Privacy Changes Everything

Jennie Rogers[1], Johes Bater[1], Xi He[2], Ashwin Machanavajjhala[3],
Madhav Suresh[1], and Xiao Wang[1]

[1] Northwestern University [2] University of Waterloo [3] Duke University

Abstract. We are storing and querying datasets with the private information of individuals at an unprecedented scale in settings ranging from IoT devices in smart homes to mining enormous collections of click trails for targeted advertising. Here, the *privacy* of the people described in these datasets is usually addressed as an afterthought, engineered on top of a DBMS optimized for performance. At best, these systems support *security* or managing access to sensitive data. This status quo has brought us a plethora of data breaches in the news. In response, governments are stepping in to enact privacy regulations such as the EU's GDPR. We posit that there is an urgent need for *trustworthy database systems* that offer end-to-end privacy guarantees for their records with user interfaces that closely resemble that of a relational database. As we shall see, these guarantees inform everything in the database's design from how we store data to what query results we make available to untrusted clients.

In this position paper we first define trustworthy database systems and put their research challenges in the context of relevant tools and techniques from the security community. We then use this backdrop to walk through the "life of a query" in a trustworthy database system. We start with the query parsing and follow the the query's path as the system plans, optimizes, and executes it. We highlight how we will need to rethink each step to make it efficient, robust, and usable for database clients.

1 Introduction

Now that storage is inexpensive, organizations collect data on practically all aspects of life, with much of it pertaining to individuals using their systems. They do so with little transparency regarding how they will analyze, share, or protect these records from prying eyes. Instead, their systems are optimized for performance. The way mainstream databases protect their contents today is haphazard at best. Beyond straightforward measures – like passwords, role-based access control, and encrypted storage – they offer scant protection for private data after the engine grants access to it and no commercial system takes into account the privacy of individuals in the database. As such, we see data breaches in the news with astounding regularity. We are already seeing governments step in to enact new laws in response to this, including the EU's GDPR and California's privacy act, the CCPA. The time has come for us to think more systematically about how to be good stewards of this growing resource. As database researchers and practitioners, we face a new challenge: how to keep the data entrusted to our systems private.

Trustworthy database systems offer end-to-end privacy guarantees with a user interface that closely resembles that of a relational database. They are designed to protect their contents as a first principle – informing how we store private data, run queries over it, and manage their outputs. In addition, they must be as easy to use as possible to make privacy-preserving techniques accessible to existing database administrators and clients. We need to reimagine these systems with privacy as a first-class citizen in their design. As we shall see, this calls for dramatic changes to almost every aspect of a DBMS's operations.

Privacy changes everything. We investigate this thesis by stepping through the life-cycle of a query with two use cases. First, we look at protecting the privacy of input data from an untrusted client who may view only approximate results from their queries. Second, we examine a scenario where private data owners outsource their operations – storage and querying – to an untrusted cloud service provider. Here, the data owners carefully choose what information, if any, about their secret records will be revealed to the service provider. The data owners may alone may view their query results.

In the private inputs setting, data owners run queries from untrusted clients. Here, the clients' query results must be sufficiently noisy such that they cannot reconstruct the data owner's private records even after repeatedly querying the engine. Differential privacy [16] addresses this by using a mechanism to introduce carefully controlled levels of noise into the query results. To date, most of the results in this space – with the exception of [28,29] – have been theoretical in nature. As we shall see integrating differential privacy into the query processing pipeline, rather than noising query results after evaluating them in a regular DBMS, may produce more precise results for the client [6,28,29].

In the cloud setting, an untrusted service provider offers storage and query processing of private data to its owner. Here, we need to ensure that data is encrypted at rest and that query processing is privacy-preserving and oblivious. A query execution is *oblivious* if its observable transcript – the movement of the program counter, accesses of the memory, network traffic – is independent of the query's inputs. *Secure computation* [57] supports these guarantees by constructing cryptographic protocols that simulate running the query on a hypothetical trusted third party by passing encrypted messages among the data owners. For settings where the database's schema as a security policy with public and private columns or tables, we may analyze these queries and create a hybrid execution plan that partitions a given query into subplans that may be executed in the clear or in secure computation [5,50,60]. Since secure computation has an overhead that is typically 1,000X or more slower than executing the same program in the clear, even incremental changes of this kind are a big performance win.

The security community has developed a myriad of techniques [31,23,53,13] for protecting private data in these settings and more. To date these solutions have been largely piece-wise, and they don't address the end-to-end workflow of a DBMS query execution. Moreover, the current offerings typically require multiple PhD-level specialists to deploy them and most of their applications are hard-coded, i.e., they support only a handful of "benchmark" queries and they do not accept ad-hoc queries written in SQL or any other well-known query language. There has been limited work on how to build end-to-end systems with provable privacy guarantees starting from how they store and query private records and concluding by perturbing their query outputs enough to prevent an attacker from revealing their secret inputs even with carefully targeted repeat querying. Offering these guarantees *while* providing a user experience that has the look and feel of a conventional DBMS will mean tackling many interesting research challenges in query processing, optimization, and more. Making trustworthy database systems efficient, robust, and usable will require a more holistic view of how a database's internals work together. This is an opportunity for the database community since many of these technologies – including differential privacy and secure computation – are just now becoming robust and efficient enough for real-world deployment.

Building privacy-preserving data management systems is hard because of the inherent complexity of DBMSs. Until now, database researchers largely focused on providing high performance with semantic guarantees like referential integrity [8,55]. In contrast, trustworthy database systems need to optimize over a multi-objective decision space – trading off among performance, the data's long-term privacy, result accuracy, and the difficult-to-quantify value of additional guarantees such as cryptographically verifying the provenance of input data or the integrity of a query's execution. Moreover, composing these assurances is a non-monotonic cost model for query optimization – some are synergistic, antagonistic, or even mutually exclusive! Mere mortals cannot reason about composing these privacy-preserving techniques in a DBMS as they exist today.

At the same time, database researchers have a lot to offer to this emerging challenge of making privacy-preserving data analytics practical and usable. We have extensive research contributions in query optimization, parallelizing large-scale analytics, materialized view selection, and more. Generalizing these techniques to trustworthy database systems will be a non-trivial undertaking. For example, in the private-inputs setting a query plan may produce more accurate results when it runs over a differentially private view of a dataset [29] although querying the view has slower performance because it reads more data from disk. Many well-known query optimizations in the database community – such as using semi-joins for parallel databases, and splitting the execution of aggregates between local and distributed computation – generalize to the cloud setting to produce big performance gains [5]. Similarly, when we run oblivious queries in the cloud we will realize much greater performance if we build our secure computation protocols for each operator on the fly – such as compiling expressions into low-level circuits – and this will build from recent work on just-in-time query compilation [36,39,43].

The rest of this paper is organized as follows. We first define trustworthy database systems in detail with two illustrative reference architectures. We then describe privacy-preserving techniques that will lay the groundwork for query evaluation in these systems. After that, we walk through how we will need to rethink the query processing pipeline to support secure and trustworthy data management. We then conclude.

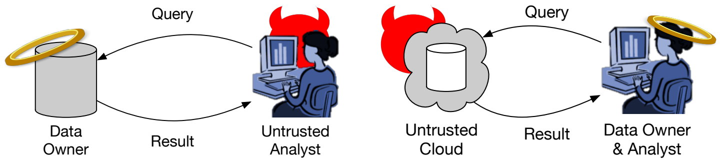
2 Background

We will now describe two motivating reference architectures for trustworthy database systems. We will then discuss two privacy-preserving techniques to support these systems: secure computation and differential privacy. As we shall see, they require integration throughout the entire query life-cycle, introducing substantial changes to most or all of the components in a DBMS.

2.1 Reference Architectures

Before delving into research challenges of trustworthy database systems, we look at two motivating scenarios for this work. To a first approximation, these systems have three roles: the data owner, the client, and the service provider. The data owner has private data that they wish to make available for querying. The client writes SQL queries against the trustworthy database system's schema and receives query results that may be precise or noisy. The service provider physically stores the private data and executes queries over it, returning the results to the client. A participant may support two of these roles. Trustworthy database systems address settings where there is at least one untrusted participant in a query over private data. Although the architectures below have a single data owner and one client, it is possible to extend these setting to multiple data owners and two or more independent clients.

It will be crucial for these systems to offer a user experience that is close to conventional engines to enable as many people as possible to benefit from privacy-preserving techniques. These systems will need to provide transparency about how they store and access data to the data owner and to the clients. They will also need to automate compliance for companies by composing high-level declarative security policies and applying them to ad-hoc queries. Thus we frame the setup for these systems in terms of the whether the participants in each of these three roles are trusted or untrusted.



(a) Private Inputs (b) Cloud

Fig. 1: Reference architectures for trustworthy database systems.

Figure 1 shows two reference architectures that we will use to motivate and illustrate this work. In each one, we denote a party as trusted with a halo. A trusted party is permitted to view the private input data we are querying. We can say that they sit within the privacy firewall. We show an untrusted party, who resides outside the privacy firewall, with horns.

In the *private inputs* architecture [26,28,35] a data owner acts as their own service provider by storing their private dataset locally and offering it for querying to an untrusted client. Hence, the client may only see noisy query results such that they cannot deduce the precise values in the data owner's tuples. From the client's perspective, this system behaves exactly like a standard DBMS. The engine will authorize the user, determine how much they are permitted to learn about the dataset, prepare and optimize a query plan that upholds the the system's privacy guarantees, execute it, and return a table of tuples to them.

When a data owner wishes to outsource their data storage and query processing to an untrusted service provider, we say that they are in the *cloud* setting [1,22,24]. Data owners encrypt their records before sending it to the service provider and issues queries over their private tuples remotely. Since the cloud provider cannot see the contents of the database, we will use advanced cryptographic techniques to protect this data, including fully homomorphic encryption [21] (to outsource the computation and storage), verifiable computation [41] (to outsource the storage), and zero-knowledge proofs [59,20] (to outsource the computation and storage). By systematically composing these techniques, a cloud service provider will execute the data owner's queries without learning anything about the private data it is storing even for ad-hoc workloads. Recall that a server's query evaluation is oblivious when it reveals no information about its secret inputs. For query evaluation, this means running in worst-case time and space to not leak information about its private inputs. Hence, a join of two relations of length n must do n^2 tuple comparisons, each of which emits a tuple that is either a dummy or a real one to mask the join's selectivity. Naturally this overhead cascades up the query tree creating an explosion in the query's intermediate cardinalities. In some cases, the outsourced server may run queries semi-obliviously, such as if they use computational differential privacy to make the query's program traces noisy [6] or if the system has a security policy where some columns are publicly readable [5,47].

The systems above are examples of trustworthy database systems. They are a small sample of the database settings that will benefit from privacy-preserving techniques. Others include privacy-preserving analytics for querying the union of the private data of multiple data owners [5,10,50,47], support for distributed "big data" platforms [3,18,60], and querying encrypted data [44,49]. The big data systems use trusted hardware to make their guarantees. Each of these settings substantially changes how we reason about and apply privacy-preserving techniques. In addition, we focus on two security guarantees in this work: privacy-preserving query processing and mechanisms for producing efficient and private query results. There are many other guarantees that are outside the scope of this work, and may be of interest in future research. They include running secret queries over publicly available data [51], running SQL over the secret inputs of multiple private data owners [5,50] and decentralized verifiable database tables a la blockchain [2,38,17].

2.2 Secure Computation

Secure computation refers to cryptographic protocols that run between a set of mutually distrustful parties. The security of these protocols allows all parties to perform computations as if there is a trusted third party who runs the program and reveals only its output. In the cloud setting, we run secure computation protocols by having two or more untrusted hosts work together to compute query results over secret data. This prevents any one host from being able to "unlock" the data on its own. The concept of secure computation was invented more than 30 years ago [58]; in the last decade, this technology has witnessed significant growth in its practicality. Numerous start-ups based on various secure computation technologies have been founded to use related cryptographic techniques to protect financial information [9], for anonymous reporting of sexual misconduct [45], private auctions [11], and more.

Secure computation has been used in the cloud and data federation settings for query evaluation over private data. In the cloud, data owners use secure computation to query their private records using an untrusted service provider [1,56]. In a data federation, oblivious query processing was researched in [5,6,50,47]. Almost all secure computation protocols follow the gate-by-gate paradigm with the following steps: 1) represent the computation as a circuit; 2) execute a secure subprotocol that securely encrypt the input data for evaluation in the circuit; 3) following the topological order of the circuit, evaluate all gates therein. Usually, the evaluation of each gate incurs some computational and communication cost, which becomes significant when the computation is complex. Many meaningful computations usually require billions of gates leading to a high computation and communication cost. Recent work studied optimizations of the cost of secure computation protocols and most practically efficient protocols right now are communication-bound owing to the need for data owners to pass messages amongst themselves to jointly evaluate each gate. In the past, secure computation was CPU-bound, but hardware optimizations, such as specialized instructions for cryptographic primitives, have shifted their bottleneck [25,7]. Presently, the only exceptions to this network-bound query evaluation are ones that heavily rely on public-key operations [27], where the computation returns to being the bottleneck. For example, secretly computing a single join with 1000 input tuples per relation incurs over 10GB of network traffic with state of the art secure computation implementations.

Zero-knowledge proofs (ZKP) can be viewed as a special type of secure computation, where only one party (i.e., prover) has the input, and the other party (i.e., verifier) obtains one bit of output indicating if a certain public predicate is true on the prover's input. For our reference trustworthy database systems, they will be useful for the client to verify that their query was evaluated faithfully over the entirety of the relations it is querying. In the private inputs setting, the data owner may use ZKPs to prove to the client that the noisy results they are receiving are correct and complete. To do this, the data owner first publishes a digest of the database, which does not reveal any information about its contents but binds the database's contents. When the data owner receives a query, they will return the result to the client with a proof of its correctness that the client verifies by combining it with the initial digest. This was studied in VSQL [59]. Cloud-based systems may offer the same assurances with the service provider generating the digest and proofs for the data owner.

2.3 Differential Privacy

Secure computation maintains the confidentiality of the input dataset during query execution, but it offers no guarantees on whether sensitive values in the dataset can be inferred or “reconstructed" from the output of a query. The classic Dinur-Nissim result [15] (aka the fundamental law of information reconstruction) states that answering $n \log^2 n$ aggregate queries (with sufficient accuracy) on a database with n rows is sufficient to accurately reconstruct an entire database. This result has practical implications: recently, the US Census Bureau ran a reconstruction attack using only the aggregate statistics released under the 2010 Decennial Census, and was able to correctly reconstruct records of address, age, gender, race and ethnicity of about 46% of the US population.

Differential privacy is the only suite of techniques that ensure safety against reconstruction attacks [16]. An algorithm is said to satisfy differential privacy if its outputs do not change significantly due to adding/removing or updating a row in the input database. Differential privacy is currently considered the gold standard for ensuring privacy in most data sharing scenarios and has been adopted by several organizations, including the US Census Bureau (for their upcoming 2020 Decennial Census), and tech companies like Google, Apple, Microsoft and Uber.

Differential privacy injects carefully controlled levels of noise into a query’s results. A private dataset begins with a privacy budget defining how much information about the data may be revealed in noisy query results. Each query receives some quantity of the privacy budget. We calibrate the noise with which we perturb our query results as a function of the query’s privacy allocation and the sensitivity of the its operators. Speaking imprecisely, a query’s sensitivity reflects how its output will change if we add, remove or modify an arbitrary row in the database.

An important property of differentially private algorithms is their composition also satisfies differential privacy. This is useful for proving the privacy guarantees of complex queries and it addresses the impossibility result by Dinur and Nissim. Moreover, querying a differentially private data release of a database does not incur any privacy cost other than that of initially noising the data release. This is useful for workloads with many queries over a single dataset.

Computational relaxations of standard differential privacy, known as computational differential privacy [37], aim to protect against computationally-bounded adversaries by protecting data in flight in the cloud. This serves as an alternative to full-oblivious query processing with its worst-case runtime. Instead computational differential privacy ensures that each party’s view of the protocol is differentially private with respect to its secret inputs. For example, consider query evaluation with secure computation on two non-colluding cloud providers. Without the computationally bounded assumption on each party, any differentially private protocols for computing the Hamming distance between two n-bit vectors incur an additive error of $\Omega(\sqrt{n})$ [34]. On the other hand, by assuming each party is computationally bounded, this error can be reduced to $O(1)$.

3 The Life of a Privacy-preserving Query

We now step through the workflow of a relational database query covering from when the client submits a query until they receive their results. We will examine how the major steps in the query processing pipeline will need to be redesigned in this emerging setting using our reference architectures from Section 2.1.

3.1 Query Parsing and Authorization

When a database engine receives a SQL statement, it first verifies that the query is free of syntax errors and resolves all names and references in it. It then converts the statement into one or more directed acyclic graphs (DAGs) of database operators. Lastly, it verifies that the user is authorized to run the query under the system's security policy.

When the parser initially verifies a SQL statement, a trustworthy database system may offer an extended syntax for queries. Although standard SQL queries are supported, the user may optionally give the system information about the how to run the query and manage its use of privacy, such that if a user is given a limited privacy budget they may split it as they see fit over their query workload giving more privacy for high-priority queries to increase the utility of their results. The parser may accept directives such as declaring a cardinality bound for a given database operator and annotations specifying the privacy budget that the query will use on the data it is accessing. Alternatively, the client may specify bounds on the accuracy of a query's results that he or she deems acceptable – ensuring that the utility of the data is not destroyed by over-noising the query results – and preventing the client from eroding the privacy budget for results that will not be useful to them.

When the planner converts the query into a DAG, it also needs to analyze the data it is querying and operations the user wishes to run to check that they are permitted by the data owner's security policy. Before we can optimize a query, we need to run information flow analysis over SQL to determine what type(s) of query processing will be necessary to uphold a given security policy. For example, if a database in the cloud has a mix of public and private columns, we will use differential privacy and secure computation only when we compute on private data. The engine will also need to solve for the sensitivity of a given operator in order to determine the noise it will need to inject for differentially private query results.

For checking query authorization, prior work has largely revolved around the user's privileges. A trustworthy database must consider many more factors such as the consent of the individuals in the dataset and the remaining privacy budget available for the data. It may also contend with how to compose many disparate privacy policies. For example, we are presently preparing to deploy a prototype of a trustworthy database system for analytics over electronic health records. Our colleagues in medicine compiled a memo listing all of the known state-level regulations pertaining to health data in the US. It is nearly 550 pages long. Research on how to compose privacy policies such as these will make it possible for trustworthy database systems to operate in complex regulatory environments.

3.2 Query Rewriting

The standard query rewriter takes the query tree from the parser and canonicalizes it for the optimizer. Here, the query planner coalesces SELECT blocks, expands any views, simplifies predicates, and more. This enables the query optimizer to produce efficient query plans and to make them consistent, i.e., where two semantically equivalent SQL statements yield identical query execution plans.

For queries running in the cloud, it is essential to have query rewrite rules that minimize the use of secure computation. Oblivious query processing typically runs at least three orders of magnitude slower than doing the same work in the clear. The query rewriter automatically applies any annotations from the query for bounding its intermediate cardinalities. It can also leverage information from the schema, such as integrity constraints and primary keys, to reduce the output size of the operators. Since the operators themselves must still run in worst-case time, the rewriter may inject "shrinkwrap" operators after an operator with a bounded cardinality to obliviously reduce its tuple count before passing them up to its parent. This technique was further developed to reduce the query's intermediate cardinalities using computational differential privacy to reveal padded versions of the true cardinality [6]. Despite a measurable privacy loss from the data owners observing intermediate results that are not exhaustively padded, clients receive precise query answers with a fast speed. Placing shrinkwrap operators in cloud query plan offers a new tuning knob in our query optimization space.

For the private inputs setting, we need to consider the level of noise added to a query's result. First we need to ensure that the sensitivity computation for the given query tree is correctly analyzed with regard to private tables for adding sufficient amount of noise. Prior work [19,26,35,28] has focused on the linear aggregates at the end of the query tree, such as COUNT and SUM, and they add noise directly to the final aggregate. However, non-linear aggregates like AVG and STD call for more complicated perturbation algorithms that add noise to the intermediate results. For example, to release the average value of a column, we first compute the noisy sum of that column and the noisy count of that column, and then take their ratio. Whether to rewrite these aggregate into multiple operators (sub-queries) to facilitate noise addition and sensitivity analysis is an important extension. Next, for more accurate query results, we may rewrite a query in a form that is more DP-friendly and use inference to work back to the original SQL statement. In this approach, the rewritten query is no longer semantically equivalent to the initial SQL statement, but the noise added to the new query answer is much reduced. For instance, in the PrivateSQL system [29], *truncation* operators are added into the query tree to limit the maximum multiplicity of joins or range of an attribute's values so that query answers (or intermediate join cardinalities as in the case of Shrinkwrap) can be released with low noise. Where to insert the new operators in the query tree and how to set the truncation threshold remains challenging in practice. In addition, PrivateSQL is able to offer flexible privacy policies to the data owner, and the sensitivity of a query depends on the privacy policy. For example, a policy that protects entire households would generally have higher sensitivity than a policy that protects individuals. PrivateSQL rewrites queries to enable automatically calculation of the appropriate sensitivity for a class of foreign key based privacy policy. Generalizing this query rewriting approach for more rich set of class policies is an open question.

3.3 Query Optimization

A conventional query optimizer takes in a query tree from the rewriter and transforms it into an efficient query execution plan by selecting the order of commutative operators, the algorithms with which each one will execute, and the access paths for its inputs. The optimizer typically uses a cost model to compare plans to pick one that will run efficiently and enumerates plans using dynamic programming.

When we optimize a trustworthy database system query, we almost always do so in a multi-objective decision space. Depending on the setting, the optimizer may negotiate trade-offs among performance, information leakage, results accuracy, and storage size (if we use materialized views). For example, a cloud deployment using computational differential privacy to reduce the size of a query's intermediate results will have to decide how to split the privacy budget over its shrinkwrap operators to get the biggest performance boost. The more privacy an operator uses, the less padding its intermediate results will need. We will need to generalize multi-objective query optimization [4,48] to tackle this challenge of creating query plans that satisfy these goals.

Moreover, optimizing information leakage gets more challenging when we consider database design. In the private inputs setting, we may create differentially-private views of the data for repeated querying so that we do not have to use our privacy budget for every query we run. We need to take a holistic view of how the major components in the DBMS work together in order to decide the best way to selectively leak information about private data so that we do not compromise information on individuals in a dataset yet still offer efficient query runtimes.

For the private inputs setting, the optimizer will need to balance competing goals of finding an efficient execution plan and one that produces private results with minimal noise. This two-dimensional optimization space will not be amenable to standard dynamic programming-style search algorithms. We suspect that the optimizer will use machine learning to find a plan that satisfies these competing goals. This will build on research in autonomic query optimization [14,33,42] and recent advances in using deep learning for the same [30,32,40,52]. The optimizer needs models for the sensitivity of a query plan and the expected noisiness of its results. It will select an access path from the initial relation, an index on it, or a differentially private view. The engine will need to automatically determine how using a noisy view of the data will impact the accuracy of a query's results and the speed of its execution. It would model its selectivity estimation using standard techniques since this information is only visible to the data owner. Unlike most prior query optimization research that is performance-focused, an engine with differentially private query results will need to work with the data owner or client to make explainable trade-offs between accuracy, privacy utilization, and runtime – perhaps by accepting bounds for one or more of these dimensions in an extended SQL syntax as described in Section 3.1.

For full-oblivious query processing in the cloud, our optimizer's decision space is limited. Since we exhaustively pad the output of each operator, reordering joins and filters does not matter. Shrinkwrapping expands our decision space by using computational differential privacy to reduce the size of intermediate cardinalities. On the other hand, the optimizer now faces the added challenge of splitting the privacy budget over the result sizes each intermediate operator in the query tree.

Even with privately padded intermediate results, the optimizer must make decisions that are data-independent. Without incorporating privacy into system catalog's statistics collection, it cannot use any statistics to order query operators, pick access paths, or to select operator algorithms. Instead the optimizer will use heuristics to estimate the size of intermediate cardinalities, like the $\frac{1}{10}$ selectivity rule [46]. Using these statistics, it will plug in a cost model for the query's secure computation.

For the optimizer's cost model, rather than estimating the number of I/Os or the CPU time a query will use, it will reason about a query's performance in terms of the number of secure computation operations – usually garbled circuit gates or arithmetic ops – it will run. This is because the cost of running the gates is predominantly network-bound, followed by being CPU bound when the network is exceptionally fast. Also, not all gates have the same CPU and network overhead. For example, XOR gates are "free" where as AND/OR gates are extremely costly. Thus finding the cheapest *circuit* representation for oblivious query operators will likely require low-level algorithm design. Optimizing at the level of a circuit will be quite different from working one operator at a time. In particular we will need new tactics to parallelize them circuits to maximize their throughput. In addition, there will be interesting research challenges in selecting the right secure computation protocol for a given query. This will require reasoning about the performance of each one and the guarantees it offers.

3.4 Plan Execution

After the query optimizer, we will have a secure and executable query plan. Right now, SQL queries usually run on a single machine or a cluster of machines that trust each other, where there is no privacy guarantee between the hosts. When privacy comes into the picture, we need to incorporate the aforementioned techniques to ensure that no (or limited information) can be revealed.

If we are operating in the cloud, for example, we can translate the optimized database operators into secure computation protocols. These programs are almost always fine-grained. Their unit of computation is the CPU instruction, usually a logical operation (AND/OR/NOT) or an arithmetic one (ADD/MULT). This means that secure computation Turing complete, but the cost of each operation is extremely high. Using secure computation, the engine now has a secure and executable physical query plan. Secure computation provides a strong security guarantee on the plan-execution computation. Recall that the query's execution must be oblivious – run such that its observable behavior is data-independent – and preserve the confidentiality of its input data. Ordinarily, we achieve the former using oblivious RAM. In SMCQL, they also tried to optimize the execution such that the non-secure portion of the program does not need to be executed in secure computation and thus improving the running time significantly. Differential privacy is an important tool to ensure the privacy of the secret input records by injecting a carefully controlled level of noise into the output of a query. A baseline approach to creating outputs is to do standard query processing and perturb the output of the query according to the cumulative sensitivity (i.e., how much an individual record can alter a query's outcome) of its operators [35]. Integrating differential privacy into our query executor will yield much better performance and query results with higher utility [12,29].

Prior works are mostly focused on two-party secure computation protocols sometimes combined with oblivious RAM. Oblivious RAM is a general purpose platform to mask and disguise memory access patterns. Other tools can potentially be helpful in this context too. For example, a multi-party computation protocol can support more than two parties where a subset of them can be corrupted. However, new analysis is required to study how to generalize the techniques in the two-party setting to the multi-party setting. Oblivious data structures are another example, that can accelerate the execution by orders of magnitude [54,60]. Existing oblivious data structures are general-purpose, and it is an important problem to design specialized oblivious data structures for query execution.

4 Conclusions

As organizations collect more and more sensitive data on their users, the need to build privacy-preserving techniques into database systems has never been greater. Ensuring the privacy of datasets as well as that of individuals within a database will require redesigns of numerous core database components. Guaranteeing that all of these components work together efficiently and correctly (in terms of composing their privacy guarantees) so that database users who are not privacy specialists may use them presents many novel research challenges. It will take deep collaborations between database researchers and members of the security community to make trustworthy database systems robust, usable, and scalable.

References

1. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: A distributed architecture for secure database services. CIDR (2005)
2. Allen, L., Antonopoulos, P., Arasu, A., Gehrke, J., Hammer, J., Hunter, J., Kaushik, R., Kossmann, D., Lee, J., Ramamurthy, R., et al.: Veritas: shared verifiable databases and tables in the cloud. In: 9th Biennial Conference on Innovative Data Systems Research (CIDR) (2019)
3. Arasu, A., Blanas, S., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R., Upadhyaya, P., Venkatesan, R.: Secure database-as-a-service with cipherbase. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. pp. 1033–1036. ACM (2013)
4. Balke, W.T., Güntzer, U.: Multi-objective query processing for database systems. In: Proceedings of the Thirtieth international conference on Very Large Databases. vol. 30, pp. 936–947. VLDB Endowment (2004)
5. Bater, J., Elliott, G., Eggen, C., Goel, S., Kho, A., Rogers, J.: SMCQL: Secure Querying for Federated Databases. Proceedings of the VLDB Endowment **10**(6), 673–684 (2017)

6. Bater, J., He, X., Ehrich, W., Machanavajjhala, A., Rogers, J.: Shrinkwrap: Differentially-Private Query Processing in Private Data Federations. Proceedings of the VLDB Endowment **12**(3), 307–320 (2019)
7. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy. pp. 478–492. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013)
8. Benedikt, M., Leblay, J., Tsamoura, E.: Querying with access patterns and integrity constraints. Proc. VLDB Endow. **8**(6) (Feb 2015)
9. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and Taxes: A Privacy-Preserving Social Study Using Secure Computation. In: Privacy Enhancing Technologies Symposium (PETS) (2016)
10. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: EuroSec. pp. 192–206. Springer (2008)
11. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg, Germany, Accra Beach, Barbados (Feb 23–26, 2009)
12. Chowdhury, A.R., Wang, C., He, X., Machanavajjhala, A., Jha, S.: Outis: Crypto-Assisted Differential Privacy on Untrusted Servers. arXiv preprint arXiv:1902.07756 pp. 1–30 (2019)
13. Crockett, E., Peikert, C., Sharp, C.: Alchemy: A language and compiler for homomorphic encryption made easy. In: CCS (2018)
14. Deshpande, A., Ives, Z.G., Raman, V.: Adaptive Query Processing. Foundations and Trends in Databases **1**(1), 1–140 (2007)
15. Dinur, I., Nissim, K.: Revealing information while preserving privacy. In: Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 202–210. PODS '03, ACM, New York, NY, USA (2003)
16. Dwork, C.: Differential privacy. International Colloquium on Automata, Languages and Programming pp. 1–12 (2006). https://doi.org/10.1007/11787006
17. El-Hindi, M., Heyden, M., Binnig, C., Ramamurthy, R., Arasu, A., Kossmann, D.: Blockchaindb-towards a shared database on blockchains. In: Proceedings of the 2019 International Conference on Management of Data. pp. 1905–1908. ACM (2019)
18. Eskandarian, S., Zaharia, M.: An Oblivious General-Purpose SQL Database for the Cloud. arXiv preprint 1710.00458 (2017)
19. Ge, C., He, X., Ilyas, I.F., Machanavajjhala, A.: Apex: Accuracy-aware differentially private data exploration. In: Proceedings of the 2019 International Conference on Management of Data. pp. 177–194. SIGMOD '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3299869.3300092
20. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 626–645. Springer (2013)
21. Gentry, C.: Fully homomorphic eneryption using ideal lattices. In: Proceedings of the 41st annual ACM symposium on Theory of computing. pp. 169–178. ACM (2009)
22. Gentry, C., Halevi, S., Raykova, M., Wichs, D.: Outsourcing private RAM computation. In: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. pp. 404–413. IEEE (2014)
23. Gupta, D., Mood, B., Feigenbaum, J., Butler, K., Traynor, P.: Using Intel Software Guard Extensions for Efficient Two-Party Secure Function Evaluation. 4th Workshop on Encrypted Computing and Applied Homomorphic Cryptography - WAHC'16 (February) (2016)

24. He, Z., Wong, W.K., Kao, B., Cheung, D.W.L., Li, R., Yiu, S.M., Lo, E.: SDB: a secure query processing system with data interoperability. VLDB **8**(12), 1876–1879 (2015). https://doi.org/2150-8097/15/08
25. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003)
26. Johnson, N., Near, J.P., Song, D.: Towards practical differential privacy for sql queries. Proc. VLDB Endow. **11**(5), 526–539 (Jan 2018). https://doi.org/10.1145/3187009.3177733
27. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
28. Kotsogiannis, I., Tao, Y., Machanavajjhala, A., Miklau, G., Hay, M.: Architecting a Differentially Private SQL Engine. In: CIDR (2019)
29. Kotsogiannis, I., Tau, Y., He, X., Fanaeepour, M., Machanavajjhala, A., Hay, M., Miklau, G.: PrivateSQL: A Differentially Private SQL Engine. Proceedings of the VLDB Endowment **12**(12) (2019)
30. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., Stoica, I.: Learning to optimize join queries with deep reinforcement learning. arXiv preprint arXiv:1808.03196 (2018)
31. Liu, C., Wang, X.S., Nayak, K., Huang, Y., Shi, E.: ObliVM : A Programming Framework for Secure Computation. Oakland pp. 359–376 (2015). https://doi.org/10.1109/SP.2015.29
32. Marcus, R., Papaemmanouil, O.: Deep reinforcement learning for join order enumeration. In: Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. p. 3. ACM (2018)
33. Markl, V., Lohman, G.M., Raman, V.: LEO: An autonomic query optimizer for DB2. IBM Systems Journal **42**(1), 98–106 (2003)
34. McGregor, A., Mironov, I., Pitassi, T., Reingold, O., Talwar, K., Vadhan, S.: The limits of two-party differential privacy. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science. pp. 81–90. FOCS '10, IEEE Computer Society, Washington, DC, USA (2010). https://doi.org/10.1109/FOCS.2010.14
35. McSherry, F.D.: Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. pp. 19–30. SIGMOD '09, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1559845.1559850
36. Menon, P., Mowry, T.C., Pavlo, A.: Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last. Proceedings of the VLDB Endowment **11**(1), 1–13 (2017)
37. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: CRYPTO, pp. 126–142. Springer (2009)
38. Nathan, S., Govindarajan, C., Saraf, A., Sethi, M., Jayachandran, P.: Blockchain meets database: Design and implementation of a blockchain relational database (2019)
39. Neumann, T., Leis, V.: Compiling database queries into machine code. IEEE Data Eng. Bull. **37**(1), 3–11 (2014)
40. Ortiz, J., Balazinska, M., Gehrke, J., Keerthi, S.S.: Learning state representations for query optimization with deep reinforcement learning. arXiv preprint arXiv:1803.08604 (2018)
41. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)

42. Pavlo, A., Angulo, G., Arulraj, J., Lin, H., Lin, J., Ma, L., Menon, P., Mowry, T.C., Perron, M., Quah, I., Others: Self-Driving Database Management Systems. In: CIDR. vol. 4, p. 1 (2017)
43. Pirk, H., Funke, F., Grund, M., Neumann, T., Leser, U., Manegold, S., Kemper, A., Kersten, M.: Cpu and cache efficient management of memory-resident databases. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE). pp. 14–25. IEEE (2013)
44. Popa, R., Redfield, C.: CryptDB: protecting confidentiality with encrypted query processing. SOSP pp. 85–100 (2011). https://doi.org/10.1145/2043556.2043566
45. Rajan, A., Qin, L., Archer, D.W., Boneh, D., Lepoint, T., Varia, M.: Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In: Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies. p. 49. ACM (2018)
46. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. SIGMOD pp. 23–34 (1979). https://doi.org/10.1145/582095.582099
47. Suresh, M., She, Z., Wallace, W., Lahlou, A., Rogers, J.: Kloakdb: A platform for analyzing sensitive data with k-anonymous query processing. CoRR **abs/1904.00411** (2019), http://arxiv.org/abs/1904.00411
48. Trummer, I., Koch, C.: Multi-objective parametric query optimization. Proceedings of the VLDB Endowment **8**(3), 221–232 (2014)
49. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing Analytical Queries over Encrypted Data. Proc. VLDB Endow. **6**(5), 289–300 (mar 2013). https://doi.org/10.14778/2535573.2488336
50. Volgushev, N., Schwarzkopf, M., Getchell, B., Varia, M., Lapets, A., Bestavros, A.: Conclave: secure multi-party computation on big data. In: European Conference on Computer Systems (2019)
51. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical Private Queries on Public Data. In: NSDI. pp. 299–313 (2017)
52. Wang, W., Zhang, M., Chen, G., Jagadish, H.V., Ooi, B.C., Tan, K.L.: Database meets deep learning: challenges and opportunities. ACM SIGMOD Record **45**(2), 17–22 (2016)
53. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: CCS (2017)
54. Wang, X.S., Nayak, K., Liu, C., Chan, T.H.H., Shi, E., Stefanov, E., Huang, Y.: Oblivious Data Structures. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14 pp. 215–226 (2014). https://doi.org/10.1145/2660267.2660314
55. Wei, Z., Leck, U., Link, S.: Entity integrity, referential integrity, and query optimization with embedded uniqueness constraints. In: ICDE (2019)
56. Wong, W.K., Kao, B., Cheung, D.W.L., Li, R., Yiu, S.M.: Secure query processing with data interoperability in a cloud database environment. In: SIGMOD. pp. 1395–1406. ACM (2014)
57. Yao, A.C.: Protocols for secure computations. In: FOCS. pp. 160–164. IEEE (1982)
58. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986)
59. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 863–880. IEEE (2017)

60. Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I.: Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In: NSDI. pp. 283–298 (2017)