DP-Sync: Hiding Update Patterns in Secure Outsourced Databases with Differential Privacy

Chenghong Wang
Duke University
chwang@cs.duke.edu

Johes Bater
Duke University
johes.bater@duke.edu

Kartik Nayak
Duke University
kartik@cs.duke.edu

Ashwin Machanavajjhala
Duke University
ashwin@cs.duke.edu

ABSTRACT

In this paper, we consider privacy-preserving update strategies for secure outsourced growing databases. Such databases allow append-only data updates on the outsourced data structure while analysis is ongoing. Despite a plethora of solutions to securely outsource database computation, existing techniques do not consider the information that can be leaked via update patterns. To address this problem, we design a novel secure outsourced database framework for growing data, DP-Sync, which interoperate with a large class of existing encrypted databases and supports efficient updates while providing differentially-private guarantees for any single update. We demonstrate DP-Sync's practical feasibility in terms of performance and accuracy with extensive empirical evaluations on real world datasets.

CCS CONCEPTS

• **Security and privacy → Data anonymization and sanitization**; **Management and querying of encrypted data**.

KEYWORDS

Update pattern; Encrypted database; Differential privacy

ACM Reference Format:
Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2021. DP-Sync: Hiding Update Patterns in Secure Outsourced Databases with Differential Privacy. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3457306

1 INTRODUCTION

In the last couple of decades, organizations have been rapidly moving towards outsourcing their data to the cloud. While this brings in inherent advantages such as lower costs, high availability, and ease of maintenance, this also results in privacy concerns for organizations. Hence, many solutions leverage cryptography and/or techniques such as differential privacy to keep the data private while simultaneously allowing secure query processing on this data [6, 9, 10, 25, 34, 42, 45, 68, 70, 76]. However, while most practical systems require us to maintain dynamic databases that support updates, research in the space of private database systems has focused primarily on static databases [6, 9, 25, 42, 45, 54]. There have been a few works which consider private database updates and answering queries on such dynamic databases [1, 29, 34, 43, 50, 70, 76]. However, none of these works consider the privacy of *when* a database is updated. In this work, we consider the problem of hiding such database update patterns.

Let us consider the following example where an adversary can breach privacy by using the timing information of updates. Consider an IoT provider that deploys smart sensors (i.e., security camera, smart bulb, WiFi access point, etc.) for a building. The provider also creates a database to back up the sensors' event data. For convenience, the database is maintained by the building administrator, but is encrypted to protect the privacy of people in the building. By default, the sensor will backup immediately when any new sensor event (i.e. a new connection to WiFi access point) occurs. Suppose that at a certain time, say 7:00 AM, only one person entered the building. Afterwards, the building admin observes three backup requests posted at times 7:00:00, 7:00:10, 7:00:20, respectively. Also suppose that the admin has access to additional non-private building information, such as that floor 3 of this building is the only floor which has three sensors with a 10 second walking delay (for an average person). Then, by looking at the specific times of updates (10 second delays) and the number of updates, the building admin can learn private information about the activity (i.e. the person went to the 3rd floor), without ever having to decrypt the stored data. This type of attack generalizes to any event-driven update where the event time is tied to the data upload time. In order to prevent such attacks, we must decouple the relationship between event and upload timings.

There are two straightforward solutions to solve this concern. The first option is to never upload any sensor data at all. While such a solution does provide necessary privacy, it does not provide us with the functionality of a database that supports updates. If an employee from the IoT provider queries the database to obtain, for example, the number of sensor events happened in a day, she will receive an inaccurate result. A second option is to back up the sensor event record at each time unit, independent of whether the sensor event actually occurred or not. Again, this does solve the privacy concern since the update does not depend on the sensor events at all. However, this introduces performance concerns: If sensor events occur relatively infrequently, then most updates are likely to be empty, or "dummy", updates, meaning that the provider will waste valuable resources on unnecessary computation. The above examples illustrate the 3-way trade-off between privacy, accuracy, and performance in the database synchronization problem. Each of the three approaches we discussed, immediate synchronization,

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457306

no synchronization, and every time unit synchronization, achieves precisely two of the three properties, but not the third.

In this work, we build DP-Sync, an append-only database outsourced by a data owner to one or more untrusted cloud service providers (server). In addition, a trusted analyst, possibly the owner, is allowed to query the database at any point in time. To ensure consistency of the outsourced data, the owner synchronizes local records and updates the outsourced data. However, making updates on outsourced data structures may leak critical information. For instance, the server can potentially detect the size of synchronized records [4, 17, 51, 69]. Cryptographic techniques such as ORAMs [76] or structured encryption [43] prevent leaking critical information on updates. However, all these methods are primarily designed to ensure that when an update occurs, attackers cannot learn sensitive information by observing changes in the outsourced data structure and not *when* these changes happen. If the adversary/cloud server has access to the exact time of the updates, even if the system employs the techniques described above to protect individual updates, it can still result in privacy breaches of owner's data. The goal of DP-Sync is to prevent such an update pattern leakage while still being performant and accurate. We now elaborate on our key contributions:

Private update synchronization. We introduce and formalize the problem of synchronizing updates to an encrypted database while hiding update patterns. Our goal is to provide a bounded differentially-private guarantee for any single update made to the cloud server. To navigate the 3-way trade-off between privacy, accuracy, and performance, we develop a framework where users can obtain customizable properties by modifying these parameters.

Differentially-private update synchronization algorithms. We provide two novel synchronization algorithms, DP-Timer and DP-ANT, that can obtain such trade-offs. The first algorithm, DP-Timer algorithm, parameterized by time T, synchronizes updates with the server every T time. Thus, for a fixed parameter T, to achieve a high amount of privacy, the algorithm asymptotes to never update the server (and hence, will not achieve accuracy). As we weaken our privacy, we can gracefully trade it for better accuracy. Similarly, by modifying T, we can obtain different trade-offs between accuracy and performance. The second algorithm DP-ANT, parameterized by a threshold θ, synchronizes with the server when there are approximately θ records to update. Thus, for a fixed parameter θ, when achieving high accuracy, the algorithm asymptotes to updating the server at each time unit and thus, poor performance. By reducing the accuracy requirement, we can gracefully trade it for better performance. Moreover, we can modify the parameter θ to obtain different trade-offs. Comparing the two algorithms, DP-ANT dynamically adjusts its synchronization frequency depending on the rate at which new records are received while DP-Timer adjusts the number of records to be updated each time it synchronizes.

Interoperability with existing encrypted databases. We design our update synchronization framework such that it can interoperate with a large class of existing encrypted database solutions. To be concrete, we provide the precise constraints that should be satisfied by the encrypted database to be compatible with DP-Sync, as well as classify encrypted databases based on what they leak about their inputs.

Evaluating DP-Sync with encrypted databases. We implement multiple instances of our synchronization algorithms with two encrypted database systems: Cryptϵ and ObliDB. We evaluate the performance of the resulting system and the trade-offs provided by our algorithms on the New York City Yellow Cab and New York City Green Boro taxi trip record dataset. The evaluation results show that our DP strategies provide bounded errors with only a small performance overhead, which achieve up to 520x better in accuracy than never update method and 5.72x improvement in performance than update every time approach.

2 PROBLEM STATEMENT

The overarching goal of this work is to build a generic framework for secure outsourced databases that limits information leakage due to database updates. We must ensure that the server, which receives outsourced data, cannot learn unauthorized information about that data, i.e., the true update history. We achieve this by proposing private synchronization strategies that the owner may use to hide both how many records are currently being outsourced and when those records were originally inserted. Though there are simple methods that effectively mask the aforementioned update history, significant tradeoffs are required. For example, one may simply prohibit the owner from updating the outsourced database, or force them to update at predefined time intervals, regardless of whether they actually need to. Though both approaches ensure that the true update history is masked, they either entirely sacrifice data availability on the outsourced database or incur a significant performance overhead, respectively. Navigating the design space of private synchronization protocols requires balancing a 3-way tradeoff between privacy, accuracy, and performance. To tackle this challenge, we formalize our research problems as follows:

- Build a generic framework that ensures an owner's database update behavior adheres to private data synchronization policies, while supporting existing encrypted databases.
- Design private synchronization algorithms that (i) hide an owner's update history and (ii) balance the trade-off between privacy, accuracy and efficiency.

In addition to the research problems above, we require our design to satisfy the following principles.

P1-Private updates with a differentially private guarantee. The proposed framework ensures that any information about a single update leaked to a semi-honest server is bounded by a differentially private guarantee. We formally define this in Definition 5.

P2-Configurable privacy, accuracy and performance. Rather than providing a fixed configuration, we develop a framework where users can customize the level of privacy, accuracy, and performance. For example, users can trade privacy for better accuracy and/or improved performance.

P3-Consistent eventually. The framework and synchronization algorithms should allow short periods of data inconsistency between the logical (held by the owner) and the outsourced (held by the server) databases. To abstract this guarantee, we follow the principles in [18] and define the concept of *consistent eventually* for our framework as follows. First, the outsourced database can temporarily lag behind the logical database by a number of records.

However, once the owner stops receiving new data, there will eventually be no logical gaps. Second, all data should be updated to the server in the same order in which they were received by the owner. In some cases, the consistent eventually definition can be relaxed by removing the second condition. In this work, we implement our framework to satisfy the definition without this relaxation.

P4-Interoperable with existing encrypted database solutions The framework should be interoperable with existing encrypted databases. However, there are some constraints. First, the encrypted databases should encrypt each record independently into a separate ciphertext. Schemes that encrypt data into a fixed size indivisible ciphertext (i.e., the ciphertext batching in Microsoft-SEAL [59]) do not qualify. Since batching may reveal additional information, such as the maximum possible records per batch. Second, the database should support or be extensible to support data updates (insertion of new records). Thus, a completely static scheme [74] is incompatible. In addition, our security model assumes the database's update leakage can be profiled as a function solely related to the update pattern. Therefore, dynamic databases with update protocol leaks more than the update pattern [52, 65] are also ineligible. Third, the corresponding query protocol should not reveal the exact access pattern [38] or query volume [53] information. Despite these constraints, our framework is generic enough to support a large number of existing encrypted databases such as [2, 4, 9, 12, 17, 21, 25, 34, 37, 52, 79, 81]. Later, in Section 6, we provide a detailed discussion on the compatibility of existing encrypted database schemes with DP-Sync.

3 DP-SYNC DESCRIPTION

In this section, we introduce DP-Sync, a generic framework for encrypted databases that hides update pattern leakage. The framework does not require changes to the internal components of the encrypted database, but rather imposes restrictions on the owner's synchronization strategy. We illustrate the general architecture and components of DP-Sync in Section 3.1 and Section 3.2, respectively.

3.1 Framework Overview

Our framework consists of an underlying encrypted database with three basic protocols, edb = (Setup, Update, Query), a synchronization strategy Sync, and a local cache σ. Our framework also defines a dummy data type that, once encrypted, is indistinguishable from the true outsourced data. The local cache σ is a lightweight storage structure that temporarily holds data received by the owner, while Sync determines when the owner needs to synchronize the cached data to the server (poses an update) and how many records are required for each synchronization. DP-Sync makes no changes to the edb and will fully inherit all of its cryptographic primitives and protocols. Figure 1 illustrates the general workflow of DP-Sync.

Our proposed framework operates as follows. Initially, the owner sets up a synchronization strategy Sync and a local cache σ, then authorizes the analyst. The owner starts with an initial database with which it invokes Sync to obtain a set of records, γ_0, to be outsourced first. The owner then runs the setup protocol (edb.Setup) with γ_0 as the input. An initial outsourced data structure is then created and stored on the server. For each subsequent time step, whenever the Sync algorithm signals the need for synchronization, the owner reads relevant records from the cache and inputs them to the update

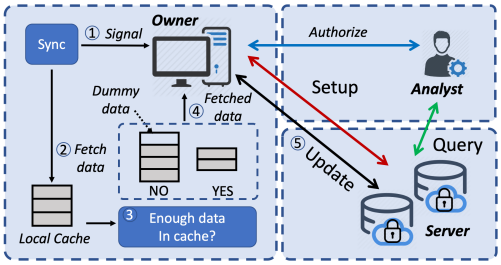


Figure 1: Overview of DP-Sync's architecture.

protocol (edb.Update) to update the outsourced structure. When there is less data than needed, the owner inputs sufficiently many dummy records in addition to the cached data.

Since all records are encrypted, the server does not know which records are dummy records and which are true records. The outsourced data structure will only change if the owner runs the update protocol, in other words, if Sync does not signal, then the outsourced structure remain unchanged. The analyst independently creates queries and runs the query protocol (edb.Query) to make requests. The server evaluates each query and returns the result to analyst. For simplicity, we assume that all queries arrive instantaneously and will be executed immediately.

3.2 Framework Components

3.2.1 Local cache. The local cache is an array $\sigma[1, 2, 3...]$ of memory blocks, where each $\sigma[i]$ represents a memory block that stores a record. By default, the local cache in DP-Sync is designed as a FIFO queue that supports three types of basic operations:

(1) **Get cache length (**$\mathsf{len}(\sigma)$**).** The operation calculates how many records are currently stored in the local cache, and returns an integer count as the result.
(2) **Write cache (**$\mathsf{write}(\sigma, r)$**).** The write cache operation takes as input a record r and appends the record to the end of the current local cache, denoted as $\sigma \parallel r \leftarrow \mathsf{write}(\sigma, r)$.
(3) **Read cache (**$\mathsf{read}(\sigma, n)$**).** Given a read size n, if $n \leq \mathsf{len}(\sigma)$, the operation pops out the first n records, $\sigma\,[1, ..., n]$, in the local cache. Otherwise, the operation pops all records in σ along with a number of dummy records equal to $|n - \mathsf{len}(\sigma)|$.

The FIFO mode ensures all records are uploaded in the same order they were received by the owner. In fact, the local cache design is flexible and can be replaced with other design scenarios. For example, it can be designed with LIFO mode if the analyst is only interested in the most recently received records.

3.2.2 Dummy records. Dummy records have been widely used in recent encrypted database designs [3, 6, 7, 9, 34, 44, 54, 64] to hide access patterns, inflate the storage size and/or distort the query response volume. In general, dummy data is a special data type that cannot be distinguished from real outsourced data when encrypted. Moreover, the inclusion of such dummy data does not affect the correctness of query results.

3.2.3 Synchronization strategy. The synchronization strategy Sync takes on the role of instructing the owner how to synchronize the local data. It decides when to synchronize their local records and guides the owner to pick the proper data to be synchronized. We explain in detail the design of Sync in section 5.

4 DP-SYNC MODEL

In this section, we describe the abstract model of DP-Sync as a secure outsourced growing database, including the key definitions (Section 4.1), security model (Section 4.3), privacy semantics (Section 4.4) , and evaluation metrics (Section 4.5).

4.1 Secure Outsourced Growing Database

We begin by introducing the main concepts of outsourced growing databases and the notations used in this work. A summary of key concepts and notations is provided in Table 1.

$\mathcal{D}_0$	initial logical database
u_t	logical update at time t; can be a single record or null
γ_t	real update at t, can be a set of records or null
$\mathcal{D}_t$	logical database at time t; $\mathcal{D}_t = \{\mathcal{D}_0 \cup u_1 \cup u_2 \ldots \cup u_t\}$
$\mathcal{D}$	the logical instance of a growing database
$\mathcal{DS}_t$	physical data on the server at time t
$\mathcal{DS}$	the physical data on the server over time
q_t	list of queries at time t
Q	set of queries over time, where $Q = \{q_t\}_{t\geq 0}$
SOGDB	secure outsourced growing database.

Table 1: Summary of notations.

A growing database consists of an initial database $\mathcal{D}_0$ and a set of logical updates $U = \{u_t\}_{t\geq 0}$ to be appended to $\mathcal{D}_0$, where $u_t \in U$ is either a single record or $\emptyset$. The former corresponds to the data received at t, while $\emptyset$ indicates no data arrives. We consider the case where at most one record arrives at any time unit for the sake of simplicity, however this can be generalized to the case where multiple records arrive in one time unit. We define the growing database as $\mathcal{D} = \{\mathcal{D}_t\}_{t\geq 0}$, where $\mathcal{D}_t$ is the logical database at time t, and $\mathcal{D}_t = \{\mathcal{D}_0 \cup u_1 \cup u_2 \ldots \cup u_t\}$. We stress that when we say a growing database has length L, it means that there could be up to L logical updates in U, that is $|U| = L$. We consider databases that support select (search), project, join and aggregations. We use $Q = \{q_t\}_{t\geq 0}$ to denote the set of queries evaluated over a growing database, where q_t is the query over $\mathcal{D}_t$.

There are three entities in the secure outsourced data model: the owner, the server, and the analyst. The owner holds a logical database, encrypts and outsources it to the server, and continually updates the outsourced structure with new data. The server stores the outsourced structure, on which it processes queries sent by an authorized analyst. For growing databases, all potential updates posted by the owner will be insertions only. We denote the records to be updated each time as γ_t, which can be a collection of records, or empty (no update has occurred). We use $\mathcal{DS} = \{\mathcal{DS}_t\}_{t\geq 0}$ to represent the outsourced structure over time, where $\mathcal{DS}_t$ is an instance of outsourced structure at time t. Typically, an instance of the outsourced structure contains a set of encrypted records as well as an optional secure data structure (i.e., secure index [22]). We now define the syntax of a secure outsourced database as follows:

Definition 1 (Secure Outsourced Growing Database). A secure outsourced database is a suite of three protocols and a polynomial-time algorithm with the following specification:

$(\perp, \mathcal{DS}_0, \perp) \leftarrow \prod_{\mathsf{Setup}}((\lambda, \mathcal{D}_0), \perp, \perp)$: is a protocol that takes as input a security parameter λ, and an initial database $\mathcal{D}_0$ from the owner. The protocol sets up the internal states of the SOGDB system and outputs an outsourced database $\mathcal{DS}_0$ to the server.

$(\perp, \mathcal{DS}'_t, \perp) \leftarrow \prod_{\mathsf{Update}}(\gamma, \mathcal{DS}_t, \perp)$: is a protocol that takes an outsourced structure $\mathcal{DS}_t$ from the server, and a collection of records γ from the owner, which will be inserted into the outsourced data. The protocol updates the outsourced structure and outputs the updated structure $\mathcal{DS}'_t$ to server.

$(\perp, \perp, a_t) \leftarrow \prod_{\mathsf{Query}}(\perp, \mathcal{DS}_t, q_t)$: is a protocol that takes an outsourced database $\mathcal{DS}_t$ from the server and a set of queries q_t from the analyst. The protocol reveals the answers a_t to the analyst.

$\mathsf{Sync}(\mathcal{D})$: is a (possibly probabilistic) stateful algorithm that takes as input a logical growing database $\mathcal{D}$. The protocols signals the owner to update the outsourced database from time to time, depending on its internal states.

The notation $(\mathsf{c}_{\mathsf{out}}, \mathsf{s}_{\mathsf{out}}, \mathsf{a}_{\mathsf{out}}) \leftarrow \mathsf{protocol}(\mathsf{c}_{\mathsf{in}}, \mathsf{s}_{\mathsf{in}}, \mathsf{a}_{\mathsf{in}})$ is used to denote a protocol among the owner, server and analyst, where c_{in}, s_{in}, and a_{in} denote the inputs of the owner, server and analyst, respectively, and $\mathsf{c}_{\mathsf{out}}$, $\mathsf{s}_{\mathsf{out}}$, and $\mathsf{a}_{\mathsf{out}}$ are the outputs of the owner, server and analyst. We use the symbol $\perp$ to represent nothing input or output. We generally follow the abstract model described in [53]. However, the above syntax refers to the *dynamic* setting, where the scheme allows the owner to make updates (appending new data) to the outsourced database. The *static* setting [53] on the other hand, allows no updates beyond the setup phase. We assume that each record from the logical database is atomically encrypted in the secure outsourced database. The outsourced database may, in addition, store some encrypted dummy records. This model is also referred to as *atomic database* [53]. In addition, we assume that the physical updates can be different from the logical updates. For instance, an owner may receive a new record every 5 minutes, but may choose to synchronize once they received up to 10 records.

4.2 Update Pattern Leakage

We now introduce a new type of volumetric leakage [11] called *update pattern* leakage. In general, an update pattern consists of the owner's entire update history transcript for outsourcing a growing database. It may include information about the number of records outsourced and their insertion times.

Definition 2 (Update Pattern). Given a growing database $\mathcal{D}$ and a SOGDB scheme Σ, the update pattern of Σ when outsourcing $\mathcal{D}$ is $\mathsf{UpdtPatt}(\Sigma, \mathcal{D}) = \{\mathsf{UpdtPatt}_t(\Sigma, \mathcal{D}_t)\}_{t\in\mathbb{N}^+\wedge t\in t'}$, with:

$$\mathsf{UpdtPatt}_t(\Sigma, \mathcal{D}_t) = (t, |\gamma_t|)$$

where $t' = \{t'_1, t'_2, ..., t'_k\}$ denotes the set of timestamps t'_i when the update occurs, and γ_t denotes the set of records synchronized to the outsourcing database at time t. We refer to the total number of records $|\gamma_t|$ updated at time t as the corresponding update volume.

Example 4.1. *Assume an outsourced database setting where the owner synchronizes 5 records to the server every 30 minutes and the minimum time span is 1 minute. Then the corresponding update pattern can be written as* $\{(0, 5), (30, 5), (60, 5), (90, 5)...\}$.

4.3 Privacy Model

Recall that in DP-Sync, there are three parties: the owner (who outsources local data), the server (who stores outsourced data),

and the analyst (who queries outsourced data). Our adversary is the server, whom we want to prevent from learning unauthorized information about individuals whose records are stored in the local data. We assume a semi-honest adversary, meaning that the server will faithfully follow all DP-Sync protocols, but may attempt to learn information based on update pattern leakage.

Update pattern leakage may reveal the number of records inserted at each time step, as the server can keep track of the insertion history. To ensure privacy, we need to strictly bound the information the server can learn. In this section, we formally define the privacy guarantee for update pattern leakage in DP-Sync.

Definition 3 (ϵ-Differential Privacy [33]). A randomized mechanism $\mathcal{M}$ satisfies ϵ-differential privacy (DP) if for any pair of neighboring databases D and D' that differ by adding or removing one record, and for any $O \subseteq \mathcal{O}$, where $\mathcal{O}$ is the set of all possible outputs, it satisfies:

$$\Pr\left[\mathcal{M}(D) \in O\right] \leq e^{\epsilon} \Pr\left[\mathcal{M}(D') \in O\right]$$

With DP, we can provide provable, mathematical bounds on information leakage. This allows us to quantify the amount of privacy leaked to the server in our scheme.

Definition 4 (Neighboring growing databases). $\mathcal{D}$ and $\mathcal{D}'$ are neighboring growing databases if for some parameter $\tau \geq 0$, the following holds: (i) $\mathcal{D}_t = \mathcal{D}'_t$ for $t \leq \tau$ and (ii) $\mathcal{D}_t$ and $\mathcal{D}'_t$ differ by the addition or removal of a single record when $t > \tau$.

In practice, Definition 4 defines a pair of growing databases that are identical at any time before $t = \tau$, and differ by at most one record at any time after $t = \tau$. After defining neighboring growing databases, we now follow the definition of event level DP [33] under continual observation, and generalize it to SOGDB setting. This allows us to describe and bound the privacy loss due to update pattern leakage in DP-Sync.

Definition 5 (SOGDB with DP update pattern). Let $\mathcal{L}_{\mathsf{U}}$ be the update leakage profile for a SOGDB system Σ. The SOGDB Σ has a differentially-private (DP) update pattern if $\mathcal{L}_{\mathsf{U}}$ can be written as:

$$\mathcal{L}_{\mathsf{U}}(\mathcal{D}) = \mathcal{L}'\left(\mathsf{UpdtPatt}(\Sigma, \mathcal{D})\right)$$

where $\mathcal{L}'$ is a function, and for any two neighboring growing databases $\mathcal{D}$ and $\mathcal{D}'$, and any $O \subseteq \mathcal{O}$, where $\mathcal{O}$ is the set of all possible update patterns, $\mathcal{L}_{\mathsf{U}}(\mathcal{D})$ satisfies:

$$\Pr\left[\mathcal{L}_{\mathsf{U}}(\mathcal{D}) \in O\right] \leq e^{\epsilon} \cdot \Pr\left[\mathcal{L}_{\mathsf{U}}(\mathcal{D}') \in O\right]$$

Definition 5 ensures that, for any SOGDB, if the update leakage is a function defined as $\mathsf{UpdtPatt}(\Sigma, \mathcal{D})$, then the information revealed by any single update is differentially private. Moreover, if each update corresponds to a different entity's (owner's) record then privacy is guaranteed for each entity. The semantics of this privacy guarantee are discussed further in Section 4.4. Note that although Definition 5 provides information theoretic guarantees on update pattern leakage, the overall security guarantee for DP-Sync depends on the security of the underlying encrypted database scheme. If the encrypted database provides information theoretic guarantees, then DP-Sync also provides information theoretic DP guarantees. If the encrypted database is semantically secure, then DP-Sync provides computational differential privacy, i.e., Definition 5 only holds for a computationally bounded adversary.

4.4 Privacy Semantics

In this section, we explore the privacy semantics of Definition 5 from the perspective of disclosing secrets to adversaries. To achieve this, we utilize the Pufferfish [56] framework to interpret the privacy semantics. One can show that if a SOGDB satisfies Definition 5, then for any single user u, and any pair of mutually exclusive secrets of u's record that span a single time step, say $\phi_u(t)$, and $\phi'_u(t)$ (an example of such pair of secrets is whether u's data was inserted or not to an growing database), the adversary's posterior odds of $\phi_u(t)$ being true rather than $\phi'_u(t)$ after seeing the SOGDB's update pattern leakage is no larger than the adversary's prior odds times e^{ϵ}. Note that this strong privacy guarantee holds only under the assumption that the adversary is unaware of the possible correlation between the user's states across different time steps. Recent works [55, 61, 82] have pointed out that with knowledge of such correlations, adversaries can learn sensitive properties even from the outputs of differentially private algorithms. Nevertheless, it is still guaranteed that the ratio of the adversary's posterior odds to the prior odds is bounded by $e^{l \times \epsilon}$ [19, 75], where l is the maximum possible number of records in a growing database that corresponds to a single user. The actual privacy loss may be much smaller depending on the strength of the correlation known to the adversary [19, 75]. We emphasize that our algorithms are designed to satisfy Definition 5 with parameter ϵ, while simultaneously satisfying all the above privacy guarantees, though the privacy parameters may differ. Thus, for the remainder of the paper, we focus exclusively on developing algorithms that satisfy Definition 5.

4.5 Evaluation Metrics

4.5.1 Efficiency metrics. To evaluate SOGDB's efficiency, we use two metrics: (1) query execution time (QET) or the time to run $\prod_{\mathsf{Query}}$ and (2) the number of encrypted records outsourced to the server. Note that in some cases the QET and the number of outsourced data may be positively correlated, as QET is essentially a linear combination of the amount of outsourced data.

4.5.2 Accuracy metrics. Ideally, the outsourced database should contain all records from the logical database at every point in time. In practice, for efficiency and privacy reasons, an owner can only sync records intermittently. This temporary data inconsistency may result in some utility loss. To measure this utility loss, we propose two accuracy metrics as follows:

Logical gap. For each time t, the logical gap between the outsourced and logical database is defined as the total number of records that have been received by the owner but have not been outsourced to the server. We denote it as $LG(t) = \left|\mathcal{D}_t - \mathcal{D}_t \cap \hat{\mathcal{D}}_t\right|$, where $\hat{\mathcal{D}}_t = \{\gamma_0 \cup \gamma_1 \cup ...\gamma_t\}$ denotes the set of records that have been outsourced to the server until time t. Intuitively, a big logical gap may cause large errors on queries over the outsourced database.

Query error. For any query q_t, query error $QE(q_t)$ is the L1 norm between the true answer over the logical database and the result obtained from $\prod_{\mathsf{Query}}$. Thus, $QE(q_t) = |\prod_{\mathsf{Query}}(\mathcal{DS}_t, q_t) - q_t(\mathcal{D}_t)|$. While query error is usually caused by the logical gap, different types of query results may be affected differently by the same logical gap. Hence, we use query error as an independent accuracy metric.

Group	Privacy	Logical gap	Total number of outsourced records
SUR	∞-DP	0	$\|\mathcal{D}_t\|$
OTO	0-DP	$\|\mathcal{D}_t\| - \|\mathcal{D}_0\|$	$\|\mathcal{D}_0\|$
SET	0-DP	0	$\|\mathcal{D}_0\| + t$
DP-Timer	ϵ-DP	$c^t_{t*} + O(\frac{2\sqrt{k}}{\epsilon})$	$\|\mathcal{D}_t\| + O(\frac{2\sqrt{k}}{\epsilon}) + \eta$
ANT	ϵ-DP	$c^t_{t*} + O(\frac{16 \log t}{\epsilon})$	$\|\mathcal{D}_t\| + O(\frac{16 \log t}{\epsilon}) + \eta$

Table 2: Comparison of synchronization strategies. c^t_{t*} counts the number of record received since last update, k denotes the number of synchronization posted so far, f is cache flush span, s is the cache flush size, and $\eta = s\lfloor t/f \rfloor$.

5 RECORD SYNCHRONIZING ALGORITHMS

In this section, we discuss our secure synchronization strategies, including naïve methods (section 5.1) and DP based strategies (section 5.2). A comparison concerning their accuracy, performance, and privacy guarantees is provided in Table 2.

5.1 Naïve Synchronization Strategies

We start with three naïve methods illustrated as follows:

(1) *Synchronize upon receipt (SUR).* The SUR policy is the most adopted strategy in real-world applications, where the owner synchronizes new data to the server as soon as it is received, and remains inactive if no data is received.
(2) *One time outsourcing (OTO).* The OTO strategy only allows the owner to synchronize once at initial stage $t = 0$. From then on, the owner is offline and no data is synchronized.
(3) *Synchronize every time (SET).* The SET method requires the owner to synchronize at each time unit, independent of whether a new record is to be updated. More specifically, for any time t, if $u_t \neq \emptyset$, the owner updates the received record. If $u_t = \emptyset$, owner updates a dummy record to server.

Given a growing database $\mathcal{D} = \{D_0, U\}$. SUR ensures any newly received data is immediately updated into the outsourcing database, thus there is no logical gap at any time. Besides, SUR does not introduce dummy records. However, SUR provides zero privacy guarantee as it leaks the exact update pattern. OTO provides complete privacy guarantees for the update pattern but achieves zero utility for all records received by the owner after $t = 0$. Thus the logical gap for any time equals to $|\mathcal{D}_t| - |\mathcal{D}_0|$. Since OTO only outsources the initial records, the total amount of data outsourced by OTO is bounded by $O(|\mathcal{D}_0|)$. SET provides full utility and complete privacy for any record, and ensures 0 logical gap at any time. However, as a cost, SET outsources a large amount of dummy records, resulting in significant performance overhead. In addition, all of the methods provide fixed privacy, performance, and/or utility. As such, none of them comply with the P3 design principle. OTO also violates P2 as no data is outsourced after initialization.

5.2 Differentially Private Strategies

5.2.1 Timer-based synchronization (DP-Timer). The timer-based synchronization method, parameterized by T and ϵ, performs an update every T time units with a varying number of records. The detailed algorithm is described in Algorithm 1.

Algorithm 1 Timer Method (DP-Timer)

Input: growing database $\mathcal{D} = \{\mathcal{D}_0, U\}$, privacy budget ϵ, timer T, and local cache σ.

1: $c \leftarrow 0, t^* \leftarrow 0$
2: $\gamma_0 \leftarrow \text{Perturb}(|\mathcal{D}_0|, \epsilon, \sigma)$
3: Signal the owner to run $\prod_{\text{Setup}}(\gamma_0)$.
4: **for** $t \leftarrow 1, 2, 3, \ldots$ **do**
5: **if** $u_t \neq \emptyset$ **then**
6: write(σ, u_t) (store u_t in the local cache)
7: **if** $t \mod T = 0$ **then**
8: $c \leftarrow \sum_{i \leftarrow t-T+1}^{t} x_i \mid (x_i \leftarrow 0$, if $u_i = \emptyset$, else $x_i \leftarrow 1)$
9: $\gamma_t \leftarrow \text{Perturb}(c, \epsilon, \sigma)$
10: Signal the owner to run $\prod_{\text{update}}(\gamma_t, \mathcal{DS}_t)$.

Initially, we assume the owner stores $\mathcal{D}_0$ in the local cache σ. DP-Timer first outsources a set of data γ_0 to the server (Alg 1:1-3), where γ_0 is fetched from σ using Perturb (defined in Algorithm 2) operator. Perturb takes as input a count c, a privacy parameter ϵ and a local cache σ to be fetched from. It first perturbs the count c with Laplace noise $\text{Lap}(\frac{1}{\epsilon})$, and then fetches as many records as defined by the noisy count from σ. When there is insufficient data in the local cache, dummy data is added to reach the noisy count. After the initial outsourcing, the owner stores all the received data in the local cache σ (Alg 1:5-7), and DP-Timer will signals for synchronization every T time steps. Whenever a synchronization is posted, the owner counts how many new records have been received since the last update, inputs it to the Perturb operator, and fetches γ_t. The fetched data is then synchronized to the server via the $\prod_{\text{Update}}$ protocol (Alg 1:8-11). The logic behind this algorithm is to provide a synchronization strategy with a fixed time schedule but with noisy record counts at each sync. The DP-Timer method strictly follow the policy of updating once every T moments, but it does not synchronize exactly as much data as it receives between every two syncs. Instead, it may synchronize with additional dummy data, or defer some data for future synchronization.

Algorithm 2 Perturbed Record Fetch

1: **function** Perturb(c, ϵ, σ)
2: $\tilde{c} \leftarrow c + \text{Lap}(\frac{1}{\epsilon})$
3: **if** $\tilde{c} > 0$ **then**
4: **return** read$(\sigma, \tilde{c})$ (read records with noisy size)
5: **else**
6: **return** $\emptyset$ (return nothing if $\tilde{c} \leq 0$)

Theorem 6. *Given privacy budget ϵ, and $k \geq 4 \log \frac{1}{\beta}$ where k denotes the number of times the owner has synchronized so far, $\beta \in (0, 1)$, and $\alpha = \frac{2}{\epsilon}\sqrt{k \log \frac{1}{\beta}}$. This satisfies $\Pr\left[LG(t) \geq \alpha + c^t_{t*}\right] \leq \beta$, where $LG(t)$ is the logical gap at time t under DP-Timer method, and c^t_{t*} counts how many records received since last update.*

Theorem 6 provides an upper bound on the logical gap incurred by DP-Timer, due to space concerns we defer the proof in the complete full version. As a direct corollary of Theorem 6, the logical gap is always bounded by $O(2\sqrt{k}/\epsilon)$. Knowing that, the logical gap can also be used to denote the total records that are on-hold by the owner, thus we can conclude that the local cache size of DP-Timer is also bounded by $O(2\sqrt{k}/\epsilon)$. However, if we consider an

indefinitely growing database, then the local cache size (logical gap) grows indefinitely. Thus to prevent the local cache (logical gap) from getting too large, we employ a cache flush mechanism which refreshes the local cache periodically. The cache flush mechanism flushes a fixed size data with a fixed interval (usually far greater than T). The flushed data will be synchronized to the server immediately. If there is less data than the flush size, the mechanism empties the cache, and synchronizes with additional dummy records. This further guarantees every time when flush is triggered, it always incurs a fixed update volume. Moreover, Theorem 6 also reveals that it is possible to get a bounded local cache size. For example, if we set the flush size $s = C$, and the flush interval $f < T(\epsilon C)^2/4\log(1/\beta)$, where $C > 0, C \in \mathbb{Z}^+$. Then at any time t, with probability at least $1-\beta$, the cache size is bounded by $O(C)$. Next, we discuss the performance overhead with respect to the DP-Timer.

Theorem 7. *Given privacy budget ϵ, flush interval f, flush size s, and $\beta \in (0,1)$. Let $\alpha = \frac{2}{\epsilon}\sqrt{k\log\frac{1}{\beta}}$, and $\eta = s\lfloor t/f \rfloor$. Then for any $t > 4T\log(\frac{1}{\beta})$, the total number of records outsourced under DP-Timer, $|\mathcal{DS}_t|$, satisfies* $\Pr[|\mathcal{DS}_t| \geq |\mathcal{D}_t| + \alpha + \eta] \leq \beta$.

Theorem 7 provides an upper bound for the outsourced data size at each time t. Moreover, it shows that the total amount of dummy data incorporated is bounded by $\eta + O(2\sqrt{k}/\epsilon)$. Due to the existence of the cache flush mechanism, DP-Timer guarantees that for a logical database with length L, all records will be synchronized before time $t = f \times L/s$. Recall that a FIFO based local cache preserves the order of incoming data, thus DP-Timer satisfies the strong eventually consistency property (P3). In addition, as shown by Theorem 6 and 7, both accuracy and performance metrics are related to $\frac{1}{\epsilon}$, which shows that DP-Timer satisfies the P2 principle.

5.2.2 Above noisy threshold (DP-ANT). The Above noisy threshold method, parameterized by θ and ϵ, performs an update when the owner receives approximately θ records. The detailed algorithm is described in Algorithm 3.

Similar to DP-Timer, DP-ANT starts with an initial outsourcing (Alg 3:1-2) and the owner then stores all newly arrived records in the local cache σ (Alg 3:6-9). After the initial outsourcing, DP-ANT splits the privacy budget to two parts ϵ_1, and ϵ_2, where ϵ_1 is used to distort the threshold as well as the counts of records received between two updates, and ϵ_2 is used to fetch data. The owner keeps track of how many new records received since the last update at every time step, distorts it with DP noise, and compares the noisy count to a noisy threshold (Alg 3:10,11). The owner synchronizes if the noisy count exceeds the noisy threshold. After each synchronization, the user resets the noise threshold with fresh DP noise (Alg 3:14) and repeats the aforementioned steps.

DP-ANT synchronizes based on how much data the owner receives. However, it does not simply set a fixed threshold for the owner to synchronize whenever the amount of data received exceeds that threshold. Instead, it utilizes a strategy that allows the owner to synchronize when the amount of received data is approximately equal to the threshold. Below, we analyze DP-ANT's accuracy and performance guarantees.

Theorem 8. *Given privacy budget ϵ and let $\alpha = \frac{16(\log t + \log 2/\beta)}{\epsilon}$. Then for $\beta \in (0,1)$, it satisfies* $\Pr\left[LG(t) \geq \alpha + c^t_{t*}\right] \leq \beta$, *where*

Algorithm 3 Above Noisy Threshold (ANT)

Input: growing database $\mathcal{D} = \{\mathcal{D}_0, U\}$, privacy budget ϵ, threshold θ, and the local cache σ.

1: $\gamma_0 \leftarrow \mathsf{Perturb}(|\mathcal{D}_0|, \epsilon, \sigma)$
2: Signal the owner to run $\prod_{\mathsf{Setup}}(\gamma_0)$.
3: $\epsilon_1 \leftarrow \frac{1}{2}\epsilon, \epsilon_2 \leftarrow \frac{1}{2}\epsilon$
4: $\tilde{\theta} \leftarrow \theta + \mathsf{Lap}(2/\epsilon_1)$, $c \leftarrow 0$, $t^* \leftarrow 0$
5: **for** $t \leftarrow 1, 2, \ldots$ **do**
6: $\quad v_t \leftarrow \mathsf{Lap}(4/\epsilon_1)$
7: $\quad$ **if** $u_t \neq \emptyset$ **then**
8: $\quad\quad$ store u_t in the local cache, $\mathsf{write}(\sigma, u_t)$
9: $\quad c \leftarrow \sum_{i \leftarrow t^*+1}^{t} x_i \mid (x_i \leftarrow 0, \text{if } u_i \leftarrow \emptyset, \text{else } x_i \leftarrow 1)$
10: $\quad$ **if** $c + v_t \geq \tilde{\theta}$ **then**
11: $\quad\quad \gamma_t \leftarrow \mathsf{Perturb}(c, \epsilon_2, \sigma)$
12: $\quad\quad$ Signal the owner to run $\prod_{\mathsf{Update}}(\gamma_t, \mathcal{DS}_t)$
13: $\quad\quad \tilde{\theta} \leftarrow \theta + \mathsf{Lap}(2/\epsilon_1), c \leftarrow 0, t^* \leftarrow t$

$LG(t)$ is the logical gap at time t under DP-ANT method, and c^t_{t*} counts how many records received since last update.

The above theorem provides an upper bound for DP-ANT's logical gap as well as its local cache size, which is $c^t_{t*} + O(16\log t/\epsilon)$. Similar to DP-Timer, we employ a cache flush mechanism to avoid the cache size grows too large. We use the following theorem to describe DP-ANT's performance:

Theorem 9. *Given privacy budget ϵ, flush interval f, flush size s, and $\beta \in (0,1)$. Let $\alpha = \frac{16(\log t + \log 2/\beta)}{\epsilon}$, and $\eta = s\lfloor t/f \rfloor$. Then for any time t, it satisfies* $\Pr[|\mathcal{DS}_t| \geq |\mathcal{D}_t| + \alpha + \eta] \leq \beta$, *where $|\mathcal{DS}_t|$ denotes the total number of records outsourced until time t.*

This theorem shows that the total overhead of DP-ANT at each time t is bounded by $s\lfloor t/f \rfloor + O(16\log t/\epsilon)$. Note that both the upper bound for the logical gap and the performance overhead is related to $1/\epsilon$, which indicates a trade-off between privacy and the accuracy or performance. With different values of ϵ, DP-ANT achieves different level of accuracy and performance (P2 principle). And the FIFO cache as well as the flush mechanism ensures the consistent eventually principle (P3). We provide the related proofs of Theorem 8 and 9 in the full version. Later in Section 8 we further evaluate how different parameters would affect the accuracy and performance of DP strategies, where readers can better understand how to set these parameters according to the desired goals.

6 CONNECTING WITH EXISTING EDBS

Interoperability of DP-Sync with an existing encrypted database is an important requirement (P4 design principle). In this section, we discuss how to connect existing encrypted databases with DP-Sync. Since our privacy model constrains the update leakage of the encrypted database to be a function only related to the update pattern, in this section we mainly focus on query leakage associated with the encrypted database to discuss the compatibility of our framework. Inspired by the leakage levels defined in [20], we categorize different encrypted database schemes based on our own leakage classification. Then we discuss which schemes under those categories can be directly connected with DP-Sync and which databases need additional improvements to be compatible with our

Leakage groups	Encrypted database scheme
L-0	VLH/AVLH [51], ObliDB [34], SEAL [31], Opaque [85], CSAGR19 [27]
L-DP	dp-MM [67], Hermetic [83], KKNO17 [54], Cryptϵ [25], AHKM19 [1], Shrinkwrap [9]
L-1	PPQED$_a$ [72], StealthDB [79], SisoSPIR [47]
L-2	CryptDB [70], Cipherbase [5], Arx [68], HardIDX [35], EnclaveDB [71]

Table 3: Summary of leakage groups and corresponding encrypted database schemes

framework. In Table 3, we summarize some notable examples of encrypted databases with their respective leakage groups. We focus on two types of leakage patterns: *access pattern* [38] and *query response volume* [53]. The access pattern is the transcript of entire memory access sequence for processing a given query, and query response volume refers to the total number encrypted records that matches with a given query. The four leakage categories are as follows:

L-2: Reveal access pattern. Encrypted databases that reveal the exact sequence of memory accesses and response volumes when processing queries fall into this category. These include many practical systems based only on searchable symmetric encryption, trusted execution environments (TEE), or on deterministic and order-preserving encryption. Recent leakage-abuse attacks [11, 20, 63] have pointed out that attackers can exploit the access pattern to reconstruct the entire encrypted database. Databases in this category are not compatible with DP-Sync. If we add our techniques to these systems, then due to the leakage from these databases, our update patterns will be leaked as well.

L-1: Reveal response volume. To hide access patterns, some schemes perform computations obliviously, e.g., using an oblivious RAM. However, many databases in this category still leak the query response volume (since obliviousness does not protect the size of the access pattern). Example databases in this category include HE-based PPQED$_a$ [72] and ORAM-based SisoSPIR [47]. Moreover, recent research [39, 53, 58, 63, 69] has shown that database reconstruction attacks are possible even if the system only leaks response volume. Therefore, there is still a risk that such systems will leak information about the amount of dummy data. Thus, to be compatible with DP-Sync, necessary measures must be taken to hide the query volume information, such as naïve padding [27], pseudorandom transformation [51], etc.

L-DP: Reveal differentially-private response volume. Some secure outsourced database schemes guarantee the leakage of only differentially-private volume information. These schemes either ensure that both access patterns and query volumes are protected using differential privacy, or they completely hide the access patterns and distort the query response volume with differential privacy. Databases with L-DP leakage are directly compatible with DP-Sync, as such schemes prevents attackers from inferring information about dummy data through the query protocol.

L-0: Response volume hiding. Some encrypted databases support oblivious query processing and only leak computationally-secure response volume information. These schemes are usually referred to as access pattern and volume hiding schemes. Encrypted databases in this category can be directly used with our framework as well, as there is no efficient way for attackers to identify dummy data information via their query protocols.

$\mathcal{M}_{\text{timer}}(\mathcal{D}, \epsilon, f, s, T)$

$\mathcal{M}_{\text{setup}}$: **output** $\left(0, |\mathcal{D}_0| + \text{Lap}(\frac{1}{\epsilon})\right)$

$\mathcal{M}_{\text{update}}$: $\forall i \in \mathbb{N}^+$, run $\mathcal{M}_{\text{unit}}(U[i \cdot T, (i+1)T], \epsilon, T)$

$\mathcal{M}_{\text{unit}}$: **output** $\left(i \cdot T,\ \text{Lap}(\frac{1}{\epsilon}) + \sum_{k=i\cdot T+1}^{(i+1)T} 1 \mid u_k \neq \emptyset\right)$

$\mathcal{M}_{\text{flush}}$: $\forall j \in \mathbb{N}^+$, **output** $(j \cdot f,\ s)$

$\mathcal{M}_{\text{ANT}}(\mathcal{D}, \epsilon, f, s, \theta)$

$\mathcal{M}_{\text{setup}}$: **output** $\left(0, |\mathcal{D}_0| + \text{Lap}(\frac{1}{\epsilon})\right)$

$\mathcal{M}_{\text{update}}$: $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$, repeatedly run $\mathcal{M}_{\text{sparse}(\epsilon_1,\epsilon_2,\theta)}$.

$\mathcal{M}_{\text{sparse}}$:

$\tilde{\theta} = \theta + \text{Lap}(\frac{2}{\epsilon_1})$, $t^* \leftarrow$ last time $\mathcal{M}_{\text{sparse}}$'s output $\neq \perp$.

$\forall i \in \mathbb{N}^+$, **output** $\begin{cases} \left(t^* + i, c_i + \text{Lap}(\frac{1}{\epsilon_2})\right) & \text{if } v_i + c_i \geq \tilde{\theta}, \\ \perp & \text{otherwise.} \end{cases}$

where $c_i = \sum_{k=t^*}^{t^*+i} 1 \mid u_k \neq \emptyset$, and $v_i = \text{Lap}(\frac{4}{\epsilon_1})$.

abort the first time when output $\neq \perp$.

$\mathcal{M}_{\text{flush}}$: $\forall j \in \mathbb{N}^+$, **output** $(j \cdot f,\ s)$

Table 4: Mechanisms to simulate the update pattern

In addition, most methods that fall in L-DP and L-0 category support dummy data by default [34, 54, 67, 83], as they use dummy data to populate the query response volume or hide intermediate sizes. In this case, our framework can directly inherit the dummy data types defined in the corresponding database scheme with no additional changes. For those schemes that do not support dummy data by default (e.g. [25]), we can either let the scheme return both dummy and real data, and let the analyst to filter true records after decryption, or we can extend all records with a *isDummy* attribute and then apply query re-writing to eliminate the effect of dummy data.We continue to provide query re-writing examples in our full version. To concretely demonstrate the compatibility of DP-Sync with existing encrypted databases, we choose database schemes ObliDB[34] and Cryptϵ[25] in L-0 and L-DP groups respectively and evaluate the resulting implementation in Section 8.

7 SECURITY PROOFS

In this section, we provide a sketch of the security proof for our proposed DP-Sync implemented with DP strategies.

Theorem 10. *The update pattern of an DP-Sync system implemented with the DP-Timer strategy satisfies Definition 5.*

Proof. (Sketch) To capture the information leakage of the update pattern, we rewrite the DP-Timer algorithm to output the total number of synchronized records at each update, instead of signaling the update protocol. The rewritten mechanism $\mathcal{M}_{\text{timer}}$ (see Table 4) simulates the update pattern when applying the DP-Timer strategy. We prove this theorem by illustrating that the composed privacy guarantee of $\mathcal{M}_{\text{timer}}$ satisfies ϵ-DP.

The mechanism $\mathcal{M}_{\text{timer}}$ is a composition of several separated mechanisms. We now discuss the privacy guarantees of each. $\mathcal{M}_{\text{setup}}$ is a Laplace mechanism, thus its privacy guarantee satisfies ϵ-DP. $\mathcal{M}_{\text{flush}}$ reveals a fixed value with fixed time span in a non data-dependent manner, thus it's output distribution is fully computational indistinguishable (satisfies 0-DP). $\mathcal{M}_{\text{update}}$ is a mechanism that repeatedly calls $\mathcal{M}_{\text{unit}}$. $\mathcal{M}_{\text{unit}}$ acts on a fixed time span (T). It counts the total number of received records within the current time period, and outputs a noisy count with $\text{Lap}(\frac{1}{\epsilon})$ at the end of the

current time period. Thus $\mathcal{M}_{\text{unit}}$ satisfies ϵ-DP guarantee. Since $\mathcal{M}_{\text{update}}$ repeatedly calls $\mathcal{M}_{\text{unit}}$ and applies it over disjoint data, the privacy guarantee of $\mathcal{M}_{\text{unit}}$ follows parallel composition [49], thus satisfying ϵ-DP. The composition of $\mathcal{M}_{\text{setup}}$ and $\mathcal{M}_{\text{update}}$ also follows parallel composition and the composition of $\mathcal{M}_{\text{flush}}$ follows simple composition [49]. Thus the entire algorithm $\mathcal{M}_{\text{timer}}$ satisfies $(\max(\epsilon, \epsilon) + 0)$ -DP, which is ϵ-DP.

□

Theorem 11. *The update pattern of an DP-Sync system implemented with the ANT strategy satisfies Definition 5.*

Proof. (Sketch) Similar to the proof of Theorem 7, we first provide $\mathcal{M}_{ANT}$ (Table 4) that simulates the update pattern of ANT strategy. We prove this theorem by illustrating the composed privacy guarantee of $\mathcal{M}_{\text{ANT}}$ satisfies ϵ-DP.

The mechanism $\mathcal{M}_{\text{ANT}}$ is a composition of several separated mechanisms. $\mathcal{M}_{\text{setup}}$ and $\mathcal{M}_{\text{flush}}$ satisfy ϵ-DP and 0-DP, respectively. We abstract the $\mathcal{M}_{\text{update}}$ as a composite mechanism that repeatedly spawns $\mathcal{M}_{\text{sparse}}$ on disjoint data. Hence, in what follows we show that $\mathcal{M}_{\text{sparse}}$, and thus also $\mathcal{M}_{\text{update}}$ (repeatedly call $\mathcal{M}_{\text{sparse}}$), satisfies ϵ-DP guarantee.

Assume a modified version of $\mathcal{M}_{\text{sparse}}$, say $\mathcal{M}'_{\text{sparse}}$, where it outputs $\top$ once the condition $v_i + c_i > \tilde{\theta}$ is satisfied, and outputs $\perp$ for all other cases. Then the output of $\mathcal{M}'_{\text{sparse}}$ can be written as $O = \{o_1, o_2, ..., o_m\}$, where $\forall\ 1 \leq i < m$, $o_i = \perp$, and $o_m = \top$. Suppose that U and U' are the logical updates of two neighboring growing databases and we know that for all i, $\Pr[\tilde{c}_i < x] \leq \Pr\left[\tilde{c}'_i < x + 1\right]$ is satisfied, where $\tilde{c}_i$ and $\tilde{c}'_i$ denotes the i^{th} noisy count when applying $\mathcal{M}'_{\text{sparse}}$ over U and U' respectively, such that:

$$
\begin{aligned}
&\Pr\left[\ \mathcal{M}'_{\text{sparse}}(U) = O\right] \\
&= \int_{-\infty}^{\infty} \Pr\left[\tilde{\theta} = x\right] \left(\prod_{1 \leq i < m} \Pr\left[\tilde{c}_i < x\right]\right) \Pr\left[\tilde{c}_m \geq x\right] dx \\
&\leq \int_{-\infty}^{\infty} e^{\epsilon/2} \Pr\left[\tilde{\theta} = x + 1\right] \left(\prod_{1 \leq i < m} \Pr\left[\tilde{c}'_i < x + 1\right]\right) \Pr\left[v_m \geq x - c_m\right] dx \\
&\leq \int_{-\infty}^{\infty} e^{\epsilon/2} \Pr\left[\tilde{\theta} = x + 1\right] \left(\prod_{1 \leq i < m} \Pr\left[\tilde{c}'_i < x + 1\right]\right) \\
&\qquad \times e^{\epsilon/2} \Pr\left[v_m + c'_m \geq x + 1\right] dx \\
&= \int_{-\infty}^{\infty} e^{\epsilon} \Pr\left[\tilde{\theta} = x + 1\right] \left(\prod_{1 \leq i < m} \Pr\left[\tilde{c}'_i < x + 1\right]\right) \Pr\left[\tilde{c}'_m \geq x + 1\right] dx \\
&= e^{\epsilon} \Pr[\mathcal{M}'_{\text{sparse}}(U') = O]
\end{aligned}
\tag{1}
$$

Thus $\mathcal{M}'_{\text{sparse}}$ satisfies ϵ-DP, and $\mathcal{M}_{\text{sparse}}$ is essentially a composition of a $\mathcal{M}'_{\text{sparse}}$ satisfying $\frac{1}{2}\epsilon$-DP together with a Laplace mechanism with privacy parameter equal to $\frac{1}{2}\epsilon$. Hence by applying simple composition [49], we see that $\mathcal{M}_{\text{sparse}}$ satisfies $(\frac{1}{2}\epsilon + \frac{1}{2}\epsilon) - DP$. Knowing that $\mathcal{M}_{\text{update}}$ runs $\mathcal{M}_{\text{sparse}}$ repeatedly on disjoint data, with parallel composition [49], the $\mathcal{M}_{\text{update}}$ then satisfies ϵ-DP. Finally, combined with $\mathcal{M}_{\text{setup}}$ and $\mathcal{M}_{\text{flush}}$, we conclude that $\mathcal{M}_{\text{ANT}}$ satisfies ϵ-DP, thus the theorem holds. □

8 EXPERIMENTAL ANALYSIS

In this section, we describe our evaluation of DP-Sync along two dimensions: accuracy and performance. Specifically, we address the following questions in our experimental studies:

- **Question-1**: How do DP strategies compare to naïve methods in terms of performance and accuracy under a fixed level of privacy? Do DP strategies guarantee bounded accuracy?
- **Question-2**: What is the impact on accuracy and performance when changing the privacy level of the DP strategies? Can we adjust privacy to obtain different levels of accuracy or performance guarantees?
- **Question-3**: With a fixed level of privacy, how does accuracy and performance change if we change the non-privacy parameters T or θ for DP-Timer and DP-ANT, respectively?

Implementation and configuration. To answer the above questions, we implement multiple instances of DP-Sync, execute them with real-world datasets as inputs, and run queries on the deployed system to evaluate different metrics. We implement the DP-Sync using two encrypted database schemes, ObliDB [34], and Cryptϵ [25], from L-0 group and L-DP group, respectively. All experiments are performed on IBM Bare metal servers with 3.8GHz Intel Xeon E-2174G CPU, 32Gb RAM and 64 bit Ubuntu 18.04.1. The ObliDB system is compiled with Intel SGX SDK version 2.9.1. We implement the client using Python 3.7, which takes as input a timestamped dataset, but consumes only one record per round. The client simulates how a real-world client device would receive new records over time. In our experiment, we assume the time span between two consecutive time stamps is 1 minute.

Data. We evaluate the two systems using *June 2020 New York City Yellow Cab taxi trip record* and *June 2020 New York City Green Boro taxi trip record*. Both data sets can be obtained from the TLC Trip Record Project [78]. We multiplex the pickup time information of each data point as an indication of when the data owner received this record. We process the raw data with the following steps: (1) Delete invalid data points with incomplete or missing values; (2) Eliminate duplicated records that occur in the same minute, keeping only one.[1] The processed data contains 18,429 and 21,300 records for Yellow Cab and Green Taxi, respectively. (3) Since the monthly data for June 2020 should have 43,200 time units in total, for those time units without associated records, we input a null type record to simulate absence of received data.

Testing query. We select three queries in our evaluation: a linear range query, an aggregation query and a join query.

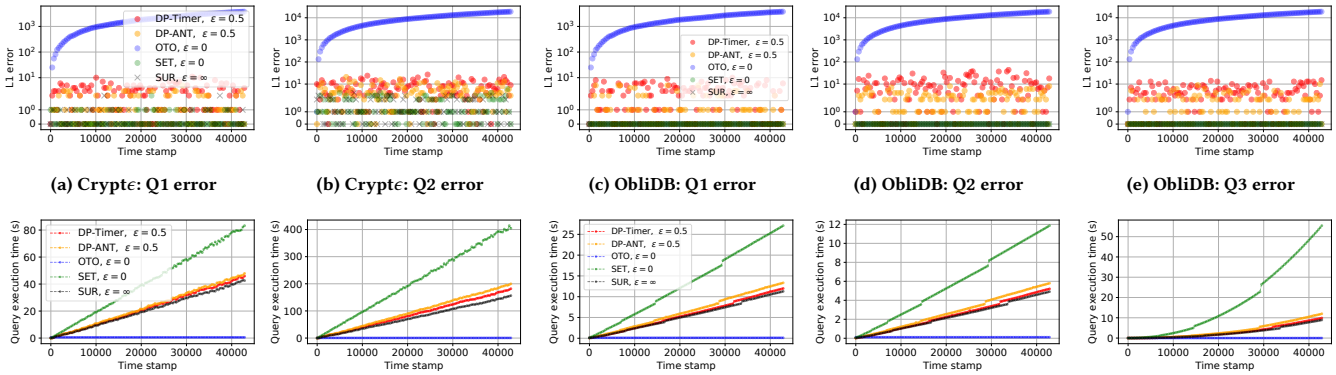
Q1-Linear range query that counts the total number of records in Yellow Cab data with pickupID within 50-100: "`SELECT COUNT(*) FROM YellowCab WHERE pickupID BETWEEN 50 AND 100`"

Q2-Aggregation query for Yellow Cab data that counts the number of pickups grouped by location:"`SELECT pickupID, COUNT(*) AS PickupCnt FROM YellowCab GROUP BY pickupID`"

Q3-Join query that counts how many times both providers have assigned trips: "`SELECT COUNT(*) FROM YellowCab INNER JOIN GreenTaxi ON YellowCab.pickTime = GreenTaxi.pickTime`".

Default setting. Unless specified otherwise, we assume the following defaults. For both DP methods, we set the default privacy

[1] At most one record occurs at each timestamp.



(a) Cryptϵ: Q1 error (b) Cryptϵ: Q2 error (c) ObliDB: Q1 error (d) ObliDB: Q2 error (e) ObliDB: Q3 error

(f) Cryptϵ: Q1 time (g) Cryptϵ: Q2 time (h) ObliDB: Q1 time (i) ObliDB: Q2 time (j) ObliDB: Q3 time

Figure 2: End-to-end comparison for synchronization strategies.

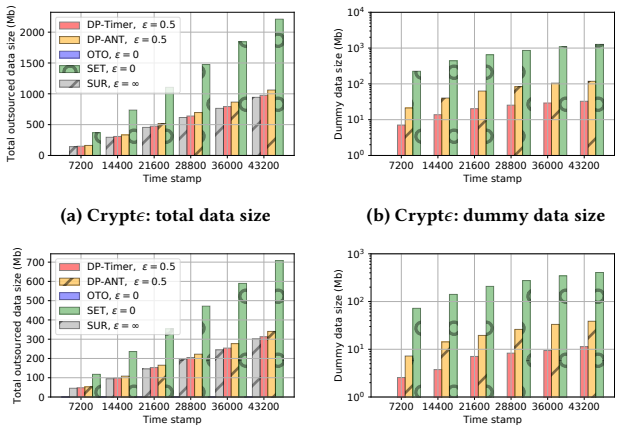
as $\epsilon = 0.5$, and cache flush parameters as $f = 2000$ (flush interval) and $s = 15$ (flush size). For DP-Timer, the default T is 30 and for DP-ANT the default $\theta = 15$. We set the ObliDB implementation as the default system and Q2 as the default testing query.

8.1 End-to-end Comparison

In this section, we evaluate **Question-1** by conducting a comparative analysis between the aforementioned DP strategies' empirical accuracy and performance metrics and that of the naïve methods. We run DP-Sync under 5 synchronization strategies and for each group we send testing queries[2] every 360 time units (corresponding to 6 hours). In each group, we report the corresponding L1 error and query execution time (QET) for each testing query as well as the outsourced and dummy data size over time. In addition, we set the privacy budget (used to distort the query answer) of Cryptϵ as 3, and we use the default setting for ObliDB with ORAM enabled.

Observation 1. The query errors for both DP strategies are bounded, and such errors are much smaller than that of OTO. Figure 2 shows the L1 error and QET for each testing query, the aggregated statistics, such as the mean L1 error and mean QET for all testing queries is reported in Table 5. First we can observe from Figure 2a and 2c that the L1 query error of Q1 for both DP strategies fluctuate in the range 0-15. There is no accumulation of query errors as time goes by. Similarly, Figure 2b, 2d, and 2e show that the errors for both Q2 and Q3 queries are limited to 0-50 under the DP strategies. Note that the query errors in the Cryptϵ group are caused by both the unsynchronized records at each time as well as the DP noise injected when releasing the query answer, but the query errors under ObliDB group are caused entirely by unsynchronized records at each time step. This is why, under the Cryptϵ group, the SET and SUR methods have non-zero L1 query errors even if these two methods guarantee no unsynchronized data at any time. For the OTO approach, since the user is completely offline after the initial phase, the outsourced database under OTO misses all records after $t = 0$, resulting in unbounded query errors. According to Table 5, the average L1 errors under OTO method are 1929.47, 9214.47, and 3702.6, respectively for Q1, Q2, and Q3, which are at least 520x of that of the DP strategies.

[2]Cryptϵ does not support join operators, thus we only test Q1 and Q2 for Cryptϵ



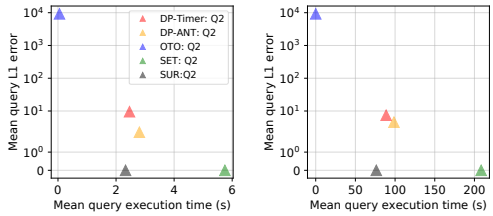
(a) Cryptϵ: total data size (b) Cryptϵ: dummy data size

(c) ObliDB: total data size (d) ObliDB: dummy data size

Figure 3: Total and dummy data size.

Observation 2. The DP methods introduce a small performance overhead compared to SUR, and achieve performance gains up to 5.72x compared to the SET method. We show the total and dummy data size under each method in Figure 3. According to Figure 3a and 3c, we find that at all time steps, the outsourced data size under both DP approaches are quite similar to that of SUR approach, with at most 6% additional overhead. However, the SET method outsources at least twice as much data as the DP methods under all cases. In total (Table 5), SET outsources at least 2.24x and 2.10x more data than DP-Timer and DP-ANT, respectively. OTO always have fixed storage size (0.056 and 0.016 Mb for Cryptϵ and ObliDB group) as it only outsources once. Note that the amount of outsourced data under the SUR schema at any time is identical to the amount of data in the logical database. Thus, any oversize of outsourcing data in contrast to SUR is due to the inclusion of dummy data. According to Figure 3b, 3d, and Table 5, SET introduces at least 11.5x, and can achieve up to 35.6x, more dummy records than DP approaches. Adding dummy data not only inflates the storage, but also results in degraded query response performance. As DP approaches much fewer dummy records, they exhibit little degradation in query performance compared to the SUR method. The SET method, however, uploads many dummy records, thus its query performance drops sharply. According to Figure 2f, 2h, 2g, 2i, 2j, at

almost all time steps, the server takes twice as much time to run Q1 and Q2 under the SET method than under DP strategies and take at least 4x more time to run Q3. Based on Table 5, the average QET for Q1 and Q2 under SET are at least 2.17x and 2.3x of that under the DP methods. It's important to point out that both Q1 and Q2 have complexity in $O(N)$, where N is the number of outsourced data. Thus for queries with complexity of $O(N^2)$, such as Q3, the performance gap between the DP strategies and the SET is magnified, in this case boosted to 5.72x. Furthermore, the number of records that SET outsources at any time t is fixed, $|\mathcal{D}_0| + t$. Thus, if the growing database $\mathcal{D} = \{\mathcal{D}_0, U\}$ is sparse (most of the logical updates $u_i \in U$ are $\emptyset$), the performance gap in terms of QET between SET and DP strategies will be further amplified. The the ratio of $(|\mathcal{D}_0| + t)/|\mathcal{D}_t|$ is relatively large if $\mathcal{D}$ is sparse.



(a) ObliDB group (b) Cryptϵ group

Figure 4: QET v.s. L1 error

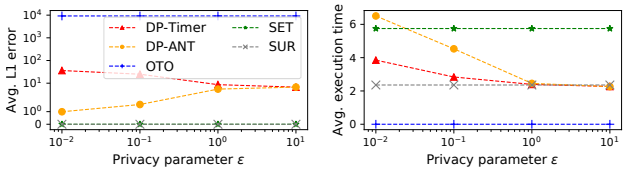
Observation 3. DP strategies are optimized for the dual objectives of accuracy and performance. To better understand the advantage of DP strategies, we compare the default query (Q2) results with respect to DP strategies and naïve methods in Figure 4, where the x-axis is the performance metric (mean query QET for all queries posted over time), and the y-axis is the accuracy metric (mean query L1 error). Though it seems that SUR is ideal (least query error and no performance overhead), it has no privacy guarantee. Both SET and OTO provide complete privacy. We observe that, the data points of SET fall in the lower right corner of each figure, indicating that the SET method completely sacrifices performance in exchange for a better accuracy guarantee. Thus SET is a private synchronization method that is optimized solely for accuracy. Another extreme case is the OTO method, where the corresponding data points fall in the upper left corner. This means that OTO completely sacrifices accuracy for performance, thus it is optimized for performance only. DP strategies provide privacy guarantees bounded by ϵ, and we observe that the corresponding data points fall in the lower left corner of the figure (close to SUR), indicating that the DP strategies provide considerable accuracy guarantees (or bounded error) at a small sacrifice in performance. This is further evidence that DP strategies are optimized for the dual objectives of accuracy and performance.

8.2 Trade-off with Changing Privacy Level

We address **Question-2** by evaluating the DP policies with different ϵ ranging from 0.001 to 10. For other parameters associated with DP strategies, we apply the default setting and evaluate them with the default testing query Q2 on the default system (ObliDB based implementation). For each ϵ, we report the mean query error and QET. We summarize our observations as follows.

Comparison Categories		SUR	SET	OTO	DP Timer	DP ANT
Cryptϵ						
Q1	**Mean L1 Err**	0.75	0.71	1929.47	3.04	0.99
	Max L1 Err	2	3	8079	11	5
	Mean QET	20.94	41.70	0.33	22.51	23.54
Q2	**Mean L1 Err**	3.36	3.39	9464.14	7.45	4.56
	Max L1 Err	13	15	18446	27	21
	Mean QET	76.34	208.47	0.72	88.75	98.58
Mean logical gap		0	0	9214.5	10.91	4.8
Total data (Mb)		943.5	2211.79	0.052	979.10	1027.31
Dummy data (Mb)		N/A	1268.29	N/A	35.6	83.8
ObliDB						
Q1	**Mean L1 Err**	0	0	1929.47	2.95	0.91
	Max L1 Err	0	0	8801	10	5
	Mean QET	5.39	14.18	0.041	5.69	6.48
Q2	**Mean L1 Err**	0	0	9214.51	9.25	2.25
	Max L1 Err	0	0	18429	44	8
	Mean QET	2.32	5.76	0.071	2.46	2.80
Q3	**Mean L1 Err**	0	0	3702.6	4.93	1.43
	Max L1 Err	0	0	7407	15	10
	Mean QET	2.77	17.86	0.095	3.12	3.86
Mean Logical gap		0	0	9214.5	10.73	2.96
Total data (Mb)		301.90	707.79	0.016	316.68	337.09
Dummy data (Mb)		N/A	405.89	N/A	14.78	35.19

Table 5: Aggregated statistics for comparison experiment



(a) Avg. L1 error v.s. Privacy (b) Avg. execution time v.s. Privacy

Figure 5: Trade-off with changing privacy.

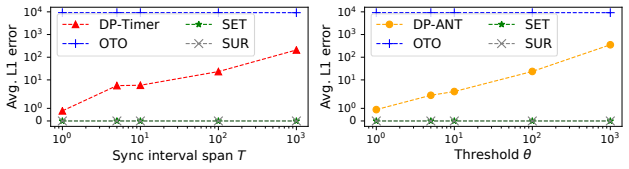
Observation 4. DP-Timer and DP-ANT exhibit different trends in accuracy when ϵ changes. Figure 5a illustrates the evaluation results for privacy versus accuracy. In general, we observe that, as ϵ increases from 0.01 to 1, the mean query error of DP-ANT increases while the error of DP-Timer decreases. Both errors change slightly from $\epsilon = 1$ to $\epsilon = 10$. Recall that DP-Timer's logic gap consists of the number of records received since the last update,c^t_{t*}, and the data delayed by the previous synchronization operation (bounded by $O(2\sqrt{k}/\epsilon)$). Since the update frequency of the DP-Timer is fixed, c^t_{t*} is not affected when ϵ changes. However, when the ϵ is smaller, the number of delayed records increases, which further leads to higher query errors. For the DP-ANT though, when the ϵ is very small, the delayed records increases as well (bouned by $O(16 \log t/\epsilon)$). However, smaller ϵ (large noise) will result in more frequent updates for the DP-ANT. This is because the large noise will cause the DP-ANT to trigger the upload condition early before it receives enough data. As a result, the number of records received since last update, c^t_{t*}, will be reduced, which essentially produces smaller query errors. In summary, for DP strategies, we observe that there is a trade-off between privacy and accuracy guarantee.

Observation 5. Both DP strategies show decreasing performance overhead when ϵ increases. Both DP methods show similar tendencies in terms of the performance metrics (Figure 5b). When ϵ increases, the QET decreases. This can be explained by Theorem 7 and 9. That is, with a relatively large ϵ, the dummy

records injected at each update will be reduced substantially. As a result, less overhead will be introduced and the query response performance is then increased. Similarly, for DP strategies, there is a trade-off between privacy and performance.

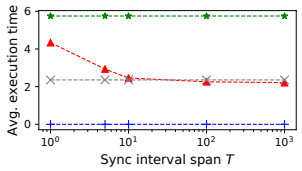
8.3 Trade-off with Fixed Privacy Level

We address **Question-3** by evaluating the DP policies with default ϵ but changing T and θ for DP-Timer and DP-ANT, respectively.

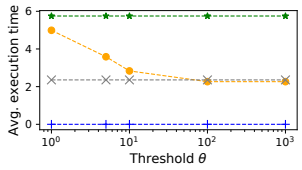


(a) Avg. L1 error v.s. T

(b) Avg. L1 error v.s. θ



(c) Avg. execution time v.s. T



(d) Avg. execution time v.s. θ

Figure 6: Trade-off with non-privacy parameters

Observation 6. Even with fixed privacy, the DP strategies can still be tuned to obtain different performance or accuracy by adjusting non-privacy parameters. From Figure 6a and 6b, we observe that the mean query errors for both methods increase when T or θ increases. This is because once T or θ is increased, the owner waits longer before making an update, which increases the logical gap. Also Figure 6c and 6d shows that the performance metric decreases as T or θ increases. This is because as T or θ increases, the owner updates less frequently, which reduces the number of dummy records that could be injected into the outsourced database.

9 RELATED WORK

Encrypted databases and their leakage. Encrypted databases is a broadly studied research topic. Existing solutions utilize techniques such as bucketization [42, 45, 46], predicate encryption [62, 74], oblivious RAM [8, 26, 31, 47, 65], structural encryption and symmetric searchable encryption (SSE) [4, 21, 29, 37, 50–52, 67, 76], functional encryption [15, 73], property-preserving encryption [2, 10, 12, 66], order-preserving encryption [2, 13], trusted execution environments [34, 71, 79] and homomorphic encryption [16, 25, 36, 72]. Recent work has revealed that these methods may be subject to information leakage through query patterns [11, 20], identifier patterns [11], access patterns [20, 30, 53] and query response volume [11, 39–41, 53]. In contrast, our work analyzes information leakage for encrypted databases through update patterns. Recent work on backward private SSE [4, 17, 37, 77], which proposes search (query) protocols that guarantee limits on information revealed through data update history, shares some similarity with our work. However, this approach is distinct from our work as they hide the update history from the query protocol. Moreover, backward private SSE permits insertion pattern leakage, revealing how many and when records have been inserted. In contrast, our work hides insertion pattern leakage through DP guarantees. Similar to our work, Obladi [26] supports updates on top of outsourced encrypted databases. However, it focuses on ACID properties for OLTP workloads and provides no accuracy guarantees for the analytics queries.

Differentially-private leakage. The concept of DP leakage for encrypted databases was first introduced by Kellaris et al. [54]. Interesting work has been done on DP access patterns [9, 23, 64, 80], DP query volume [67] and DP query answering on encrypted data [1, 25, 60]. However, most of this work focuses on the static database setting. Agarwal et al. [1] consider the problem of answering differentially-private queries over encrypted databases with updates. However, their work focuses mainly on safeguarding the query results from revealing sensitive information, rather than protecting the update leakage. Lécuyer et al. [60] investigate the method to privately update an ML model with growing training data. Their work ensures the adversary can not obtain useful information against the newly added training data by continually observing the model outputs. However, they do not consider how to prevent update pattern leakage. Kellaris et al. [54] mention distorting update record size by adding dummy records, but their approach always overcounts the number of records in each update, which incorporates large number of dummy records. Moreover, their main contribution is to protect the access pattern of encrypted databases rather than hiding the update patterns. In addition, none of these approaches formally defined the update pattern as well as it's corresponding privacy, and none of them have considered designing private synchronization strategies.

Differential privacy under continual observation. The problem of differential privacy under continual observation was first introduced by Dwork et al. in [33], and has been studied in many recent works [14, 24, 28, 32, 84]. These approaches focus on designing DP streaming algorithms and are not specific to outsourced databases. In particular, although [28] analyzes privacy for growing databases, unlike our work, their model assumes that the server has full access to all outsourced data.

10 CONCLUSION

In this paper, we have introduced a new type of leakage associated with modern encrypted databases called update pattern leakage. We formalize the definition and security model of SOGDB with DP update patterns. We also proposed the framework DP-Sync, which extends existing encrypted database schemes to SOGDB with DP update patterns. DP-Sync guarantees that the entire data update history over the outsourced data structure is protected by differential privacy. This is achieved by imposing differentially-private strategies that dictate the owner's synchronization of local data.

Note that DP-Sync currently only supports single table schema. Supporting multi-relational table databases require additional security models, data truncation techniques [57] and secure protocols to compute the sensitivity [48] over multiple tables. We leave the design of these techniques for future work.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants 2016393, 2029853; and by DARPA and SPAWAR under contract N66001-15-C-4067.

REFERENCES

[1] Archita Agarwal, Maurice Herlihy, Seny Kamara, and Tarik Moataz. 2019. Encrypted Databases for Differential Privacy. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 170–190.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 563–574.

[3] Joshua Allen, Bolin Ding, Janardhan Kulkarni, Harsha Nori, Olga Ohrimenko, and Sergey Yekhanin. 2019. An algorithmic framework for differentially private data analysis on trusted processors. In *Advances in Neural Information Processing Systems*. 13657–13668.

[4] Ghous Amjad, Seny Kamara, and Tarik Moataz. 2019. Forward and backward private searchable encryption with SGX. In *Proceedings of the 12th European Workshop on Systems Security*. 1–6.

[5] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase.. In *CIDR*.

[6] Arvind Arasu and Raghav Kaushik. 2013. Oblivious query processing. *arXiv preprint arXiv:1312.4012* (2013).

[7] Gilad Asharov, T-H Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Locality-preserving oblivious ram. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 214–243.

[8] Johes Bater, Satyender Goel, Gregory Elliott, Abel Kho, Craig Eggen, and Jennie Rogers. 2016. SMCQL: Secure querying for federated databases. *Proceedings of the VLDB Endowment* 10, 6 (2016), 673–684.

[9] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient SQL query processing in differentially private data federations. *Proceedings of the VLDB Endowment* 12, 3 (2018), 307–320.

[10] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. 2007. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*. Springer, 535–552.

[11] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2019. Revisiting Leakage Abuse Attacks. *IACR Cryptol. ePrint Arch.* 2019 (2019), 1175.

[12] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. 2009. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 224–241.

[13] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*. Springer, 578–595.

[14] Jean Bolot, Nadia Fawaz, Shanmugavelayutham Muthukrishnan, Aleksandar Nikolov, and Nina Taft. 2013. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*. 284–295.

[15] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public key encryption with keyword search. In *International conference on the theory and applications of cryptographic techniques*. Springer, 506–522.

[16] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *Theory of cryptography conference*. Springer, 325–341.

[17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1465–1482.

[18] Sebastian Burckhardt. 2014. Principles of eventual consistency. (2014).

[19] Yang Cao, Masatoshi Yoshikawa, Yonghui Xiao, and Li Xiong. 2017. Quantifying differential privacy under temporal correlations. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 821–832.

[20] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679.

[21] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic searchable encryption in very-large databases: data structures and implementation.. In *NDSS*, Vol. 14. Citeseer, 23–26.

[22] Melissa Chase and Seny Kamara. 2010. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 577–594.

[23] Guoxing Chen, Ten-Hwang Lai, Michael K Reiter, and Yinqian Zhang. 2018. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 810–818.

[24] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. Pegasus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1375–1388.

[25] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2019. Cryptc: Crypto-assisted differential privacy on untrusted servers. SIGMOD.

[26] Natacha Crooks, Matthew Burke, Ethan Cecchetti, Sitar Harel, Rachit Agarwal, and Lorenzo Alvisi. 2018. Obladi: Oblivious Serializable Transactions in the Cloud. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 727–743. https://www.usenix.org/conference/osdi18/presentation/crooks

[27] Shujie Cui, Xiangfu Song, Muhammad Rizwan Asghar, Steven D Galbraith, and Giovanni Russello. 2019. Privacy-preserving searchable databases with controllable leakage. *arXiv preprint arXiv:1909.11624* (2019).

[28] Rachel Cummings, Sara Krehbiel, Kevin A Lai, and Uthaipon Tantipongpipat. 2018. Differential privacy for growing databases. In *Advances in Neural Information Processing Systems*. 8864–8873.

[29] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.

[30] Jonathan L Dautrich Jr and Chinya V Ravishankar. 2013. Compromising privacy in precise query protocols. In *Proceedings of the 16th International Conference on Extending Database Technology*. 155–166.

[31] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. {SEAL}: Attack Mitigation for Encrypted Databases via Adjustable Leakage. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.

[32] Cynthia Dwork. 2010. Differential privacy in new settings. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 174–183.

[33] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 715–724.

[34] Saba Eskandarian and Matei Zaharia. 2017. Oblidb: Oblivious query processing using hardware enclaves. *arXiv preprint arXiv:1710.00458* (2017).

[35] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. 2017. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 386–408.

[36] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[37] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1038–1055.

[38] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[39] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 315–331.

[40] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2019. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1067–1083.

[41] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted databases: New volume attacks against range queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 361–378.

[42] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 216–227.

[43] Florian Hahn and Florian Kerschbaum. 2014. Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 310–320.

[44] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. 2017. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1389–1406.

[45] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure multidimensional range queries over outsourced data. *The VLDB Journal* 21, 3 (2012), 333–358.

[46] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. 2004. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 720–731.

[47] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2016. Private large-scale databases with distributed searchable symmetric encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 90–107.

[48] 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.

[49] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2015. The composition theorem for differential privacy. In *International conference on machine learning*. 1376–1385.

[50] Seny Kamara and Tarik Moataz. 2018. SQL on structurally-encrypted databases. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 149–180.

[51] Seny Kamara and Tarik Moataz. 2019. Computationally volume-hiding structured encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 183–213.

[52] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 965–976.

[53] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. 2016. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1329–1340.

[54] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2017. Accessing data while preserving privacy. *arXiv preprint arXiv:1706.01552* (2017).

[55] Daniel Kifer and Ashwin Machanavajjhala. 2011. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 193–204.

[56] Daniel Kifer and Ashwin Machanavajjhala. 2014. Pufferfish: A framework for mathematical privacy definitions. *ACM Transactions on Database Systems (TODS)* 39, 1 (2014), 1–36.

[57] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.

[58] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 297–314.

[59] Kim Laine. 2017. Simple encrypted arithmetic library 2.3. 1. *Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf* (2017).

[60] Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu. 2019. Privacy Accounting and Quality Control in the Sage Differentially Private ML Platform. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 181–195. https://doi.org/10.1145/3341301.3359639

[61] Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. 2016. Dependence Makes You Vulnberable: Differential Privacy Under Dependent Tuples.. In *NDSS*, Vol. 16. 21–24.

[62] Yanbin Lu. 2012. Privacy-preserving Logarithmic-time Search on Encrypted Data in Cloud.. In *NDSS*.

[63] Evangelia Anna Markatou and Roberto Tamassia. 2019. Full database reconstruction with access and search pattern leakage. In *International Conference on Information Security*. Springer, 25–43.

[64] Sahar Mazloom and S Dov Gordon. 2018. Secure computation with differentially private access patterns. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 490–507.

[65] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. 2014. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 639–654.

[66] Omkant Pandey and Yannis Rouselakis. 2012. Property preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 375–391.

[67] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 79–93.

[68] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2016. Arx: A Strongly Encrypted Database System. *IACR Cryptol. ePrint Arch.* 2016 (2016), 591.

[69] Rishabh Poddar, Stephanie Wang, Jianan Lu, and Raluca Ada Popa. 2020. Practical Volume-Based Attacks on Encrypted Databases. *arXiv preprint arXiv:2008.06627* (2020).

[70] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2012. CryptDB: processing queries on an encrypted database. *Commun. ACM* 55, 9 (2012), 103–111.

[71] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. Enclavedb: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[72] Bharath Kumar Samanthula, Wei Jiang, and Elisa Bertino. 2014. Privacy-preserving complex query evaluation over semantically secure encrypted data. In *European Symposium on Research in Computer Security*. Springer, 400–418.

[73] Emily Shen, Elaine Shi, and Brent Waters. 2009. Predicate privacy in encryption systems. In *Theory of Cryptography Conference*. Springer, 457–473.

[74] Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig. 2007. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 350–364.

[75] Shuang Song, Yizhen Wang, and Kamalika Chaudhuri. 2017. Pufferfish privacy mechanisms for correlated data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1291–1306.

[76] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical Dynamic Searchable Encryption with Small Leakage.. In *NDSS*, Vol. 71. 72–75.

[77] Shi-Feng Sun, Xingliang Yuan, Joseph K Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. 2018. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 763–780.

[78] NYC Taxi, Limousine Commission, et al. 2020. *NYC Yellow Taxi Trip Records*. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[79] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. Stealthdb: a scalable encrypted database with full SQL query support. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 370–388.

[80] Sameer Wagh, Paul Cuff, and Prateek Mittal. 2018. Differentially private oblivious ram. *Proceedings on Privacy Enhancing Technologies* 2018, 4 (2018), 64–84.

[81] Songrui Wu, Qi Li, Guoliang Li, Dong Yuan, Xingliang Yuan, and Cong Wang. 2019. Servedb: Secure, verifiable, and efficient range queries on outsourced database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 626–637.

[82] Yonghui Xiao and Li Xiong. 2015. Protecting locations with differential privacy under temporal correlations. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1298–1309.

[83] Min Xu, Antonis Papadimitriou, Andreas Haeberlen, and Ariel Feldman. [n.d.]. Hermetic: Privacy-preserving distributed analytics without (most) side channels. *External Links: Link Cited by* ([n. d.]).

[84] Xiaokuan Zhang, Jihun Hamm, Michael K Reiter, and Yinqian Zhang. 2019. Statistical Privacy for Streaming Traffic.. In *NDSS*.

[85] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 283–298.