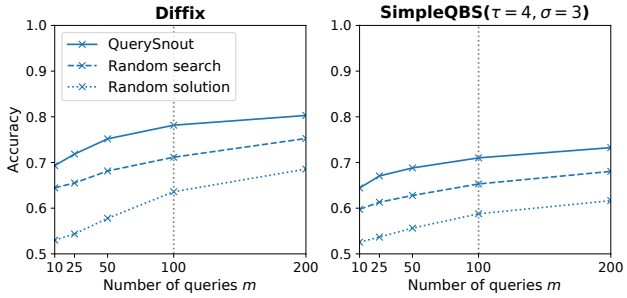

Figure 4: Impact of the number of queries on the accuracy of QuerySnout. We report the accuracy of QuerySnout (which uses an evolutionary search), of a random search, and of a solution chosen uniformly at random as we vary the number of queries m. The results are computed on the Adult dataset in the AUXILIARY scenario.

(σ from 0 to 3). The gap in accuracy is even larger (8.4% on average) in the EXACT-BUT-ONE scenario, finding attacks with > 90% accuracy across all values of (τ, σ) considered. Noticeably, QuerySnout finds much better attacks, by 16.7% on average, than the manual ones for high values of σ (from 2 to 4) and $\tau \geq 1$.

6.4 Comparison with random search

To evaluate the impact of our search procedure, we compare our results with the accuracy obtained by a random search. The random search uses the same fitness evaluation but samples a new population of random solutions at every generation rather than mutating and keeping previous solutions. The best solution found over all generations is used. We provide more details on this baseline in Appendix A.6. We here compare our approach to random search for Diffix, Table Builder, and two selected version of SimpleQBS: ($\tau = 4, \sigma = 3$) and ($\tau = 3, \sigma = 4$). We find that our approach strongly improves upon a random search procedure using the same parameters, across all datasets and QBSes (see Table 4 in the Appendix). For instance, in the AUXILIARY scenario, our approach outperforms the random search by 6.3% for Diffix, 5.3% for TableBuilder, 6.7% for SimpleQBS($\tau = 4, \sigma = 3$), and 5.4% for SimpleQBS($\tau = 3, \sigma = 4$), each time averaged over the datasets.

6.5 Impact of the number of queries

Limiting the number of queries is, along with authentication, a popular "non-QBS" defense used in practice. This can be done directly, often per user and per period of time, or through budgeting [42]. We here evaluate the impact of limiting or increasing the number of queries made by our attack to the target QBS. For simplicity, we report results on a deterministic QBS (Diffix) and a non-deterministic QBS (SimpleQBS($\tau = 4, \sigma = 3$)) on the Adult dataset in the AUXILIARY scenario. The τ and σ of SimpleQBS were chosen to be comparable to those of Diffix, same threshold mean and similar average noise per query (for queries with 5 conditions).

Fig. 4 shows that, as expected, the performance of QuerySnout increases with the number of queries m, albeit slowly. Indeed, even with as little as 10 queries, QuerySnout still reaches an accuracy of 69.3% against Diffix and 64.4% against SimpleQBS(4, 3). Increasing the number of queries would further increase the accuracy of the attack, e.g., against systems that do no implement some kind of query limiting mechanisms.

We also evaluate the impact of m on both random search and a naïve baseline consisting of one solution sampled uniformly at random from the search space of solutions $\mathcal{S}^m$. Our results show that QuerySnout's mutation operators enable it to consistently strongly outperform the random search, which itself –as expected– outperforms a random solution. It is however interesting to note that, when the number of queries is large, even a single random solution achieves better than random accuracy.

6.6 Impact of search parameters

We study the impact of different components on the performance of QuerySnout. Fig. 11 in the Appendix shows that increasing the size of the population P, the number of generations N, or the number of datasets (auxiliary QBSes) results in better performance, with decreasing returns. However, the marginal increase in performance comes at a computational cost, as doubling any of the parameters will double the running time. In some cases, the memory footprint also increases significantly, e.g., increasing linearly with the number of auxiliary datasets.

For the mutation operators, Table 6 in the Appendix shows "copy" to be the most impactful one. For non-deterministic systems, "copy" duplicates a query and – with equal probability – either keeps the

duplicate unchanged or modifies it with p_{change} and p_{swap}. For deterministic systems, "copy" duplicates a query and always modifies it, since repeating the same query would yield the same answer. In both cases, removing this component negatively impacts the evolutionary search by a significant margin. Even though queries are still being modified (according to p_{modify}), it becomes harder for QuerySnout to find difference queries or to repeat good queries many times. As for the mutations of query operators, the search is robust to removing either "swap" or "change", as there is significant overlap between the two mutations. We further hypothesize that "swap" is likely to be more useful when increasing the number of condition operators $|C_s|$ (e.g., allowing for $<, \leq, \geq, >$ operators).

7 BUDGET-BASED SYSTEMS

Some QBSes, mostly based on ε–differential privacy [22], require the user to divide a given privacy budget ε between the queries they ask. We here present a simple heuristic to apply QuerySnout to QBSes that require a budget, which we call Budget-Based Systems (BBS).

7.1 Extending QuerySnout to BBSes

Formally, we model budget-based systems similarly to other QBSes, with the addition of a parameter called the *partial budget* of the query, specifying the fraction of the total budget (ε) to use on this query: $R : \mathcal{D} \times \mathcal{Q} \times (0, 1]$. A new query is only answered if the sum of partial budgets used and of the new query's partial budget is less than or equal to 1.

We here propose a simple heuristic to transform the multiset of queries manipulated by QuerySnout to a list of pairs of query and partial budget. Given a solution of m queries $s = (c^1, \ldots, c^m)$ we group identical queries to obtain k unique queries with multiplicities $(c^{i_1}, m_1), \ldots, (c^{i_k}, m_k)$ such that $\sum_{i=1}^{k} m_i = m$. We allocate to each unique query a partial budget proportional to its multiplicity. This way, we perform k queries to the QBS, where the partial budget allocated to the i-th query is equal to $\frac{m_i}{m}$. This heuristic assumes *monotonicity of accuracy*: performing any query q with full budget yields more accurate results than repeating a query k times with a partial budget of $1/k$ and averaging out the results. We discuss this in Appendix A.9.

Note that while running the evolutionary search, we are performing the queries on QBSes instantiated on the auxiliary datasets (resetting their budget at each iteration), and only the final $k \leq m$ queries will be performed on the target QBS. Hence the budget of the target QBS is only used once.

7.2 DPLaplace

We consider a simple budget-based mechanism using addition of Laplace noise. The mechanism assumes that a total budget ε is allocated by the data curator, and that each query is answered by adding independent Laplace noise scaled with $(p\varepsilon)^{-1}$, where p is the fractional budget allocated for this query. This ensures that each answer satisfies $(p\varepsilon)$–Differential Privacy (DP) [22] and, by composition [42], performing k queries with fractional budgets $(p_1, \ldots, p_k)$ satisfies $(\sum_{i=1}^{k} p_i\varepsilon)$–DP. Formally, having answered $k-1$ queries with total fractional budget $p_{1:k-1}$, the QBS answers the k^{th} query with fractional budget p_k iff $p_{1:k-1} + p_k \leq 1$, and answers as:

$$R_{\text{DPLaplace}(\varepsilon)}(D, Q, p) = T(D, Q) + L \text{ with } L \sim Lap\left((p\,\varepsilon)^{-1}\right)$$

Similarly to the other systems, we round answers to the nearest integer and threshold the answers at 0, which doesn't affect the privacy guarantees. We prove in Appendix A.9 that this QBS satisfies the monotonicity of accuracy assumption. Although the DPLaplace mechanism is particularly simple, it serves as the core component of many more complex systems implementing Differential Privacy, such as PINQ [42], Chorus [45] and PriPearl [37]. We instantiate this QBS for $\varepsilon \in \{1, 5, 10\}$. As we explain below, finding solutions for our search procedure is challenging against DPLaplace.

Known optimal attack. A known *optimal* attack exists against DPLaplace: a uniqueness attack that performs the query q = COUNT WHERE $\bigwedge_{i=0}^{n-1}(a_i = r^u_{a_i}) \wedge a_n = 0$ with partial budget 1 and returns $s = 0$ iff the count is larger than 0.5. We prove that this attack achieves maximal accuracy in Appendix A.10. Given a number of queries m, this attack can be mapped to the solution in $\mathcal{S}^m$ consisting of the query q repeated m times. While the attack is simple for a knowledgeable attacker, it present challenges for our search procedure. Showing that QuerySnout can be extended to budget-based mechanisms is therefore important.

Finding good solutions with our current black-box search is challenging here, particularly for small values of ε. Indeed, a query that is not repeated (equivalently, that has a small partial budget) will be answered with a noise of large standard deviation ≈ 14 for $\varepsilon = 1$ and $m = 10$. This has two consequences: first, estimating the fitness of solutions is difficult (due to the increased randomness), which can lead to overfitting, both of the rule G and the fitness; second, it complicates the discovery of good queries, since the signal of optimal queries is significantly smaller than the noise added. For instance, even the optimal query $\bigwedge_{i=1}^{n-1}(a_i = r^u_{a_i}) \wedge s = 0$ has little signal if not repeated, since performing it once (with budget $\frac{1}{10}$) leads to an attack with accuracy of at most $\approx 52.4\%$.

Attack parameters. Unlike the previous mechanisms, increasing the number of queries in a solution m amounts to adding more noise to the answers. The standard deviation of the noise is indeed proportional to m, due to the budget being split among the queries (and is equal to $\sqrt{2}m/\varepsilon$ for queries with multiplicity 1). We thus use smaller solutions of $m = 10$ queries. All other parameters are identical as those in Sec. 6, except we update the mutation operators accordingly to $p_{\text{copy}} = p_{\text{modify}} = \frac{1}{m} = 0.1$.

7.3 Empirical results

Table 3 shows that the attacks discovered by QuerySnout match the performance of the optimal attack for large budget values $\varepsilon \in \{5, 10\}$ but fall slightly short in the $\varepsilon = 1$ case. Indeed, in the latter case QuerySnout performs on average 6.4% worse than the optimal attack for AUXILIARY, and 4.1% for EXACT-BUT-ONE.

These results show that QuerySnout can already find the optimal attacks against budget-based mechanisms for medium to small amounts of noise, and very good ones when a large amount of noise is added. We believe the search can be further improved in future work.

In Appendix A.6, we report the difference in accuracy between QuerySnout and the random search for DPLaplace. Table 5 shows

Table 3: Accuracy against DPLaplace for different values of ε in the AUXILIARY and EXACT-BUT-ONE scenarios. We report the mean and standard deviation over 5 repetitions.

	DPLaplace($\varepsilon = 1$)			**DPLaplace($\varepsilon = 5$)**			**DPLaplace($\varepsilon = 10$)**		
AUXILIARY	Adult	Census	Insurance	Adult	Census	Insurance	Adult	Census	Insurance
QuerySnout (automated)	62.9 (0.8)	64.2 (0.8)	63.6 (0.8)	94.2 (1.0)	94.6 (1.0)	94.9 (0.7)	98.6 (0.4)	99.4 (0.2)	99.0 (0.3)
Uniqueness attack (manual)	69.2 (0.8)	70.2 (0.9)	70.4 (1.2)	95.6 (0.8)	95.5 (0.7)	95.9 (0.6)	99.6 (0.2)	99.6 (0.2)	99.7 (0.1)
EXACT-BUT-ONE	Adult	Census	Insurance	Adult	Census	Insurance	Adult	Census	Insurance
QuerySnout (automated)	64.2 (0.7)	65.0 (0.6)	65.6 (0.8)	95.4 (0.8)	95.3 (0.7)	95.7 (0.7)	99.5 (0.3)	99.6 (0.1)	99.7 (0.1)
Uniqueness attack (manual)	68.6 (1.1)	68.9 (2.6)	69.5 (2.6)	95.7 (1.1)	95.6 (1.1)	96.0 (0.6)	99.5 (0.3)	99.5 (0.4)	99.8 (0.2)

QuerySnout to vastly outperform the random search for all datasets, scenarios, and values of ε. Specifically, in the AUXILIARY scenario, the gap is of 11.5% on average across the datasets for $\varepsilon = 1$, with the random search (at 52.1%) barely improving on the random guess baseline. Similar or higher gaps (up to 29.3%) are obtained for other datasets, scenarios, and values of ε. This result both confirms that finding good solutions for DPLaplace is challenging, and shows that QuerySnout is able to efficiently explore the search space even in complex cases.

Overall, the gap between QuerySnout and the random search is much larger for DPLaplace compared to the other QBSes considered in this paper. This is due to (1) random solutions yielding low accuracy against DPLaplace (as explained above) and (2) the noise added by the other QBSes to answers on one query being overall smaller ($\leq \sqrt{12}$ for Diffix and $\sigma \leq 5$ for SimpleQBS, compared to $10\sqrt{2}$ for DPLaplace($\varepsilon = 1$)) and independent of the number of queries.

8 DISCUSSION

QuerySnout *automatically* finds attacks that equate and often outperform previous manual attacks across a range of systems and datasets. This demonstrates the potential of automating attacks against QBSes and opens the door for new, much more complex attacks to be discovered. Our work is the first to enable systems to be automatically tested before deployment to identify and patch vulnerabilities. We now discuss in more detail the extensive potential avenues for future work.

Extensions. QuerySnout currently operates in the same constrained space explored by manual attacks. More complex and accurate attacks exploiting e.g. different SQL operators ($\leq, \geq$) or logical expressions (such as OR) are, in our opinion, likely to exist. Finding such attacks however presents some challenges as the size of the search space increases fast with new operators or conditions per attributes, and with it the computational cost of the search. Specialised mutation operators can be used to overcome this challenge, for instance, using empirical discrete gradient approximation [21]. Furthermore, generative encodings [60] can be used to improve the expressiveness of the genotype and enhance the capability of QuerySnout to find regularities in query sets.

QuerySnout currently relies on a set of assumptions, primarily that the target record is unique in the private dataset and that the attacker has access to auxiliary knowledge on the target dataset. We believe both can be relaxed. In particular, different auxiliary information can likely be used, for instance a few statistics on the data from which synthetic data is generated [58], or syntactically similar datasets from another distribution. These would not require changing the method as QuerySnout only requires two samplers π_{train} and π_{val}.

The attacks that we consider here are *non-interactive*: the attacker chooses which queries to perform *before* observing answers from the target QBS. It is likely that attacks could be made more accurate or efficient if future automated attacks were to use the answer to previous queries to inform the choice of the next queries [30].

Finally, QuerySnout could be extended to use additional QBS outputs such as the time it took for the answer to a query to be computed (such as exploited by Boenisch et al. [10]), information about the query provided by the QBS, or warnings and error messages produced by the QBS (e.g., allowing our system to treat bucket suppression as different from a 0). This would enable the system to discover so-called *side-channel attacks* in addition to the privacy attacks that this paper focuses on.

Membership inference attacks. QuerySnout currently focuses on attribute inference attacks. Another class of attacks, membership inference attacks (MIA), has been considered in the privacy literature. MIAs are at the core of formal privacy guarantees such as differential privacy [22] and can lead to privacy concerns when being part of a dataset is sensitive. More importantly here, membership inference can also be a first step towards attribute inference attacks by allowing the attacker to verify the assumption that the target record is in the dataset. QuerySnout can be extended to membership inference attacks with minimal changes. Indeed, the main change to our method is in the sampling of auxiliary datasets used to train the attack: after sampling a dataset (as a subset of D' in the AUXILIARY scenario, or as D in the EXACT-BUT-ONE), with equal probability either a random record in the dataset is replaced with the target record, or the auxiliary dataset left unchanged. The rule G can then be trained to predict membership (whether the target record is in the data) rather than the value of the sensitive attribute. The rest of the method can be applied unchanged, although it is likely that designing specific mutations for membership inference might result in better accuracy.

Automating the analysis of solutions. In this paper, we manually inspected the queries found by QuerySnout to understand what vulnerabilities of the QBS they exploited. This analysis could likely be automated in the future, e.g. using techniques from interpretability in machine learning [55] to understand the relative importance of queries in the attack. This would greatly help better understand what the attack exploits and how the attack could be mitigated and patched to make the system stronger.

Automating the defense. QuerySnout discovers highly accurate attacks against real-world systems in the AUXILIARY scenario, and new, more powerful, attacks in the EXACT-BUT-ONE scenario. We believe that defending against these attacks will likely require both improving these systems but also ensuring that appropriate risk mitigation strategies are in place, including authentication, query limits, and logging. We believe however that the existence of an automated attack system such as QuerySnout might help develop stronger mitigation strategies such as anomaly detection and attack detection mechanisms.

Order of queries. We here assume that the order of queries does not matter to the QBS (Sec 4.2), a property satisified by all the query-based systems we consider, as well as, in our opinion, the majority of real-world systems. In some cases however, this might not be the case. For instance, one could imagine a system auditing queries in order to avoid answering the second half of difference query pairs [44]. Such a QBS could be represented by a function $R_k : \mathcal{D} \times \mathcal{Q} \times \mathcal{H} \to \mathbb{R}$, where $\mathcal{H}$ is the set of *histories*, i.e., lists of query-answer pairs. Here, solutions would be represented as *ordered* lists of queries rather than multisets, which increases the size of the solution search space by a factor of approximately $m!/3^n$. Our method can be adapted by adding a mutation operator which changes the query order.

9 RELATED WORK

Attacks have long been used to evaluate and improve query-based systems. Difference attacks against counting queries were studied by Denning et al. [19], as well as solutions based on randomisation [18], leading to the development of averaging attacks. The seminal linear reconstruction attack by Dinur and Nissim [20] allows to retrieve the records of a dataset using (many) counting queries, and gives a lower bound on the noise added by query-based systems to prevent it. Further work by Dwork and Yekhanin [25] showed that this attack could be performed efficiently with a smaller number of queries, and that the impossibility result extends to curators that aim to answer a small subset of queries accurately (while maintaining average noise level satisfying the lower bound). Diffix [27], developed by Aircloak, was improved through two bounty challenges [7, 8], where researchers were invited to develop attacks against it. Researchers showed that Diffix was vulnerable to reconstruction attacks using direct [13] and quasi-identifiers [12], as well as membership inference attacks [51]. Gadotti et al. proposed an attack specifically exploiting the noise structure of Diffix [30].

Query auditing [44] is a line of research which aims to detect whether queries can be answered without disclosing private information. Auditing can be performed in the offline setup, where m queries are audited together before being answered, and in the online setup, where each query is audited individually before being answered without knowledge of subsequent queries. While the latter is known to be challenging, even the former can be difficult: for instance, query auditing is co-NPHard for counting queries [38]. Research on query auditing has been restricted to systems that answer queries exactly ($R(d, q) = T(d, q)$). QuerySnout can be seen as an automated auditor applicable to more general QBSes.

Differential privacy [22] is a formal guarantee of privacy, which was proposed as a solution to the reconstruction attack of Dinur and Nissim. Researchers have proposed query-based systems guaranteeing differential privacy, such as PINQ [42], PriPearl [37], and Flex [36]. For the counting queries we consider, all three systems behave similarly to the DPLaplace QBS we attack in this paper. More complex mechanisms guaranteeing differential privacy have also been developed, such as query release through adaptive projection [5, 23] and the matrix mechanism [40, 41].

Our method uses a technique similar to shadow modelling to infer the rule G that combines answers to queries. Shadow modelling has been used extensively for property inference [4] and membership inference [58] attacks against machine learning algorithms. Pyrgelis et al. have used it to attack aggregated location data [50], and Stadler et al. have used it to develop membership inference attacks against synthetic data [59]. Machine learning-based attacks have also been used by Bichsel et al. to detect violations of DP [9].

Evolutionary algorithms have been applied to adversarial attacks against machine learning models [2, 15, 46, 61]. They are especially suitable under black-box attack scenarios where the attacker does not have access to the gradients or only hard (discrete) labels are available from the model. To the best of our knowledge, our work is the first to use evolutionary algorithms for attacks against QBSes.

10 CONCLUSION

In this paper, we propose QuerySnout, the first approach to automatically discover attribute inference attacks against query-based systems. QuerySnout discovers a multiset of queries and a rule to combine them. We learn the rule by training a machine learning classifier on answers from auxiliary QBSes protecting datasets sampled from the auxiliary knowledge available to the attacker. We use an evolutionary algorithm to find an optimal multiset of queries by iteratively improving a population of solutions using a mutation operator specifically tailored for this task. We show QuerySnout to find attacks against two deterministic real-world mechanisms, Diffix and TableBuilder, and a non-deterministic system (SimpleQBS). Across systems and datasets, the attacks found equate and often outperform previous known manual attacks. Finally, we show how QuerySnout can be extended to QBSes that require a budget and show it to approximately match the optimal accuracy against systems implementing the Laplace mechanism for large values of ϵ. We also discuss extensions of our attack to other attack models (e.g., membership inference) and query syntax.

Taken together, our results show how QuerySnout can be used to automatically find powerful attacks against QBSes and evaluate the privacy protection they offer. We believe that automated attack discovery procedures such as QuerySnout will help detect issues before systems are deployed, helping to patch the systems and mitigate risks, and ultimately help design query-based systems offering a high level of protection in practice.

Acknowledgements. We thank Bozhidar Stevanoski for his feedback, the anonymous reviewers for their comments which have helped improve the paper, and Tianhao Wang for shepherding our paper. A.-M. C. was partially funded by UKRI Research England via the "Policy Support Fund 2021/22 Evidence-based policy making" call. We acknowledge computational resources and support provided by the Imperial College Research Computing Service[6].

[6] http://doi.org/10.14469/hpc/2232

REFERENCES

[1] 2016. General Data Protection Regulation. https://gdpr-info.eu/.

[2] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, and Mani B Srivastava. 2019. Genattack: Practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1111–1119.

[3] Hassan Jameel Asghar and Dali Kaafar. 2020. Averaging Attacks on Bounded Noise-based Disclosure Control Algorithms. *Proceedings on Privacy Enhancing Technologies* 2 (2020), 358–378.

[4] Giuseppe Ateniese, Giovanni Felici, Luigi Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. 2013. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *International Journal of Security and Networks* 10 (06 2013). https://doi.org/10.1504/IJSN.2015.071829

[5] Sergul Aydore, William Brown, Michael Kearns, Krishnaram Kenthapadi, Luca Melis, Aaron Roth, and Ankit Siva. 2021. Differentially private query release through adaptive projection. *arXiv preprint arXiv:2103.06641* (2021).

[6] Borja Balle, Giovanni Cherubin, and Jamie Hayes. 2022. Reconstructing Training Data with Informed Adversaries. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1556–1556.

[7] Felix Bauer. 2017. Announcing the first ever bug bounty program for a privacy protection solution. https://aircloak.com/announcing-the-first-ever-bug-bounty-program-for-a-privacy-protection-solution/

[8] Felix Bauer. 2020. The World's Only Anonymization Bug Bounty – Round Two! https://aircloak.com/the-worlds-only-anonymization-bug-bounty-round-two/

[9] Benjamin Bichsel, Samuel Steffen, Ilija Bogunovic, and Martin Vechev. 2021. DP-sniper: black-box discovery of differential privacy violations using classifiers. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 391–409.

[10] Franziska Boenisch, Reinhard Munz, Marcel Tiepelt, Simon Hanisch, Christiane Kuhn, and Paul Francis. 2021. Side-Channel Attacks on Query-Based Data Anonymization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1254–1265.

[11] James Chipperfield, Daniel Gow, and Bronwyn Loong. 2016. The Australian Bureau of Statistics and releasing frequency tables via a remote server. *Statistical Journal of the IAOS* 32, 1 (2016), 53–64.

[12] Aloni Cohen, Sasho Nikolov, Schutzman Zachary, and Jonathan Ullman. 2020. Reconstruction Attacks in Practice. https://differentialprivacy.org/diffix-attack/. Accessed: 2021-6-25.

[13] Aloni Cohen and Kobbi Nissim. 2018. Linear Program Reconstruction in Practice. *TPDP 2018* (Oct. 2018).

[14] Helen J Curtis and Ben Goldacre. 2018. OpenPrescribing: normalised data and software tool to research trends in English NHS primary care prescribing 1998–2016. *BMJ open* 8, 2 (2018), e019921.

[15] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *International conference on machine learning*. PMLR, 1115–1124.

[16] Kenneth Alan De Jong. 1975. *An analysis of the behavior of a class of genetic adaptive systems*. PhD Thesis. University of Michigan.

[17] Yves-Alexandre de Montjoye, Sébastien Gambs, Vincent Blondel, Geoffrey Canright, Nicolas De Cordes, Sébastien Deletaille, Kenth Engø-Monsen, Manuel Garcia-Herranz, Jake Kendall, Cameron Kerry, et al. 2018. On the privacy-conscientious use of mobile phone data. *Scientific data* 5, 1 (2018), 1–6.

[18] Dorothy E Denning. 1980. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems (TODS)* 5, 3 (1980), 291–315.

[19] Dorothy E Denning and Peter J Denning. 1979. The tracker: A threat to statistical database security. *ACM Transactions on Database Systems (TODS)* 4, 1 (1979), 76–96.

[20] Irit Dinur and Kobbi Nissim. 2003. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 202–210.

[21] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[22] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

[23] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[24] Cynthia Dwork, Adam Smith, Thomas Steinke, Jonathan Ullman, and Salil Vadhan. 2015. Robust traceability from trace amounts. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 650–669.

[25] Cynthia Dwork and Sergey Yekhanin. 2008. New efficient attacks on statistical disclosure control mechanisms. In *Annual International Cryptology Conference*. Springer, 469–480.

[26] William Feller. 1968. Probability theory and its applications, vol. 1 New York.

[27] Paul Francis, Sebastian Probst Eide, and Reinhard Munz. 2017. Diffix: High-utility database anonymization. In *Annual Privacy Forum*. Springer, 141–158.

[28] Paul Francis, Sebastian Probst-Eide, Pawel Obrok, Cristian Berneanu, Sasa Juric, and Reinhard Munz. 2018. Extended diffix. *arXiv preprint arXiv:1806.02075* (2018).

[29] Bruce Fraser and Janice Wooton. 2005. A proposed method for confidentialising tabular output to protect against differencing. *Monographs of Official Statistics: Work Session on Statistical Data Confidentiality* (2005), 299–302.

[30] Andrea Gadotti, Florimond Houssiau, Luc Rocher, Benjamin Livshits, and Yves-Alexandre de Montjoye. 2019. When the signal is in the noise: Exploiting Diffix's Sticky Noise. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1081–1098.

[31] Graham Greenleaf. 2021. Global Data Privacy Laws 2021: Despite COVID Delays, 145 Laws Show GDPR Dominance. (2021). https://doi.org/10.2139/ssrn.3836348

[32] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. 2008. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genet* 4, 8 (2008), e1000167.

[33] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[34] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. 2020. Auditing differentially private machine learning: How private is private sgd? *Advances in Neural Information Processing Systems* 33 (2020), 22205–22216.

[35] Noah Johnson, Joseph P. Near, and Dawn Song. 2017. Towards Practical Differential Privacy for SQL Queries. *ArXiv e-prints* (Jun 2017). http://arxiv.org/abs/1706.09479 arXiv: 1706.09479.

[36] Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment* 11, 5 (2018), 526–539.

[37] Krishnaram Kenthapadi and Thanh TL Tran. 2018. Pripearl: A framework for privacy-preserving analytics and reporting at linkedin. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2183–2191.

[38] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. 2003. Auditing boolean attributes. *J. Comput. System Sci.* 66, 1 (2003), 244–253.

[39] David Lazer, Alex Pentland, Lada Adamic, Sinan Aral, Albert-Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. 2009. Social science. Computational social science. *Science* 323, 5915 (2009), 721–723.

[40] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau, and Andrew McGregor. 2010. Optimizing linear counting queries under differential privacy. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 123–134.

[41] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal* 24, 6 (2015), 757–781.

[42] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 19–30.

[43] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.

[44] Shubha U Nabar, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. 2008. A survey of query auditing techniques for data privacy. In *Privacy-Preserving Data Mining*. Springer, 415–431.

[45] Joe Near. 2018. Differential privacy at scale: Uber and berkeley collaboration. In *Enigma 2018 (Enigma 2018)*.

[46] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[47] Axel Oehmichen, Shubham Jain, Andrea Gadotti, and Yves-Alexandre de Montjoye. 2019. OPAL: High performance platform for large-scale privacy-preserving location data analytics. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 1332–1342.

[48] Christine M O'Keefe, Stephen Haslett, David Steel, and Ray Chambers. 2008. Table builder problem-confidentiality for linked tables. (2008).

[49] OpenSafely. 2021. Secure analytics platform for NHS electronic health records. https://www.opensafely.org/ Accessed on Nov 8, 2021.

[50] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. 2018. Knock knock, who's there? Membership inference on aggregate location data. *NDSS* (2018).

[51] Pyrgelis, Apostolos. 2018. On Location, Time, and Membership: Studying How Aggregate Location Data Can Harm Users' Privacy. https://www.benthamsgaze.org/2018/10/02/on-location-time-and-membership-studying-how-aggregate-location-data-can-harm-users-privacy/.

[52] UCI Machine Learning Repository. 1996. Adult Data Set. https://archive.ics.uci.edu/ml/datasets/adult

[53] UCI Machine Learning Repository. 2000. Census-Income (KDD) Data Set. https://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)

[54] UCI Machine Learning Repository. 2000. Insurance Company Benchmark (COIL 2000) Data Set. https://archive.ics.uci.edu/ml/datasets/Insurance+Company+

Benchmark+(COIL+2000)

[55] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[56] Yosef Rinott, Christine M. O'Keefe, Natalie Shlomo, and Chris Skinner. 2018. Confidentiality and Differential Privacy in the Dissemination of Frequency Tables. *Statist. Sci.* 33, 3 (2018), 358 – 385. https://doi.org/10.1214/17-STS641

[57] Kumara Sastry, David Goldberg, and Graham Kendall. 2005. Genetic algorithms. In *Search methodologies*. Springer, 97–125.

[58] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[59] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. 2020. Synthetic Data–Anonymisation Groundhog Day. *arXiv preprint arXiv:2011.07018* (2020).

[60] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[61] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation* 23, 5 (2019), 828–841.

[62] Eric J Topol. 2019. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine* 25, 1 (2019), 44–56.