sciendo

Proceedings on Privacy Enhancing Technologies ; 2019 (3):170–190

Archita Agarwal, Maurice Herlihy, Seny Kamara, and Tarik Moataz

Encrypted Databases for Differential Privacy

Abstract: The problem of privatizing statistical databases is a well-studied topic that has culminated with the notion of differential privacy. The complementary problem of *securing* these differentially private databases, however, has—as far as we know—not been considered in the past. While the security of private databases is in theory orthogonal to the problem of private statistical analysis (e.g., in the central model of differential privacy the curator is trusted) the recent real-world deployments of differentially-private systems suggest that it will become a problem of increasing importance. In this work, we consider the problem of designing encrypted databases (EDB) that support differentially-private statistical queries. More precisely, these EDBs should support a set of encrypted operations with which a curator can securely query and manage its data, and a set of private operations with which an analyst can privately analyze the data. Using such an EDB, a curator can securely outsource its database to an untrusted server (e.g., on-premise or in the cloud) while still allowing an analyst to privately query it. We show how to design an EDB that supports private histogram queries. As a building block, we introduce a differentially-private encrypted counter based on the binary mechanism of Chan et al. (*ICALP*, 2010). We then carefully combine multiple instances of this counter with a standard encrypted database scheme to support differentially-private histogram queries.

Keywords: Structured Encryption, Differential Privacy

DOI 10.2478/popets-2019-0042
Received 2018-11-30; revised 2019-03-15; accepted 2019-03-16.

1 Introduction

A statistical database system is a database management system designed to support statistical analysis on the data it stores. Statistical database systems are ubiquitous and support decision making in almost every domain, including, technology, finance, education, Government, sports and national security; just to name a few. In fact, the proliferation and significance of statistical databases has motivated the important and active field of *private statistical analysis* which includes work on privacy attacks [23, 53, 62] and on the design of statistical mechanisms that achieve various notions of privacy like k-anonymity, ℓ-diversity and, ultimately, differential privacy as introduced by Dwork, McSherry, Nissim and Smith [25].

In this setting, a trusted curator stores the database and responds to statistical queries made by an untrusted analyst. By answering these queries with a differentially-private mechanism, the curator can guarantee that the analyst gets responses within some bound of the correct answer in such a way that the presence or absence of an individual person/entity does not affect the output of the mechanism by much. Over the last fifteen years, research in differential privacy has produced a multitude of mechanisms to support a wide array of statistical analyses. In turn, these advances have lead to real-world deployments of differentially-private statistical databases in various domains. Some of the most high-profile examples include the Census Bureau's deployment of differential privacy in their *OnTheMap* project which supports analysis on the travel patterns of commuters [49], Apple's deployment of local differential privacy to study emoji usage, health data and media preferences [22], and Google's deployment of local differential privacy to analyze Chrome settings [30].

While it is clear that the problem of "privatizing" statistical databases has received significant attention, as far as we know, the complementary problem of *securing* differentially private statistical databases has not been considered. While the security of statistical databases is in theory orthogonal to the problem of private statistical analysis—indeed in this setting the curator is trusted—we believe that the growing number of real-world deployments of differential privacy will increasingly highlight the problem of how exactly curators should protect their data. This question is even more pertinent given the highly sensitive nature of the data—which is why differentially-private mechanisms are used in the first place—and the constant occurrences of data breaches.

Archita Agarwal: Brown University, E-mail: archita_agarwal@brown.edu
Maurice Herlihy: Brown University, E-mail: maurice_herlihy@brown.edu
Seny Kamara: Brown University, E-mail: seny_kamara@brown.edu
Tarik Moataz: Brown University, E-mail: tarik_moataz@brown.edu

Security via pan privacy. One approach to addressing this security problem is to use a *pan-private* mechanism [26, 27] which, roughly speaking, generates a representation of the data—called the state—in such a way that differential privacy is preserved even against an analyst that has access to this state (in addition to the answers to its statistical queries). For example, by using a pan-private mechanism and only storing the state, a curator could maintain a differential-privacy guarantee in case of a data breach. While this is an improvement over storing the database in plaintext, this approach has some limitations. For instance, since the pan-private mechanisms is lossy, the curator may not be able to recover its data. Another limitation is that it only guarantees differential privacy against *any* adversary, whereas it would be preferable to provide stronger guarantees against non-analyst adversaries. In other words, while differential privacy is a strong (and perhaps the best) guarantee possible against an adversary that is allowed to compute statistical queries on the data, one would like a stronger guarantee against adversaries that cannot make such queries. A third limitation is that many pan-private mechanisms achieve poor utility if used to guard against more than a single intrusion which severely limits their usefulness in protecting against data breaches, malicious servers etc.

Security via encrypted databases. Over the last fifteen years, the area of encrypted search has emerged as a promising solution to the problem of database security and data breaches. Using an encrypted database, a client/application can store and query its data on an untrusted server without exposing itself to the risk of insider and outsider threats, data breaches and even unintentional data disclosures (e.g., due to misconfigured access control lists). Solutions for various kinds of databases and with varying levels of security have been considered in the past, including for simple key-value stores (i.e., dictionaries) [17, 20], multi-maps [18, 21, 39, 41, 42, 54], graph databases [20, 51] and relational databases [37, 57]. The availability of such a diverse set of encrypted database solutions, motivates the following natural question:

> *Can we design private encrypted databases; that is, encrypted databases that support differentially-private statistical queries?*

With such a solution, a curator could store its database on any untrusted server while still allowing analysts to conduct private statistical analysis on the data. As a concrete example, the Census Bureau could outsource the storage and management of its OnTheMap data to a server in Amazon's cloud without losing any of the properties provided by its differentially-private mechanism.

At a high-level, our goal is to design encrypted databases that support both encrypted and private operations. The encrypted operations include query and update operations and are used by the curator to query and maintain the database. The private operations include statistical query operations used by the analyst to analyze the data. Roughly speaking, a private encrypted database should provide security against an untrusted server and differential-privacy against an untrusted analyst.

1.1 Our Contributions

In this work we introduce and address the problem of *securing* private statistical databases. We make the following contributions.

Private encrypted databases. One of the contributions of this work is to delineate the scope of the problem and to propose a set of reasonable properties that any encrypted and private statistical database should achieve. We do this formally by extending the notion of structured encryption [20] to support, in addition to a set of encrypted queries, a set of differentially-private queries. The encrypted queries are used by the curator to query and modify the database while the private queries are used by the analyst to analyze the data.

New adversarial models. In our setting, there are multiple adversaries to consider, including the server that stores the database, the analyst that analyzes the data and, possibly, an adversary that compromises the server at a point in time and gets a snapshot of the database.

More formally, the first corresponds to a *persistent* adversary that compromises the server perpetually and can monitor the transcripts of the interactions between the curator, the analyst and the server. The second corresponds to a *statistical* adversary that has access to the *results* of the private queries but not necessarily to the transcript of the entire interaction. This captures the standard untrusted analyst in differential privacy that tries to infer information about the data from the responses of its queries. The third is a *snapshot* adversary that gets multiple accesses to the database stored on the server but does not have access to the transcripts of

any interactions. This captures scenarios such as data breaches, devices thefts, and subpoenas.

There are several interesting subtleties that emerge from this setting that must be addressed. Recall that in the traditional setting of differential privacy, the curator is trusted so one does not consider any security/privacy guarantees for the analyst. In our setting, however, the database is stored on an untrusted server so the analyst's queries must also be protected. Another subtlety is that, in our setting, the data stored in the database is continually modified which requires differential privacy to hold under *continual observations*.

An encrypted database for private histogram queries. As a concrete goal, we focus on designing private encrypted databases that support differentially-private histogram queries. Histograms are one of the most common and central queries in statistical analysis and many other important queries can be formulated as histograms (e.g., contingency tables, marginals).

A naive approach to solving our problem would be for the curator to use a structured encryption scheme to produce an EDB and then use a pan-private histogram mechanism to produce a pan-private state. The curator would then use the EDB to query and manage its data and the server would use the pan-private state to answer analyst queries. The main limitation of this approach, however, is that it only guarantees differential privacy against persistent and snapshot adversaries whereas we would like a stronger guarantee.

To achieve this, we have to combine techniques from structured encryption and differential privacy in a more careful manner. Our first step, is to design an encrypted counter that supports an encrypted add operation and a differentially-private read operation. We build such a counter using additively homomorphic encryption and a differentially-private counter of Chan, Shi and Song [19]. We refer to the resulting construction as CPX. We then combine a standard encrypted data structure with several instantiations of our encrypted counter to build a private encrypted database that supports private histogram queries. There are several subtleties that come up—not only due to our new adversarial models—but also due to subtle and potentially dangerous interactions between the operations on the encrypted database and the contents of the private counter. To avoid these pitfalls, we have to carefully consider how the queries of the curator are executed so that they do not affect the security of the CPX counters. This results in a scheme we call HPX that has the same (encrypted) query complexity as its underlying encrypted database and (encrypted) update complexity that is linear in the underlying encrypted database's update complexity and in the number of histogram bins. When HPX's encrypted counters are instantiated with CPX, it has private query complexity that is logarithmic in the number of updates[1].

We note that designing a framework that composes *any* differentially private algorithm with *any* structured encryption scheme (while maintaining efficiency) is nontrivial problem and we leave it as an open problem.

2 Related Work

Differential privacy. In their seminal work [25], Dwork, McSherry, Nissim and Smith introduced the notion of differential privacy that guarantees privacy of individuals by ensuring similar outputs of queries on data irrespective of whether an individual's information is present or absent in the data. Since its conception, differential privacy has been an active research area [6–9, 29, 44, 50], we refer the readers to [24, 28] for a comprehensive survey.

Differential privacy has also been deployed in practice, see for example the systems used at Google [30, 31], Apple [22], the US Census Bureau [2, 49] and Uber [35, 63].

Differential privacy with continual observations. Dwork, Naor, Pitassi and Rothblum introduced a new setting for differential privacy [26] where the data is continuously being modified. The motivation was to consider scenarios where data analysis required repeated computations over dynamic data, for example, real-time traffic analysis, social trends observation and disease outbreak discovery. Calandrino, Kilzer, Narayanan, Felten and Shmatikov [12] showed that continual release of statistics leaks more information to the adversary and is a bigger privacy threat. In [26], the authors construct an ϵ-differentially-private *continual counter* with a small error. Specifically, they show that with probability $1-\delta$, the error at time step t is at most $O(\frac{1}{\epsilon}\log^{1.5} t \log\frac{1}{\delta})$. They also show that with probability at least δ, any ϵ-differentially-private mechanism for λ time steps must incur an additive error of at least $\Omega(\frac{1}{\epsilon}(\log\lambda + \log\frac{1}{\delta}))$. Chan, Shi and Song, in an independent work, also constructed a similar counter with the same privacy and

1 While the HPX construction considers fixed number of histogram bins, Section A of Appendix explains how to extend HPX to setup histogram bins dynamically.

error guarantees [19]. The continual observation model has been widely adopted and several problems have been considered in the literature [3, 13–16, 45, 47, 59, 64, 65].

Structured encryption. Structured encryption (STE) was introduced by Chase and Kamara in [20] as a generalization of searchable symmetric encryption (SSE) [21, 60]. There are STE constructions for various data structures including multi-maps [10, 11, 17, 18, 20, 21, 32, 38, 39, 41, 42, 54], graphs [20, 36, 51, 67] and two-dimensional arrays [20, 43].

Snapshot security. Snapshot security was discussed informally in several works [34, 56] but first formalized in the context of property-preserving encryption (PPE) by Lewi and Wu [46] and in the context of STE by Amjad, Kamara and Moataz in [5].

3 Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. $[n]$ is the set of integers $\{1,\ldots,n\}$, and $2^{[n]}$ is the corresponding power set. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. Given a sequence $\mathbf{v}$ of n elements, we refer to its ith element as v_i or $\mathbf{v}[i]$. If S is a set then $|S|$ refers to its cardinality. If x and y are two integer sequences of length n, then $||\cdot||$ denotes the Hamming distance. We denote by $\mathsf{Lap}(b)$ the Laplace distribution with parameter b that has probability density function $\frac{1}{2b}e^{-\frac{|x|}{b}}$ with mean 0 and variance $2b^2$.

Abstract data types. An *abstract data type* specifies the functionality of a data structure. It is a collection of data objects together with a set of operations defined on those objects. Examples include sets, dictionaries (also known as key-value stores or associative arrays) and graphs. The operations associated with an abstract data type fall into one of the two categories: query operations, which return information about the objects; and update operations, which modify the objects. If the abstract data type supports only query operations it is *static*, otherwise it is *dynamic*. We denote by $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{U}$ the query, response and update spaces of the data object, respectively.

Data structures. A *data structure* for a given data type is a representation in some computational model of an object of the given type.[2] Typically, the representation is optimized to support the type's query operation as efficiently as possible. For data types that support multiple queries, the representation is often optimized to efficiently support as many queries as possible. As a concrete example, the dictionary *type* can be represented using various data structures depending on which queries one wants to support efficiently. Hash tables support Get and Put in expected $O(1)$ time whereas balanced binary search trees support both operations in worst-case $\log(n)$ time. For ease of understanding and to match colloquial usage, we will sometimes blur the distinction between data types and structures.

Public-key encryption. A public key encryption scheme is a set of three polynomial-time algorithms $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ such that Gen is a probabilistic algorithm that takes a security parameter k and returns a pair of private and public keys $(\mathrm{sk}, \mathrm{pk})$; Enc is a probabilistic algorithm that takes a public key pk and a message m and returns a ciphertext ct; Dec is a deterministic algorithm that takes a private key sk and a ciphertext c and returns m if pk was the public key under which ct was produced. A public key encryption scheme is an *additive homomorphic encryption* scheme if for any two messages m_1 and m_2, and any $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathsf{Gen}(1^k)$ for all $k \in \mathbb{N}$, we have that

$$\mathsf{Dec}\big(\mathrm{sk}, \mathsf{Enc}(\mathrm{pk}, m_1) \cdot \mathsf{Enc}(\mathrm{pk}, m_2)\big) = m_1 + m_2.$$

Informally, a public key encryption scheme is secure against chosen-plaintext attacks if the ciphertexts it outputs do not reveal any partial information about the plaintext even to an adversary that can adaptively query an encryption oracle.

3.1 Structured Encryption

Structured encryption (STE) schemes [20] encrypt data structures in such a way that they can be queried. STE schemes can be distinguished depending on the type of operations they support. This includes *non-interactive* and *interactive* schemes where the former require only a single message while the latter require several rounds for queries and updates. STE schemes can also be *static* or *dynamic* where the former do not support update operations whereas the latter do. We can also distinguish between *response-revealing* and *response-hiding* schemes where the former reveal the response to queries whereas the latter do not.

2 In this work, the underlying model will always be the word RAM.

Interactive response-revealing STE schemes are used as follows. During the setup phase, the client constructs an encrypted data structure EDS under a key K. If the scheme is stateful, the setup also outputs a state st. The server then receives EDS from the client. During the query phase, the client and server execute a two-party protocol where the client inputs its query q, key K and state st while the server inputs the encrypted structure EDS. The client receives a response r and an updated state st' while the server receives nothing. Similarly, during the update phase, either an add or remove, the client and server execute a two-party protocol where the client inputs its update u^+/u^-, key K and state st while the server inputs the encrypted structure EDS. The client receives an updated state st' while the server receives an updated structure EDS'. We formally define an interactive response-revealing STE as follows:

Definition 3.1 (Structured encryption)**.** *An interactive structured encryption scheme* $\Sigma_{\mathsf{STE}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Add}, \mathsf{Remove})$ *consists of one polynomial-time algorithm and three interactive protocols that work as follows:*

- $(st, K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*: is a probabilistic algorithm that takes as inputs the security parameter* k *and a data structure* DS *and receives a state* st*, a key* K*, and an encrypted data structure* EDS*.*
- $(st', r; \perp) \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}(st, K, q; \mathsf{EDS})$ *is a (probabilistic) protocol between the client* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The client inputs its state* st*, the key* K *and the query* q*, while the server inputs the encrypted data structure* EDS*. The client receives a response* r *and the server receives nothing.*
- $(st'; \mathsf{EDS}') \leftarrow \mathsf{Add}_{\mathbf{C},\mathbf{S}}(st, K, u^+; \mathsf{EDS})$*: is a (probabilistic) protocol between the client* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The client inputs its state* st*, the key* K *and the update* $u^+ \in \mathbb{U}$*, while the server inputs the encrypted data structure* EDS*. As output, the client receives an updated state* st' *and the server receives an updated state* EDS'*.*
- $(st'; \mathsf{EDS}') \leftarrow \mathsf{Remove}_{\mathbf{C},\mathbf{S}}(st, K, u^-; \mathsf{EDS})$*: is a (probabilistic) protocol between the client* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The client inputs its state* st*, the key* K *and the update* $u^- \in \mathbb{U}$*, while the server inputs the encrypted data structure* EDS*. As output, the client receives an updated state* st' *and the server receives an updated state* EDS'*.*

We defer the correctness and security definition of an interactive structured encryption scheme to Appendix B.

A note on operation failures. Similarly to plaintext data structures, operations over encrypted structures may fail. For queries, failures are in general the result of an empty response or the nonexistence of the query in the structure. We typically capture a query failure by a null $\perp$ element added to the data type response space. However, update failures are in general more complex. For example, in set data structures, an add update might fail if the element already exists in the structure. An update also might fail if it intends to remove an element that does not exist or simply because the structure is empty. At a high level, we say that an update *fails* if the update does not modify the encrypted structure. Formally, an update $u^+ \in \mathbb{U}$ fails if $\mathsf{EDS} = \mathsf{EDS}'$ where $(st'; \mathsf{EDS}') \leftarrow \mathsf{Add}_{\mathbf{C},\mathbf{S}}(st, K, u; \mathsf{EDS})$ for all $k \in N$, for all $\mathsf{poly}(k)$-size structures DS, for all (st, K, EDS) output by $\mathsf{Setup}(1^k, \mathsf{DS})$ (similary for Remove).[3]

A note on data structures and databases. Our main construction, HPX, can be used to support histogram queries on any encrypted data structure. Unfortunately, most of the differential privacy and cryptography literature does not distinguish between data structures (e.g., arrays, two-dimensional arrays, dictionaries, multi-maps, trees etc.) and databases (e.g., relational databases, NoSQL databases). So to remain consistent with the literature, in the remainder of this paper we will refer to arbitrary data structures as databases. So anytime we refer to a database DB or an encrypted database EDB what we mean is some arbitrary data structure DS and some arbitrary encrypted data structure EDS, respectively.

4 Private Structured Encryption

In this Section, we extend the notion of STE to support private queries. We refer to the resulting primitive as a private structured encryption (PSTE) scheme. With a PSTE scheme, a curator can encrypt its database in such a way that it can query and manage its database through a set of encrypted queries and such that an analyst can query the database through a set of private queries. In Definition 4.1 below, we describe the syntax of a PSTE scheme. Here, we only describe schemes that support one encrypted and one private query operations but the syntax can be extended to support schemes with

3 The encrypted structure can also be the output of a previous update protocol executions. The definition can be naturally extended.

multiple encrypted and private queries in the natural way.

Definition 4.1 (Private structured encryption)**.** *An interactive private structured encryption scheme* $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ *consists of five polynomial-time protocols that work as follows:*

- $\big((st, K_{\mathbf{C}}); \mathsf{PEDB}; K_{\mathbf{A}}\big) \leftarrow \mathsf{Setup}_{\mathbf{C},\mathbf{S},\mathbf{A}}\big((1^k, \epsilon, \mathsf{DB}); \perp; \perp\big)$*: is a three-party protocol between the curator* $\mathbf{C}$*, the server* $\mathbf{S}$ *and the analyst* $\mathbf{A}$*. The curator inputs the security parameter* 1^k*, the privacy parameter* ϵ *and a database* DB*, while the server and the analyst input nothing. The curator receives a state* st *and a key* $K_{\mathbf{C}}$*, the server receives an encrypted database* PEDB *and the analyst receives a key* $K_{\mathbf{A}}$*.*
- $(st'; \mathsf{PEDB}') \leftarrow \mathsf{EAdd}_{\mathbf{C},\mathbf{S}}\big((st, K_{\mathbf{C}}, u^+); \mathsf{PEDB}\big)$*: is a two-party protocol between the curator* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The curator inputs its state* st*, a key* $K_{\mathbf{C}}$ *and an update* $u^+ \in \mathbb{U}$*, while the server inputs the encrypted database* PEDB*. As output, the curator receives the updated state* st' *and the server receives an updated encrypted database* PEDB'*.*
- $(st'; \mathsf{PEDB}') \leftarrow \mathsf{ERemove}_{\mathbf{C},\mathbf{S}}\big((st, K_{\mathbf{C}}, u^-); \mathsf{PEDB}\big)$ *is a two-party protocol between the curator* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The curator inputs its state* st*, a key* $K_{\mathbf{C}}$ *and an update* $u^- \in \mathbb{U}$*, while the server inputs the encrypted database* PEDB*. As output, the curator receives the updated state* st' *and the server receives an updated encrypted database* PEDB'*.*
- $\big((st', r); \perp\big) \leftarrow \mathsf{EQuery}_{\mathbf{C},\mathbf{S}}\big((st, K_{\mathbf{C}}, q); \mathsf{PEDB}\big)$ *is a two-party protocol between the curator* $\mathbf{C}$ *and the server* $\mathbf{S}$*. The curator inputs its state* st*, a key* $K_{\mathbf{C}}$ *and a query* q*, while the server inputs the encrypted database* PEDB*. The curator receives a response* r *and an updated state* st'*, and the server receives nothing.*
- $(r^p; \perp) \leftarrow \mathsf{PQuery}_{\mathbf{A},\mathbf{S}}\big((K_{\mathbf{A}}, q^p); \mathsf{PEDB}\big)$*: is a two-party protocol between the analyst* $\mathbf{A}$ *and the server* $\mathbf{S}$*. The analyst inputs its key* $K_{\mathbf{A}}$ *and its private query* $q^p \in \mathbb{P}$*, while the server inputs the encrypted database* PEDB*. As output, the analyst receives a private response* $r^p \in \mathbb{O}$ *while the server receives nothing.*

For visual clarity, we sometimes omit the subscripts of the protocols when the parties involved is clear from the context.

A PSTE scheme is used as follows. During a setup phase, the curator, server and analyst execute the Setup protocol on the curator's database DB. This results in a secret key $K_{\mathbf{C}}$ and state st for the curator, an encrypted database PEDB for the server and a secret key $K_{\mathbf{A}}$ for the analyst. To query the database, the curator executes the EQuery protocol with the server. To manage the database (i.e., to add or remove items), it executes the EAdd and $\mathsf{ERemove}$ protocols with the server. To perform a statistical query on the data, the analyst executes the PQuery protocol with the server. Note that the syntax of a PSTE scheme is similar to that of an STE scheme since the curator uses the STE operations to manage its encrypted database. The main difference is the addition of a PQuery protocol which is needed for the analyst to perform its analytical queries.

4.1 Correctness and Utility

We say that a PSTE scheme is correct if the encrypted query protocol always returns the correct response with high probability (possibly) on an updated encrypted database output by an update operation (addition or removal operation) that has previously occurred. Moreover, correctness also should hold independently of the private queries made by the analyst. We formally define correctness as follows.

Definition 4.2 (Correctness)**.** *Let* $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ *be a PSTE scheme. We say that* Δ *is correct if for all* $k \in \mathbb{N}$*, for all* $\epsilon > 0$*, for all* $\mathsf{poly}(k)$*-size databases* DB_0*, for all* $(st, K_{\mathbf{C}}; \mathsf{PEDB}_0; K_{\mathbf{A}})$ *output by* $\mathsf{Setup}\big((1^k, \epsilon, \mathsf{DB}_0); \perp; \perp\big)$ *and all* $\mathsf{poly}(k)$*-size sequences of operations* $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_\lambda)$*, for all* $i \in [\lambda]$*, if* $\sigma_i = q_i$*, then* $\mathsf{EQuery}(st_{i-1}, K_{\mathbf{C}}, q_i; \mathsf{PEDB}_{i-1})$ *returns, with all but negligible probability, a response* r_i *equal to the result of querying* DB_{i-1} *on* q_i*, where* PEDB_{i-1} *and* DB_{i-1} *result from applying all the update operations in* $(\sigma_1, \ldots, \sigma_{i-1})$ *to the encrypted database* PEDB_0 *and the plaintext database* DB_0*, respectively.*

Utility. The correctness definition above guarantees that the curator will get correct responses to its encrypted queries (with high probability) but does not say anything about the utility of the analyst's private queries. Intuitively, in the setting of differential privacy we say that a mechanism is useful if its responses are close to the correct responses. Here, we apply the same intuition to the encrypted database's PQuery operation. More precisely, we say that PQuery is (α, δ)-useful if, with probability at least $1 - \delta$, it produces responses within an additive factor of α to the true response. We formally define this below.

Definition 4.3 (Utility)**.** *Let* $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ *be a PSTE scheme. We say that* Δ *is*

(α, δ)-useful if for all $\mathsf{poly}(k)$*-size sequences of operations* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\lambda)$*, for all* $i \in [\lambda]$*, if* $\sigma_i = q_i^p$*,*

$$\Pr\left[\, |r_i^p - r_i| \leq \alpha \,\right] \geq 1 - \delta$$

where $(r_i^p; \perp) \leftarrow \mathsf{PQuery}\big((K_{\mathbf{A}}, q_i^p); \mathsf{PEDB}_{i-1}\big)$*,* r_i *is the correct response and* PEDB_{i-1} *results from applying all the updates in* $(\sigma_1, \ldots, \sigma_{i-1})$ *to the encrypted database* PEDB_0 *generated with* Setup*.*

4.2 Security and Privacy

As discussed in Section 1, our adversarial setting is more complex than that of STE and of differential privacy. In particular, in our context we have to consider two adversaries: namely, the server and the analyst. In addition, the server can be corrupted in two possible ways: a persistent corruption and a snapshot corruption. We summarize these models as follows:

- a *persistent adversary* corrupts the server perpetually. It has access to the encrypted database and can monitor both the encrypted and private query operations. This captures a corrupted server (e.g., an untrusted cloud service or a possibly corrupted on-premise server).
- a *snapshot adversary* is an adversary that corrupts the server only at fixed points in time. It can access a copy of the encrypted database at the time of corruption, but cannot see the encrypted or private query operations. This models certain kinds of data-breaches, subpoenas and thefts.
- a *statistical adversary* is an adversary that corrupts the analyst. It can see the responses to the private queries but cannot see encrypted private database or the encrypted query/update operations. This captures an untrusted analyst.

Collusions. While providing security against each of these adversaries is non-trivial, we will discuss in Section 7 how to maintain the same security guarantees when some of these adversaries collude. Specifically, we will consider the cases when: (1) a snapshot adversary colludes with a statistical adversary; and (2) a persistent adversary colludes with a statistical adversary. Note that the case of a snapshot adversary colluding with a persistent adversary reduces to the case of a persistent adversary.

Security against a persistent adversary. The security of PSTE schemes is a natural extension of the security of standard STE schemes. Intuitively, we require that a PSTE scheme guarantees that the encrypted database reveals no information about its underlying database beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$; that the encrypted query protocol reveals no information about the database and the query beyond the query leakage $\mathcal{L}_{\mathsf{Q}}$; that the add and remove protocols reveal no information about the database and the updates u^+ and u^-, respectively, than the add and remove leakages $\mathcal{L}_{\mathsf{A}}$ and $\mathcal{L}_{\mathsf{R}}$; and that the private query protocol reveals no information about the database and the private query q^p beyond the private query leakage $\mathcal{L}_{\mathsf{P}}$. If this holds for non-adaptively chosen operations then this is referred to as non-adaptive security. If, on the other hand, the operations can be chosen adaptively, we have the stronger notion of adaptive security [20, 21]. Observe that the security definition is parametrized with leakage functions for the setup, query and update operations. An example of a concrete setup leakage pattern is the size of the database. Examples of concrete query leakage patterns are the query equality pattern (also known as the search pattern in the SSE literature) or the response identity pattern (also known as the access pattern in the SSE literature). Note that different schemes have different leakage profiles; we refer the reader to [20, 21, 40] for more details.

Definition 4.4 (Adaptive persistent security)**.** *Let* $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ *be a private structured encryption scheme. Consider the following probabilistic experiments where* $\mathcal{A}$ *is stateful adversary,* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are stateful simulators,* $\mathcal{C}$ *is a challenger,* $\mathcal{L}_{\mathsf{S}}$*,* $\mathcal{L}_{\mathsf{Q}}$*,* $\mathcal{L}_{\mathsf{A}}$*,* $\mathcal{L}_{\mathsf{R}}$*, and* $\mathcal{L}_{\mathsf{P}}$ *are leakage profiles, and* $z \in \{0,1\}^*$*:*

$\mathbf{Real}_{\Delta,\mathcal{A}}(k, \epsilon)$*: given* z*, the adversary* $\mathcal{A}$ *outputs a database* DB *and receives an encrypted database* PEDB*, where* $(st, K; \mathsf{PEDB}; K_{\mathbf{A}}) \leftarrow \mathsf{Setup}\big((1^k, \epsilon, \mathsf{DB}); \perp; \perp\big)$*. The adversary then adaptively chooses a* $\mathsf{poly}(k)$*-size sequence of operations* $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_\lambda)$*. For all* $i \in [\lambda]$*, if*

- *(add operations) if* σ_i *is an add operation* u_i^+*, then the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$ *execute*

$$(st'; \mathsf{PEDB}') \leftarrow \mathsf{EAdd}\big((st, K, u_i^+); \mathsf{PEDB}\big),$$

  with $\mathcal{C}$ *playing the curator and* $\mathcal{A}$ *playing the server.*
- *(remove operations) if* σ_i *is a remove operation* u^-*, then the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$ *execute*

$$(st'; \mathsf{PEDB}') \leftarrow \mathsf{ERemove}\big((st, K, u_i^-); \mathsf{PEDB}\big),$$

  with $\mathcal{C}$ *playing the curator and* $\mathcal{A}$ *playing the server.*

– *(encrypted query) if σ_i is an encrypted query q_i, then the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ execute*

$$(st', r; \perp) \leftarrow \mathsf{EQuery}\big((st, K, q_i); \mathsf{PEDB}\big),$$

with $\mathcal{C}$ playing the curator and $\mathcal{A}$ playing the server.
– *(private query) if σ_i is a private query q_i^p, then the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ execute*

$$(r^p; \perp) \leftarrow \mathsf{PQuery}\big((K_{\mathbf{A}}, q_i^p); \mathsf{PEDB}\big),$$

with $\mathcal{C}$ playing the analyst and the adversary playing the server.

Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.

Ideal$_{\Delta,\mathcal{A},\mathcal{S}_1,\mathcal{S}_2}(k,\epsilon)$*: given z, the adversary $\mathcal{A}$ generates a database* DB*. The simulator $\mathcal{S}_1$ is then given z and setup leakage $\mathcal{L}_{\mathsf{S}}(\mathsf{DB})$ from which it outputs an encrypted database* PEDB*. Given* PEDB*, $\mathcal{A}$ adaptively chooses a* poly(k)*-size sequence of operations $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_\lambda)$. For all $i \in [\lambda]$,*

– *(add operation) if σ_i is an add operation u_i^+, then $\mathcal{S}_1(\mathcal{L}_{\mathsf{A}}(\mathsf{DB}, u_i^+))$ and $\mathcal{A}$ execute* EAdd*, with the simulator playing the curator and the adversary playing the server.*
– *(remove operation) if σ_i is a remove operation u^-, then $\mathcal{S}_1(\mathcal{L}_{\mathsf{R}}(\mathsf{DB}, u_i^-))$ and $\mathcal{A}$ execute* ERemove*, with the simulator playing the curator and the adversary playing the server.*
– *(encrypted query) if σ_i is an encrypted query q_i, then $\mathcal{S}_1(\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, q_i))$ and $\mathcal{A}$ execute* EQuery*, with the simulator playing the curator and the adversary playing the server.*
– *(private query) if σ_i is a private query q_i^p, then $\mathcal{S}_2(\mathcal{L}_{\mathsf{P}}(\mathsf{DB}, q_i^p))$ and $\mathcal{A}$ execute* PQuery*, with the simulator playing the analyst and the adversary playing the server.*

Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.

We say that Δ is adaptively $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{A}}, \mathcal{L}_{\mathsf{R}}, \mathcal{L}_{\mathsf{P}})$-secure if there exists PPT *simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ such that for all* PPT *adversaries $\mathcal{A}$, for all $\epsilon > 0$ and all $z \in \{0,1\}^*$, the following expression is negligible in k:*

$$|\Pr[\mathbf{Real}_{\Delta,\mathcal{A}}(k,\epsilon) = 1] - \Pr[\mathbf{Ideal}_{\Delta,\mathcal{A},\mathcal{S}_1,\mathcal{S}_2}(k,\epsilon) = 1]|$$

Security against a snapshot adversary. Security against a persistent adversary is a strong notion of security in that it guarantees security against an adversary that not only has access to the encrypted database but also to transcripts of the operations. In many real-world scenarios, however, we are concerned with a weaker adversary that can periodically access the encrypted database but, in particular, cannot access any of the query transcripts. This captures, for example, certain kinds of data breaches, malicious employees, thefts and subpoenas. This kind of adversary is referred to as a *snapshot* adversary and it was recently formalized in the context of structured encryption by Amjad, Kamara and Moataz [5].

In the following Definition, we adapt the formalization of [5] to our context. Intuitively, the adversary is given access to multiple snapshots of the encrypted database each of which is interspersed with a batch of encrypted and private queries (and updates). We then require that encrypted database reveals no information about the underlying database and the sequence of operations executed prior to the snapshot, beyond some snapshot leakage $\mathcal{L}_{\mathsf{SN}}$. In the worst case, the adversary can have a snapshot after every operation.

Definition 4.5 (Snapshot security)**.** *Let $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ be a private structured encryption scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, $\mathcal{C}$ is a stateful challenger, $\mathcal{L}_{\mathsf{SN}}$ is a stateful leakage function, $z \in \{0,1\}^*$, and $m \geq 1$:*

Real$_{\Delta,\mathcal{A}}(k,\epsilon,m)$*: given z, the adversary $\mathcal{A}$ outputs an initial database* DB_0*. The challenger then computes $(st, K_{\mathbf{C}}; \mathsf{PEDB}_0; K_{\mathbf{A}}) \leftarrow \mathsf{Setup}\big(1^k, \epsilon, \mathsf{DB}_0; \perp; \perp\big)$. Given PEDB_0, $\mathcal{A}$ outputs an adaptively chosen* poly(k)*-size sequence $\boldsymbol{\sigma}_1 = (\sigma_{1,1}, \ldots, \sigma_{1,m})$. For all $i \in [n]$,*

– *the challenger applies the operations in $\boldsymbol{\sigma}_i$ to PEDB_{i-1} which results in PEDB_i;*
– *given PEDB_i, the adversary outputs a sequence of operations $\boldsymbol{\sigma}_{i+1} = (\sigma_{i+1,1}, \cdots, \sigma_{i+1,m})$ where $\boldsymbol{\sigma}_{i+1}$ refers to the sequence of operations between the i^{th} and the $(i+1)^{th}$ snapshots, where $\sigma_{i+1,j} \in \{$*PQuery*, *EAdd*, *ERemove*, *EQuery*$\}$;*

Finally, $\mathcal{A}$ outputs a bit b that is returned by the experiment.

Ideal$_{\Delta,\mathcal{A},\mathcal{S}}(k,\epsilon,m)$*: given z, the adversary $\mathcal{A}$ outputs a database* DB_0*. The simulator $\mathcal{S}(z, \mathcal{L}_{\mathsf{SN}}(\mathsf{DB}_0, \perp))$ simulates* PEDB_0*. The adversary $\mathcal{A}(\mathsf{PEDB}_0)$ outputs an adaptively chosen* poly(k)*-size sequence $\boldsymbol{\sigma}_1 = (\sigma_{1,1}, \cdots, \sigma_{1,m})$. For all $i \in [n]$,*

– *the challenger applies the operations in $\boldsymbol{\sigma}_i$ to DB_{i-1} which results in DB_i;*
– *the simulator $\mathcal{S}(\mathcal{L}_{\mathsf{SN}}(\mathsf{DB}_i, \boldsymbol{\sigma}_i))$ simulates PEDB_i;*
– *given PEDB_i, the adversary $\mathcal{A}$ outputs a sequence of operations $\boldsymbol{\sigma}_{i+1} = (\sigma_{i+1,1}, \cdots, \sigma_{i+1,m})$;*

Finally, $\mathcal{A}$ outputs a bit b that is output by the experiment.

We say that Δ is $(n, \mathcal{L}_{\mathsf{SN}})$-snapshot secure if there exists a PPT *simulator $\mathcal{S}$ such that for all* PPT *adversaries $\mathcal{A}$, for all $\epsilon > 0$ and all $z \in \{0,1\}^*$, the following expression is negligible in k:*

$$\left|\Pr\left[\mathbf{Real}_{\Delta,\mathcal{A}}(k,\epsilon,m)=1\right] - \Pr\left[\mathbf{Ideal}_{\Delta,\mathcal{A},\mathcal{S}}(k,\epsilon,m)=1\right]\right|$$

Breach resistance. In [5], the authors formalize the notion of a *breach-resistant* STE scheme as a scheme with snapshot leakage that reveals at most the size of the underlying structure at the time of the snapshot. We extend this to the setting of private structured encryption by allowing the snapshot leakage to include, in addition, the size of the private query space.

Definition 4.6 (Breach resistance)**.** *Let $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ be an $(m, \mathcal{L}_{\mathsf{SN}})$-snapshot secure private structured encryption scheme. We say that Δ is breach-resistant if for all $m = \mathsf{poly}(k)$,*

$$\mathcal{L}_{\mathsf{SN}}(\mathsf{DB}, \boldsymbol{\sigma}_i) = \Big(|\mathsf{DB}_i|, |\mathbb{P}|\Big)$$

where DB_i is the structure that results from applying $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_i$ to DB_0 and $\mathbb{P}$ is the private query space.

Remark. Note that our security definitions for PSTE against both persistent and snapshot adversaries are inspired from previous works [5, 20, 21]. The main difference, however, is that we allow the analyst to perform PQuery operations as part of the overall sequence of operations. In fact, the analyst's queries can also impact the encrypted structure so an additional leakage pattern needs to be considered in our security definitions.

Security against a statistical adversary. We now turn to our notion of privacy against a statistical adversary which captures privacy against an untrusted analyst. Specifically, we wish to guarantee differential privacy against the analyst. Informally, differential privacy formalizes privacy by requiring that the probability of the output of a mechanism is roughly the same for any two "neighboring" databases.

Continual observations. In our setting, we do not have a fixed database so we cannot use the standard definition of differential privacy. Because our database is dynamic we need a variant referred to as differential privacy under continual observations [26]. This definition is formalized as follows. Let $\mathcal{M}$ be a privacy mechanism, let $S \subseteq \mathsf{Im}(\mathcal{M})^\lambda$ and let DB_0 be a database. The definition says that $\mathcal{M}$ is ϵ-differentially-private over continual observations if for all neighboring sequences of curator operations $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\lambda)$ and $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_\lambda)$,

$$\Pr\left[\Big(\mathcal{M}(\mathsf{DB}_1), \ldots, \mathcal{M}(\mathsf{DB}_\lambda)\Big) \in S\right] \leq e^\epsilon \cdot \Pr\left[\Big(\mathcal{M}(\mathsf{DB}'_1), \ldots, \mathcal{M}(\mathsf{DB}'_\lambda)\Big) \in S\right],$$

where DB_i results from applying σ_i to DB_{i-1} and DB'_i results from applying σ'_i to DB'_{i-1}. In our context, the PQuery protocol is effectively a private mechanism so it would be tempting to just apply the definition above.

Variable queries. The difficulty with this is that in the definition above, the mechanism $\mathcal{M}$ is a fixed operation that has no operand whereas PQuery allows the analyst to choose an operand q^p. There are several possible ways to handle this but here we take the worst-case which is to assume that, after each curator operation, the analyst makes every possible private query $q^p \in \mathbb{P}$. If we can quantify the privacy against such an analyst, it will provide an upper bound on the privacy we achieve against all possible analysts.

Before we formalize this intuition, we describe our notion of neighboring sequences. Let an operation σ be composed of an operator/operand pair (op, o). We consider two sequences of curator *update* operations $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\lambda)$ and $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_\lambda)$ neighbors if they differ by at most a single *update* operation and operand; that is, there exists at most a single $i \in [\lambda]$ such that $(\mathsf{op}_i, o_i) \neq (\mathsf{op}'_i, o'_i)$.

Definition 4.7 (Differential privacy)**.** *Let $\Delta = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PQuery})$ be a private structured encryption scheme. We say that Δ is ϵ-differentially-private (under continual observations) if for all neighboring sequences of curator operations $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\lambda)$ and $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_\lambda)$, and for any subset $S \subseteq \mathbb{O}^\lambda$,*

$$\Pr\left[\Big(\big(r_{1,q^p}\big)_{q^p\in\mathbb{P}}, \ldots, \big(r_{\lambda,q^p}\big)_{q^p\in\mathbb{P}}\Big) \in S\right] \leq e^\epsilon \cdot \Pr\left[\Big(\big(r'_{1,q^p}\big)_{q^p\in\mathbb{P}}, \ldots, \big(r'_{\lambda,q^p}\big)_{q^p\in\mathbb{P}}\Big) \in S\right],$$

where r_{i,q^p} is the result from executing $(r_{i,q^p}; \perp) \leftarrow \mathsf{PQuery}(K_{\mathbf{A}}, q^p; \mathsf{PEDB}_i)$ and PEDB_i results from applying the curator operations $(\sigma_1, \ldots, \sigma_i)$ to PEDB_0. Here r'_{i,q^p} is defined analogously.

5 CPX: A Private Encrypted Counter

A counter is a data structure that stores an integer and supports Add and Read operations. Add takes as input a counter ctr and an integer $a \in \{-1, 0, 1\}$ and returns an updated counter ctr′ that stores $c + a$, where c is the current value of the counter. The Read operation takes as input a counter ctr and returns its current value. Counters are a basic data structure that are heavily used throughout all of computer science. A *private* counter [19, 26] is a counter that supports a differentially-private read operation. This is typically achieved by adding noise to the counter value. In this section, we are concerned with designing an *encrypted* private counter which supports encrypted add operations and private read operations. Note that CPX builds on the binary mechanism of Chan et al. [19]. While CPX follows almost the same steps as [19], it adds noise and increments the counters homomorphically as opposed to over plaintext values. Our modification also relies on slight changes to the way the counter is set up. Below, we recall the binary mechanism and describe our new encrypted add operations.

The binary mechanism. The binary mechanism is a differentially-private (under continual observations) counter introduced by Chan, Shi and Song [19] which offers reasonable privacy and utility tradeoffs. The counter supports increment, decrement and no-op operations represented as additions of values in $\{-1, 0, 1\}$. When read, it returns a value that is the true counter with some noise added to it.

The counter is represented as a *range tree* with a local register. The local register just stores the total number of addition operations performed until now. A range tree is a complete binary tree T that stores values in such a way that all the values within a range can be computed efficiently. More precisely, range trees maintain a tree with λ leaves associated to λ values. Each node in the tree represents a unique range: the tth leaf represents the range $[t, t]$ whereas an internal node that is the root of a subtree with leaves ranging from t_1 to t_2 represents the range $[t_1, t_2]$. The *cover* of a range $[t_1, t_2]$ is defined as the set of nodes such that the union of the ranges of the nodes is equal to $[t_1, t_2]$. The *minimum cover* is the cover with minimum cardinality. Range trees have the property that the size of the minimum cover of a range $[1, t]$, for any t, is at most $2 \log t$. We abuse the notation and refer to the minimum cover of $[1, t]$ as the minimum cover of t.

During setup, the binary mechanism takes as input a parameter λ denoting the maximum number of addition operations to be performed on the counter, and it initializes a range tree with λ leaves. Each node stores a value of 0. For the tth add operation, it takes as input a value $a \in \{-1, 0, 1\}$, and adds a to all the nodes on the path from $[t, t]$ to the root. It also adds noise $\gamma_t \leftarrow \mathsf{Lap}(2 \log \lambda / \epsilon)$ to all the nodes of the form $[\cdot, t]$ on this path. To answer read operations, the mechanism outputs the sum of the nodes in the minimum cover of t, where t is the current register value which denotes the number of add operations performed until this point.

We now describe an encrypted private counter based on the binary mechanism called CPX. Our construction is described in Fig. (1) and works as follows.

Setup. The Setup algorithm takes as input the security parameter 1^k, the privacy parameter ϵ and the number of update operations to be made λ. The curator initializes a binary tree T with λ leaves where every node consists of an additively-homomorphic encryption of 0. The private encrypted counter ectr is a tuple composed of the tree T, the privacy parameter ϵ, a time step t initialized to 0 and the public key of the homomorphic encryption.

Encrypted add. To add an integer $a \in \{-1, 0, 1\}$ to the encrypted counter, the curator sends the encryption of the integer ct_a to the server. The server, given the time step t, first fetches all the nodes in the path P_t starting from the root and ending at the leaf $[t, t]$. The server then adds ct_a to every node in P_t. The server then selects all nodes in P_t with a range of the form $[., t]$, to which it adds the encryption of a freshly sampled noise. The server also increments the internal time step t.

Private read. To read the value of the counter after the ith operation, the server simply sends to the analyst the sum of all nodes belonging to the minimum cover of $[1, i]$ which the analyst later decrypts.

5.1 Efficiency and Utility

The complexity of both the encrypted add and private query are

$$O(\log \lambda \cdot \mathsf{time}_{\mathsf{AHE}}),$$

where λ is the upper bound on the number of updates while $\mathsf{time}_{\mathsf{AHE}}$ is the time complexity of a homomorphic

Let $\mathsf{Lap}(\cdot)$ denote the Laplace distribution. Let $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. Consider the private encrypted counter scheme $\Delta_{\mathsf{ctr}} = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{PRead})$ defined as follows:

– $\mathsf{Setup}\left(1^k, \epsilon, \lambda, \mathsf{ctr}\right)$:
1. generate $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{AHE.Gen}(1^k)$;
2. initialize a binary tree T with λ leaves;
3. for each node $\mathsf{N} \in \mathsf{T}$,
(a) set $\mathsf{N} = \mathsf{AHE.Enc}(\mathrm{pk}, 0)$;
4. set $t = 0$;
5. $\mathbf{C}$ receives $K_{\mathbf{C}} = \mathrm{pk}$, $\mathbf{S}$ receives $\mathsf{ectr} = (\mathsf{T}, \epsilon, t, \mathrm{pk})$ and $\mathbf{A}$ receives $K_{\mathbf{A}} = \mathrm{sk}$;

– $\mathsf{EAdd}_{\mathbf{C},\mathbf{S}}\left(K_{\mathbf{C}}, a; \mathsf{ectr}\right)$:
1. $\mathbf{S}$ parses ectr as $(\mathsf{T}, \epsilon, t, \mathrm{pk})$;
2. $\mathbf{S}$ sets $t = t + 1$;
3. $\mathbf{C}$ sends $\mathrm{ct}_a \leftarrow \mathsf{AHE.Enc}(\mathrm{pk}, a)$ to $\mathbf{S}$;
4. $\mathbf{S}$ does the following
(a) it finds the path P_t from the $[t, t]$ leaf of T to its root;
(b) for all nodes N in P_t, it sets $\mathsf{N} = \mathsf{N} \cdot \mathrm{ct}_a$;
(c) it creates a subset $S \subset P_t$ of nodes in P_t whose range has the form $[\cdot, t]$;
(d) for each node N in S,
i. it computes $\gamma_{\mathsf{N}} \leftarrow \mathsf{Lap}(\frac{2 \log \lambda}{\epsilon})$;
ii. it sets $\mathsf{N} = \mathsf{N} \cdot \mathsf{AHE.Enc}(\mathrm{pk}, \gamma_{\mathsf{N}})$;
(e) $\mathbf{C}$ receives $\perp$ and $\mathbf{S}$ receives an updated encrypted counter $\mathsf{ectr}' = (\mathsf{T}', \epsilon, t, \mathrm{pk})$;

– $\mathsf{PRead}_{\mathbf{A},\mathbf{S}}\left(\perp; \mathsf{ectr}\right)$:
1. $\mathbf{S}$ does the following:
(a) it parses ectr as $(\mathsf{T}, \epsilon, t, \mathrm{pk})$;
(b) it finds the minimal cover C_t of the interval $[1, t]$;
(c) it sends $\mathrm{ct}_r = \sum_{\mathsf{N} \in C_t} \mathsf{N}$ to $\mathbf{A}$;
2. $\mathbf{A}$ computes $r \leftarrow \mathsf{AHE.Dec}(\mathrm{sk}, \mathrm{ct}_r)$;
3. $\mathbf{A}$ receives r and $\mathbf{S}$ receives $\perp$;

Fig. 1. CPX: Range tree based Encrypted private counter.

operation. This stems from the fact that the size of the minimum cover in a range tree is upper bounded by $2 \log \lambda$. Moreover, the round complexity of both protocols is equal to 1.

Theorem 5.1. *For each $t \in [\lambda]$,* CPX *is (α, δ)-useful, where $\alpha = O(\frac{1}{\epsilon} \cdot (\log \lambda)\sqrt{\log \lambda} \cdot \log \frac{1}{\delta})$, and $\epsilon, \delta > 0$.*

The proof of Theorem 5.1 is in Appendix. C.1.

5.2 Security

In the following, we detail the security guarantees of CPX against persistent, snapshot and statistical adversaries, respectively.

Against a persistent adversary. The setup leakage of CPX consists of the size of the tree which is equal to the upper bound on the number of updates λ such that

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{ctr}}(\mathsf{ctr}) = \lambda.$$

Both the add leakage and the private query leakage of CPX are equal to $\perp$ as they do not depend of the curator's input such that

$$\mathcal{L}_{\mathsf{A}}^{\mathsf{ctr}}(\mathsf{ctr}, a) = \perp \quad \text{and} \quad \mathcal{L}_{\mathsf{P}}^{\mathsf{ctr}}(\mathsf{ctr}, \perp) = \perp.$$

Theorem 5.2. *If* AHE *is CPA-secure, then CPX is $(\mathcal{L}_{\mathsf{S}}^{\mathsf{ctr}}, \mathcal{L}_{\mathsf{A}}^{\mathsf{ctr}}, \mathcal{L}_{\mathsf{P}}^{\mathsf{ctr}})$-secure.*

The proof of Theorem 5.2 is deferred to the full version of the paper.

Leakage against a snapshot adversary. The snapshot leakage $\mathcal{L}_{\mathsf{SN}}^{\mathsf{ctr}}$ of CPX is equal, for all $i \in [m]$, to

$$\mathcal{L}_{\mathsf{SN}}^{\mathsf{ctr}}(\mathsf{ctr}, \boldsymbol{\sigma}_i) = \lambda.$$

Theorem 5.3. *If* AHE *is CPA-secure, then CPX is $(m, \mathcal{L}_{\mathsf{SN}}^{\mathsf{ctr}})$-snapshot secure.*

The proof of Theorem 5.3 is deferred to the full version of the paper. Note that CPX is breach-resistant based on Definition 4.6 when $m = \mathsf{poly}(k)$.

Against a statistical adversary. As detailed above, CPX is an encrypted variant of the binary mechanism. In particular, the private responses of CPX have the same exact distribution (on the same sequences of operations) as the binary mechanism and, therefore inherits the same differential privacy parameters. In particular we have the following Theorem.

Theorem 5.4. *The* PRead *protocol in CPX is ϵ-differentially private as per Definition 4.7.*

The proof of this theorem is similar to the one of the binary mechanism in [19] and we defer it to the full version of this work.

6 HPX: An Encrypted Database for Private Histograms Queries

In this section, we describe our encrypted database scheme, HPX, which supports private histogram queries. With our solution, a curator can outsource its statistical database to an untrusted server. The curator can update the data using the EDB's encrypted queries and an analyst can analyze the data using the EDB's

private queries. We note that while our work relies on the counter of Chan et al., we address a different problem than the one considered in [19, 26]. While these works construct pan-private counters, we design an encrypted database that supports differentially-private histogram queries. In the setting of [19, 26], the curator is a trusted party that stores a pan-private counter and answers read queries from an untrusted analyst. In our setting, the curator first outsources its encrypted database to an untrusted server. It can then query and update its outsourced encrypted database by interacting with the server. In addition, an untrusted analyst can make differentially-private histogram queries on the encrypted database by interacting with the untrusted server. As one can see, we not only consider a different object than [19, 26], but our adversarial model is also very different: in addition, to an adversarial analyst, we also have to consider an adversarial server.

Overview. At a high-level, HPX works as follows. We are given a database DB and a histogram $\mathsf{H} = \{\mathsf{ctr}_1, \ldots, \mathsf{ctr}_n\}$ initialized to 0. We also assume the existence of a function Map_1 that maps database updates u to multiple counters in H. Our construction produces $(n+1)$ encrypted structures: EDB and $\mathsf{ectr}_1, \ldots, \mathsf{ectr}_n$. The first results from encrypting the database DB with an appropriate structured encryption scheme. The curator interacts with EDB to query and update its database. The structures $(\mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$ are private encryptions of the histogram counters $(\mathsf{ctr}_1, \ldots, \mathsf{ctr}_n)$ and are used by the server to answer the analyst's queries. Whenever the curator *successfully* updates its EDB with an update u^+, it also increments the ith counter ctr_i by 1, where $i \in \mathsf{Map}_1(u^+)$. To hide which counter was updated, however, it also increments all the remaining counters by 0.

The efficiency of our scheme depends on the efficiency of the underlying encrypted database EDB and is linear in n. In terms of security, we provide a black-box leakage analysis of HPX's leakage. We then show that if its underlying encrypted structures are instantiated with zero leakage or almost zero-leakage constructions then HPX's leakage profile is minimal.

HPX makes use of a dynamic response-hiding structured encryption scheme $\Sigma_{\mathsf{DB}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Add}, \mathsf{Remove})$ and a private encrypted counter scheme $\Delta_{\mathsf{ctr}} = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{PRead})$ (e.g., the construction described in Section 5). Our construction is described in detail in Figs. (2) and (3) and works as follows.

Setup. The Setup algorithm takes as input a database DB and the size of the histogram n. It initializes n counters $(\mathsf{ctr}_1, \ldots, \mathsf{ctr}_n)$. It then encrypts DB with Σ_{DB} which results in EDB, and encrypts each ctr_i with Δ_{ctr} which results in $(\mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$. The private EDB is $\mathsf{PEDB} = (\mathsf{EDB}, \mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$.

Encrypted queries. To query the EDB, the curator and server execute the $\Delta.\mathsf{Query}$ protocol where the curator inputs its query q and the server inputs PEDB. This protocol in turn executes EDB's query protocol on the same inputs and returns the response to the curator.

Database updates. To add or remove an item u to/from the database, the curator first updates EDB using the appropriate update protocol (either $\Sigma_{\mathsf{DB}}.\mathsf{Add}$ or $\Sigma_{\mathsf{DB}}.\mathsf{Remove}$). The server then returns a message to the curator indicating whether the operation succeeded or failed. The curator and server then update all the private encrypted counters $(\mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$ as follows. If the operation succeeded and $i \in \mathsf{Map}_1(u)$ (i.e., the update is associated to the ith counter) then ectr_i is incremented by 1 (in case of addition, otherwise it is decremented by 1), otherwise ectr_i is incremented by 0. Incrementing all the counters—even the ones that are not relevant to the update—hides which counter is updated from the server. This is crucial to hide the values of the histogram from the server. To see why, suppose we did not hide which counter is updated after an EDB operation (i.e., an EAdd or $\mathsf{ERemove}$). Note that the server knows whether the EDB operation was an add or a remove and whether the operation succeeded or failed. It can determine the success or failure simply by comparing EDB to EDB'. If, in addition to this, it also knew which counter got updated after the EDB operation, it would know whether that counter was incremented, decremented or kept constant. And from this information the server effectively knows the histogram.

Private queries. To query the ith counter, the analyst and the server execute a private read operation on ectr_i.

6.1 Efficiency and Utility

The query complexity of HPX is equal to the query complexity of the underlying database encryption scheme Σ_{DB}. In other words, its query complexity is

$$O\left(\mathsf{time}^q_{\mathsf{DB}}(q)\right),$$

where $\mathsf{time}^q_{\mathsf{DB}}$ is the query complexity Σ_{DB}. Similarly, the round complexity of the query protocol is equal to the round complexity of the query protocol of Σ_{DB}. The update complexity (whether it is an EAdd or a $\mathsf{ERemove}$)

Let $\Sigma_{\mathsf{DB}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Add}, \mathsf{Remove})$ be a database encryption scheme and let $\Delta_{\mathsf{ctr}} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{EAdd}, \mathsf{PRead})$ be a PSTE scheme for counters. Let λ be a upper bound on the number of updates and ϵ be the differential privacy parameter. Consider the PSTE scheme $\Delta_{\mathsf{HIS}} = (\mathsf{Setup}, \mathsf{EAdd}, \mathsf{ERemove}, \mathsf{EQuery}, \mathsf{PHist})$ defined as follows:

– $\mathsf{Setup}_{\mathbf{C},\mathbf{S},\mathbf{A}}\Big(\big(1^k, \epsilon, \mathsf{DB}\big); \perp; \perp\Big)$:

1. $\mathbf{C}$ does the following:
(a) computes $(K_{\mathsf{DB}}, st_{\mathsf{DB}}, \mathsf{EDB}) \leftarrow \Sigma_{\mathsf{DB}}.\mathsf{Setup}(1^k, \mathsf{DB})$;
(b) for all $i \in [n]$, it computes

$$(K^i_{\mathbf{C}}; \mathsf{ectr}_i; K^i_{\mathbf{A}}) \leftarrow \Delta_{\mathsf{ctr}}.\mathsf{Setup}_{\mathbf{C},\mathbf{S},\mathbf{A}}(1^k, \epsilon, \lambda, \mathsf{ctr}_i);$$

2. $\mathbf{C}$ receives $K_{\mathbf{C}} = (K_{\mathsf{DB}}, K^1_{\mathbf{C}}, \ldots, K^n_{\mathbf{C}})$ and $st_{\mathbf{C}} = st_{\mathsf{DB}}$;
3. $\mathbf{S}$ receives $\mathsf{PEDB} = (\mathsf{EDB}, \mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$;
4. $\mathbf{A}$ receives $K_{\mathbf{A}} = (K^1_{\mathbf{A}}, \ldots, K^n_{\mathbf{A}})$.

– $\mathsf{EAdd}_{\mathbf{C},\mathbf{S}}\Big(\big(st_{\mathbf{C}}, K_{\mathbf{C}}, u^+\big); \mathsf{PEDB}\Big)$:

1. $\mathbf{C}$ parses $st_{\mathbf{C}}$ as st_{DB} and $K_{\mathbf{C}}$ as $(K_{\mathsf{DB}}, K_1, \ldots, K_n)$;
2. $\mathbf{S}$ parses PEDB as $(\mathsf{EDB}, \mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$;
3. $\mathbf{C}$ and $\mathbf{S}$ execute

$$(st'_{\mathsf{DB}}; \mathsf{EDB}') \leftarrow \Sigma_{\mathsf{DB}}.\mathsf{Add}_{\mathbf{C},\mathbf{S}}(st_{\mathsf{DB}}, K_{\mathsf{DB}}, u^+; \mathsf{EDB});$$

4. if $\mathsf{EDB} = \mathsf{EDB}'$ then $\mathbf{S}$ sets $\mathsf{fail} = 1$ otherwise it sets $\mathsf{fail} = 0$;
5. $\mathbf{S}$ sends fail to $\mathbf{C}$;
6. for all $i \in [n]$,
(a) if $\mathsf{fail} = 0$ and $i \in \mathsf{Map}_1(u)$, then $\mathbf{C}$ and $\mathbf{S}$ execute

$$(\perp; \mathsf{ectr}'_i) \leftarrow \Delta_{\mathsf{ctr}}.\mathsf{EAdd}_{\mathbf{C},\mathbf{S}}(K_i, +1; \mathsf{ectr}_i);$$

(b) otherwise they execute

$$(\perp; \mathsf{ectr}'_i) \leftarrow \Delta_{\mathsf{ctr}}.\mathsf{EAdd}_{\mathbf{C},\mathbf{S}}(K_i, 0; \mathsf{ectr}_i);$$

7. $\mathbf{C}$ receives an updated state $st'_{\mathbf{C}} = st'_{\mathsf{DB}}$;
8. $\mathbf{S}$ receives an updated structure $\mathsf{PEDB}' = (\mathsf{EDB}', \mathsf{ectr}'_1, \ldots, \mathsf{ectr}'_n)$.

– $\mathsf{ERemove}_{\mathbf{C},\mathbf{S}}\Big(\big(st_{\mathbf{C}}, K_{\mathbf{C}}, u^-\big); \mathsf{PEDB}\Big)$:

– similar to EAdd except that
1. on line (3) $\mathbf{C}$ computes $\Sigma_{\mathsf{DB}}.\mathsf{Remove}_{\mathbf{C},\mathbf{S}}(st_{\mathsf{DB}}, K_{\mathsf{DB}}, u^-; \mathsf{EDB})$;
2. on line (6a) $\mathbf{C}$ and $\mathbf{S}$ execute

$$(\perp; \mathsf{ectr}'_i) \leftarrow \Delta_{\mathsf{ctr}}.\mathsf{EAdd}_{\mathbf{C},\mathbf{S}}(K_i, -1; \mathsf{ectr}_i);$$

– $\mathsf{EQuery}_{\mathbf{C},\mathbf{S}}\Big(\big(st_{\mathbf{C}}, K_{\mathbf{C}}, q\big); \mathsf{PEDB}\Big)$:

1. $\mathbf{C}$ parses $st_{\mathbf{C}}$ as st_{DB} and $K_{\mathbf{C}}$ as K_{DB};
2. $\mathbf{S}$ PEDB as $(\mathsf{EDB}, \mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$;
3. $\mathbf{C}$ and $\mathbf{S}$ execute

$$(r; \perp) \leftarrow \Sigma_{\mathsf{DB}}.\mathsf{Query}_{\mathbf{C},\mathbf{S}}(st_{\mathsf{DB}}, K_{\mathsf{DB}}, q; \mathsf{EDB})$$

4. $\mathbf{C}$ receives r;
5. $\mathbf{S}$ receives $\perp$.

Fig. 2. HPX: our PSTE construction for histograms (Part 1).

– $\mathsf{PHist}_{\mathbf{A},\mathbf{S}}\Big(\big(K_{\mathbf{A}}, q^p\big); \mathsf{PEDB}\Big)$:

1. $\mathbf{A}$ parses $K_{\mathbf{A}}$ as $(K_1, \ldots, K_n)$;
2. $\mathbf{S}$ parses PEDB as $(\mathsf{EDB}, \mathsf{ectr}_1, \ldots, \mathsf{ectr}_n)$;
3. $\mathbf{A}$ and $\mathbf{S}$ execute

$$(r^p; \perp) \leftarrow \Delta_{\mathsf{ctr}}.\mathsf{PRead}_{\mathbf{A},\mathbf{S}}\Big(K_{q^p}; \mathsf{ectr}_{q^p}\Big)$$

4. $\mathbf{A}$ outputs r^p and $\mathbf{S}$ outputs $\perp$.

Fig. 3. HPX: our PSTE construction for histograms (Part 2).

of HPX is

$$O\Big(\mathsf{time}^u_{\mathsf{DB}}(v) + n \cdot \mathsf{time}_{\mathsf{ctr}}(\lambda)\Big),$$

where $\mathsf{time}^u_{\mathsf{DB}}$ is the update complexity of the database encryption scheme Σ_{DB}, $\mathsf{time}_{\mathsf{ctr}}$ is the add complexity of the private counter encryption scheme Δ_{ctr}, n is the number of bins in the histogram, and λ is the upper bound on the number of updates. We stress that even though the update complexity is linear in n, the number of bins is often decided a-priori by the analyst and is usually a small constant [4, 48, 58, 66]. In this case, the update complexity of HPX is

$$O\Big(\mathsf{time}^u_{\mathsf{DB}}(v) + \mathsf{time}_{\mathsf{ctr}}(\lambda)\Big).$$

The round complexity of both EAdd and a $\mathsf{ERemove}$ is

$$O\Big(\mathsf{rounds}^u_{\mathsf{DB}}(v) + \mathsf{rounds}^a_{\mathsf{ctr}}(\lambda)\Big),$$

where $\mathsf{rounds}^u_{\mathsf{DB}}$ and $\mathsf{rounds}^a_{\mathsf{ctr}}$ are the round complexities of the update and the add protocols of Σ_{DB} and Δ_{ctr}, respectively. The private query complexity of HPX is

$$O\Big(\mathsf{time}^p_{\mathsf{ctr}}(\lambda)\Big),$$

where $\mathsf{time}^p_{\mathsf{ctr}}$ is the private query complexity of Δ_{ctr}. Similarly, the round complexity is equal to the round complexity of the private query complexity of Δ_{ctr}.

Additional remarks about update complexity. As discussed above, in practice, the number of bins is a small constant but there are applications where the number of bins might be $\omega(1)$. In particular, Sturges [61] shows that, when the data is approximately normally distributed, then setting n to

$$n = 1 + \lceil \log |\mathbb{U}| \rceil$$

is sufficient for data analytics, where $\mathbb{U}$ is the update space of the database. In this case, the update complexity is equal to

$$O\Big(\mathsf{time}^u_{\mathsf{DB}}(v) + \log |\mathbb{U}| \cdot \mathsf{time}_{\mathsf{ctr}}(\lambda)\Big).$$

Note also that there is a variant of our construction that reduces the update complexity to

$$O\Big(\mathsf{time}^{u}_{\mathsf{DB}}(v) + n\Big).$$

at the cost of increasing the private query complexity to

$$O\Big(\mathsf{time}^{p}_{\mathsf{ctr}}(\lambda) + \log\lambda \cdot \mathsf{time}_{\mathsf{ctr}}(\lambda)\Big)$$

Due to space limitation, we defer the details of this variant to the full version of this work.

Theorem 6.1. *For each $t \in [\lambda]$,* HPX *is (α, δ)-useful, where $\alpha = O(\frac{1}{\epsilon} \cdot (\log\lambda)\sqrt{\log\lambda} \cdot \log\frac{1}{\delta})$, and $\epsilon, \delta > 0$.*

The utility of HPX is equal to the utility of CPX. This is because for each private query, one of the counters is read and returned.

6.2 Security

In the following, we detail the security guarantees of HPX against standard, snapshot and statistical adversaries, respectively.

Security against a persistent adversary. The setup leakage of HPX consists of the setup leakage of its underlying building blocks. In particular, this includes the setup leakage of the database encryption scheme and the setup leakage of the encrypted private counter, for all n counters. The query leakage is equal to the query leakage of the underlying database encryption scheme. The update leakage, whether it is an add or remove leakage, includes the update leakage of the underlying database encryption scheme and, for all $i \in [n]$, the add leakage of the underlying encrypted private counter scheme. The private query leakage of a query consists of, the private query leakage of the underlying private counter scheme along with the counter identifier. We now give a precise description of HPX's leakage profile and show that it is adaptively-secure with respect to it. Its setup leakage is

$$\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = \Big(\mathcal{L}^{\mathsf{db}}_{\mathsf{S}}(\mathsf{DB}), \Big(\mathcal{L}^{\mathsf{ctr}}_{\mathsf{S}}(\mathsf{ctr}_i)\Big)_{i\in[n]}\Big).$$

Its query leakage is equal to

$$\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, q) = \mathcal{L}^{\mathsf{db}}_{\mathsf{Q}}(\mathsf{DB}, q).$$

Its respective add and remove leakages $\mathcal{L}_{\mathsf{A}}$ and $\mathcal{L}_{\mathsf{R}}$ on the respective add and remove updates u^+ and u^- are equal to for all $a \in \{-1, 0, 1\}$

$$\mathcal{L}_{\mathsf{A}}(\mathsf{DB}, u^+) = \Big(\mathcal{L}^{\mathsf{db}}_{\mathsf{A}}(\mathsf{DB}, u^+), \big(\mathcal{L}^{\mathsf{ctr}}_{\mathsf{A}}(\mathsf{ctr}_i, a)\big)_{i\in[n]}\Big),$$

$$\mathcal{L}_{\mathsf{R}}(\mathsf{DB}, u^-) = \Big(\mathcal{L}^{\mathsf{db}}_{\mathsf{R}}(\mathsf{DB}, u^-)), \big(\mathcal{L}^{\mathsf{ctr}}_{\mathsf{A}}(\mathsf{ctr}_i, a)\big)_{i\in[n]}\Big).$$

Its private query leakage is equal to

$$\mathcal{L}_{\mathsf{P}}(\mathsf{DB}, q^p) = \Big(q^p, \mathcal{L}^{\mathsf{ctr}}_{\mathsf{P}}(\mathsf{ctr}_{q^p}, \perp)\Big).$$

Theorem 6.2. *If Σ_{DB} is $(\mathcal{L}^{\mathsf{db}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{db}}_{\mathsf{Q}}, \mathcal{L}^{\mathsf{db}}_{\mathsf{A}}, \mathcal{L}^{\mathsf{db}}_{\mathsf{R}})$-secure, and Δ_{ctr} is $(\mathcal{L}^{\mathsf{ctr}}_{\mathsf{S}}, \mathcal{L}^{\mathsf{ctr}}_{\mathsf{A}}, \mathcal{L}^{\mathsf{ctr}}_{\mathsf{P}})$-secure, then HPX is $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{A}}, \mathcal{L}_{\mathsf{R}}, \mathcal{L}_{\mathsf{P}})$-secure.*

The proof of Theorem 6.2 deferred to the full version of the paper.

Security against a snapshot adversary. The snapshot leakage $\mathcal{L}_{\mathsf{SN}}$ of HPX is equal to the snapshot leakage of its underlying primitives such that

$$\mathcal{L}_{\mathsf{SN}}(\mathsf{DB}, \boldsymbol{\sigma}_i) = \Big(\mathcal{L}^{\mathsf{db}}_{\mathsf{SN}}(\mathsf{DB}, \boldsymbol{\sigma}_i), \Big(\mathcal{L}^{\mathsf{ctr}}_{\mathsf{SN}}(\mathsf{ctr}_j, \boldsymbol{\sigma}_i)\Big)_{j\in[n]}\Big).$$

Theorem 6.3. *If Σ_{DB} is $(m, \mathcal{L}^{\mathsf{db}}_{\mathsf{SN}})$-snapshot secure, and Δ_{ctr} is $(m, \mathcal{L}^{\mathsf{ctr}}_{\mathsf{SN}})$-snapshot secure, then HPX is $(m, \mathcal{L}_{\mathsf{SN}})$-snapshot secure.*

The proof of Theorem 6.3 is deferred to the full version of the paper. Clearly, if both Σ_{DB} and Δ_{ctr} are breach-resistant then HPX is breach-resistant as well based on Definition 4.6.

Leakage vs. update complexity and utility. Recall that the update complexity is linear in n because all the counters are updated during an HPX update. More precisely, the counters in $\mathsf{Map}_1(v)$ are incremented with $+1/-1$ and the rest are incremented with 0. This is to hide the identity of the counters in $\mathsf{Map}_1(v)$ from the adversaries. If one is willing to leak this, the update complexity improves to

$$O\Big(\mathsf{time}^{u}_{\mathsf{DB}}(v) + |\mathsf{Map}_1(v)| \cdot \mathsf{time}_{\mathsf{ctr}}(\lambda)\Big),$$

while the add/remove leakage and the snapshot leakage will now include, in addition, the identities of the updated counters $\mathsf{Map}_1(v)$. This will also improve the utility of the counters because, roughly speaking, they will be incremented less often so less noise will be added.

Security against a statistical adversary. To show that our HPX construction is differentially private, we need to show that PHist is differentially private as per Definition 4.7. For this, we first show that any two neighboring update operation sequences *induce* operation sequences on the counters that differ in at most two time places (refer to Lemma D.2).As the next step of the

proof, we show that, given that Δ_{ctr} is differentially private, PHist is differentially private. We state below our main theorem.

Theorem 6.4. *If Δ_{ctr} is ϵ-differentially-private, then HPX is $2\epsilon \cdot \max_{u \in \mathbb{U}}(|\mathsf{Map}_1(u)|)$-differentially private as per Definition 4.7.*

We provide the proof of Theorem 6.4 in Appendix D.1.

7 Handling Collusions

In this Section, we describe how our constructions can be extended to handle collusions.

Snapshot and statistical adversary. First, note that collusion between a snapshot and statistical adversary is equivalent to the adversary considered in the pan-privacy literature. In [19], Chan et al. construct a pan-private counter for a single unannounced intrusion and multiple announced intrusions based on the same binary mechanism described in Section 5. In the former, additional noise is added to all the nodes of the binary tree during setup to prevent the adversary from getting the real node values at the time of intrusion. This added noise protects against the intrusion but degrades the utility by a factor of 2. They then extend this method to support multiple announced intrusions by adding increasing amounts of noise and further degrading the utility. In particular, k intrusions reduce the utility by $O(\sqrt{k+1})$.

To handle this kind of collusion, it suffices to replace all the counters in CPX with the pan-private counters of [19]. To encrypt the database, we then use snapshot-secure STE scheme with zero-leakage; that is, with leakage that reveals at most some public parameter (e.g., the security parameter or any parameter that is chosen independently of the database). Note that this is simple to achieve by padding the database and encrypting it with a breach-resistant STE scheme [5].

Persistent and statistical adversary. This is a stronger setting than above because achieving differential privacy requires us to hide whether an add or a remove operation occurred. This is particularly difficult to do against an adversary that holds the encrypted database EDB because the size of EDB before and after an operation reveals which operation occured. This essentially requires us to use an EDB with zero-leakage updates. Another challenge is that many STE schemes leak the sizes of query responses which could be correlated with the size of EDB. To handle this, we need to use an STE scheme with zero-leakage queries. One possible instantiation is to make use of ORAM simulation [33] where all query responses in the database have to be padded to be equal to the same length– a function of some public parameter. This instantiation, however, will induce a polylogarithmic multiplicative overhead on both the query and update complexities.

Notice that when the persistent and statistical adversaries collude, there is no need to encrypt the counters since the analyst holds the decryption key. In addition, since the server samples the noise for the counters, it can easily infer the increments made. We can therefore use plaintext counters maintained in a simple register (i.e., there is no need to use the binary tree mechanism of Chan et al.) and use the following approach to maintain them. To increment the counters, the curator sends to the server an increment of $+1/-1/0$ with noise for each counter. Upon receiving the noisy increment, it adds it to the current value in the register. The PRead protocol returns the current register value. We note that this approach is similar to the second simple counting mechanism of [19]. In fact, using a proof very similar to that of Theorem 1.3 of [19], this approach can be shown to be ϵ-differentially private and (α, δ)-useful, where $\alpha = O\big(\frac{\sqrt{\lambda}}{\epsilon} \cdot \log \frac{1}{\delta}\big)$ and $\epsilon, \delta > 0$. Note that this approach meets Dwork *et al.*'s lower bound of $\sqrt{k}$ on utility for k intrusions [26], proving that this construction is almost tight.

8 Efficiency Estimates

In this Section, we estimate the efficiency of our construction. To do this we conducted micro-benchmarks of all the building blocks and then estimated the cost of the various HPX operations. We now explain why our methodology is sound. Recall that the update complexity of HPX is

$$\mathsf{time}^{u}_{\mathsf{DB}}(v) + n \cdot \mathsf{time}_{\mathsf{ctr}}(\lambda),$$

and $\mathsf{time}_{\mathsf{ctr}}(\lambda) = \log \lambda \cdot \mathsf{time}_{\mathsf{AHE}}$. Since, the add complexity of the private encrypted counter is independent of the data and is fixed, the update complexity of HPX on real data depends solely on the update complexity of the database encryption scheme $\mathsf{time}^{u}_{\mathsf{DB}}$ (since we use the database encryption scheme in a black-box way). Note that this time depends on the specific instantiation used and these running times have been well documented in

the relevant works [5, 11, 17, 18]. The same argument holds for encrypted queries and private queries.

Experimental setup. We consider the case where the additively-homomorphic encryption scheme is instantiated with Paillier [55]. We further assume the database is a multi-map and is encrypted using DLS, the dual-secure multi-map encryption scheme of Amjad, Kamara and Moataz [5]. We used implementations of Paillier and DLS from the `Javallier` [1] and `Clusion` libraries [52], respectively. Our test environment was a MacBook Pro 3.1 GHz Intel Core i7 with 16 GB of memory.

Micro-benchmarks. We instantiated Paillier with a 2048-bit key. Key generation, encryption, decryption and addition took on average 446, 70, 18 and 0.08 ms, respectively. Experiments in [5] reported that with a multi-map MM holding 10 millions pairs, their multi-map encryption scheme DLS took 10 minutes during Setup, 1 microsecond during Get and 33 ms during Put. We used Sturges' formula on our dataset to determine the bins, which resulted in 25 bins. We set $\lambda = 2^{32} \approx 4.3$ billion.

Estimates. Based on the micro-benchmarks above, for a multi-map holding 10 million pairs, we have the following estimates for the HPX operations:

- Setup takes $\mathsf{time}_{\mathsf{MM}}^{\mathsf{setup}} + n \cdot \mathsf{time}_{\mathsf{AHE}}^{\mathsf{keygen}} \approx 10.19$ mins.
- EAdd and ERemove take

$$\mathsf{time}_{\mathsf{MM}}^{\mathsf{put}} + n \cdot \log \lambda \cdot (\mathsf{time}_{\mathsf{AHE}}^{\mathsf{add}} + \mathsf{time}_{\mathsf{AHE}}^{\mathsf{enc}}) \approx 0.945 \text{ ms.}$$

- EQuery takes $\mathsf{time}_{\mathsf{MM}}^{\mathsf{get}} = 1$ microsecond.
- PQuery takes $\log \lambda \cdot \mathsf{time}_{\mathsf{AHE}}^{\mathsf{add}} + \mathsf{time}_{\mathsf{AHE}}^{\mathsf{dec}} \approx 21.172$ ms.

9 Acknowledgement

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Javallier. https://github.com/snipsco/paillier-libraries-benchmarks/tree/master/java-javallier.

[2] J. Abowd. The challenge of scientific reproducibility and privacy protection for statistical agencies., 15 September 2016. https://www2.census.gov/cac/sac/meetings/2016-09/2016-abowd.pdf.

[3] G. Acs and C. Castelluccia. A case study: privacy preserving release of spatio-temporal density in paris. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1679–1688. ACM, 2014.

[4] G. Acs, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *2012 IEEE 12th International Conference on Data Mining*, pages 1–10. IEEE, 2012.

[5] G. Amjad, S. Kamara, and T. Moataz. Breach-resistant structured encryption. *IACR Cryptology ePrint Archive*, 2018:195, 2018.

[6] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 901–914. ACM, 2013.

[7] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–282. ACM, 2007.

[8] G. Barthe, M. Gaboardi, E. J. Gallego Arias, J. Hsu, A. Roth, and P.-Y. Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. *ACM SIGPLAN Notices*, 50(1):55–68, 2015.

[9] A. Blum, K. Ligett, and A. Roth. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, 60(2):12, 2013.

[10] R. Bost. Sophos - forward secure searchable encryption. In *ACM Conference on Computer and Communications Security (CCS '16)*, 20016.

[11] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *ACM Conference on Computer and Communications Security (CCS '17)*, 2017.

[12] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. " you might also like:" privacy risks of collaborative filtering. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 231–246. IEEE, 2011.

[13] J. Cao, Q. Xiao, G. Ghinita, N. Li, E. Bertino, and K.-L. Tan. Efficient and accurate strategies for differentially-private sliding window queries. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 191–202. ACM, 2013.

[14] Y. Cao and M. Yoshikawa. Differentially private real-time data release over infinite trajectory streams. In *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, volume 2, pages 68–73. IEEE, 2015.

[15] Y. Cao and M. Yoshikawa. Differentially private real-time data publishing over infinite trajectory streams. *IEICE TRANSACTIONS on Information and Systems*, 99(1):163–175, 2016.

[16] Y. Cao, M. Yoshikawa, Y. Xiao, and L. Xiong. Quantifying differential privacy under temporal correlations. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 821–832. IEEE, 2017.

[17] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[18] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[19] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information and System Security (TISSEC)*, 14(3):26, 2011.

[20] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[21] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[22] A. Differential Privacy Team. Learning with privacy at scale.

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

[24] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[25] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006.

[26] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 715–724. ACM, 2010.

[27] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[28] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[29] C. Dwork and G. N. Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

[30] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[31] G. Fanti, V. Pihur, and Ú. Erlingsson. Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proceedings on Privacy Enhancing Technologies*, 2016(3):41–61, 2016.

[32] E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

[33] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[34] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why your encrypted database is not secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 162–168. ACM, 2017.

[35] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *Vertica*, 1:1000.

[36] S. Kamara. Restructuring the NSA metadata program. In *Workshop on Applied Homomorphic Cryptography*, Lecture Notes in Computer Science. Springer, 2014.

[37] S. Kamara and T. Moataz. Sql on structurally-encrypted databases. *IACR Cryptology ePrint Archive*, 2016:453, 2016.

[38] S. Kamara and T. Moataz. SQL on structurally-encrypted databases. *IACR Cryptology ePrint Archive*, 2016:453, 2016.

[39] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 94–124, 2017.

[40] S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 339–370. Springer, 2018.

[41] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[42] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[43] S. Kamara and L. Wei. Garbled circuits via structured encryption. In *Financial Cryptography Workshops*, pages 177–188, 2013.

[44] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[45] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias. Differentially private event sequences over infinite streams. *Proceedings of the VLDB Endowment*, 7(12):1155–1166, 2014.

[46] K. Lewi and D. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *ACM Conference on Computer and Communications Security (CCS '16)*, 2016.

[47] H. Li, L. Xiong, and X. Jiang. Differentially private synthesization of multi-dimensional data using copula functions. In *Advances in database technology: proceedings. International Conference on Extending Database Technology*, volume 2014, page 475. NIH Public Access, 2014.

[48] H. Li, L. Xiong, X. Jiang, and J. Liu. Differentially private histogram publication for dynamic datasets: an adaptive sampling approach. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1001–1010. ACM, 2015.

[49] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 277–286. IEEE Computer Society, 2008.

[50] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 94–103.

IEEE, 2007.

[51] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 504–517. ACM, 2015.

[52] T. Moataz. Clusion. https://github.com/encryptedsystems/Clusion.

[53] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[54] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 639–654. IEEE, 2014.

[55] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[56] R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591, 2016.

[57] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[58] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment*, 6(14):1954–1965, 2013.

[59] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux. Quantifying location privacy. In *Security and privacy (sp), 2011 ieee symposium on*, pages 247–262. IEEE, 2011.

[60] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[61] H. A. Sturges. The choice of a class interval. *Journal of the american statistical association*, 21(153):65–66, 1926.

[62] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[63] K. Tezapsidis. Uber releases open source project for differential privacy, 13 July 2017. https://medium.com/uber-security-privacy/differential-privacy-open-source-7892c82c42b6.

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

[65] Y. Xiao, L. Xiong, L. Fan, and S. Goryczka. Dpcube: differentially private histogram release through multidimensional partitioning. *arXiv preprint arXiv:1202.5358*, 2012.

[66] J. Xu, Z. Zhang, X. Xiao, Y. Yang, G. Yu, and M. Winslett. Differentially private histogram publication. *The VLDB Journal - The International Journal on Very Large Data Bases*, 22(6):797–822, 2013.

[67] Y. Zhang, A. O'Neill, M. Sherr, and W. Zhou. Privacy-preserving network provenance. *Proc. VLDB Endow.*, 10(11):1550–1561, Aug. 2017.

A Setting Up Counters Dynamically

We provide a high level idea of how a private counter can be set up dynamically at run time. Assume the new counter ectr_N needs to be set up between the t^{th} and $t+1^{th}$ update operations. As a first step, the curator, the server and the analyst execute $\Delta_{\mathsf{ctr}}.\mathsf{Setup}$ to initialize the new counter ectr_N at the server. The curator then queries EDB with $\Sigma_{\mathsf{DB}}.\mathsf{Query}$ to compute the count of ectr_N. Let r be the response.

Depending on how Query is implemented, r itself can be the count for ectr_N or it can be a set whose cardinality is the count of ectr_N. Here, assume that r is the count. Since each update operation can increment a counter at most by 1, we know that $t \geq r$. The curator then initializes ectr_N by executing $\Delta_{\mathsf{ctr}}.\mathsf{EAdd}$ a total of r times with value 1, and $\Delta_{\mathsf{ctr}}.\mathsf{EAdd}$ a total of $t-r$ times with value 0.

We note that this dynamic setup process is different than the standard one but that it only affects the leakage provided to a persistent adversary. For example, the server now learns the exact count r of ectr_N as well as the "query" associated with the new counter. However, the new scheme remains breach-resistant since the snapshot adversary only learns that a new counter has been setup. Moreover, it is also differentially private against the analyst since the steps taken to initialize the counter in the middle are equivalent to the steps that would have been taken, had the counter been set up in the beginning.

B Correctness and security definition of STE

Security. The standard notion of security for STE guarantees that: (1) an encrypted structure reveals no information about its underlying structure beyond the setup leakage $\mathcal{L}_{\mathsf{S}}$; (2) that the query protocol reveals no information about the structure and the queries beyond the query leakage $\mathcal{L}_{\mathsf{Q}}$; and that (3) the add/remove protocols reveal no information about the structure and

the updates u^+/u^- beyond the add/remove leakages $\mathcal{L}_{\mathsf{A}}/\mathcal{L}_{\mathsf{R}}$.

If this holds for non-adaptively chosen operations then the scheme is said to be non-adaptively secure. If, on the other hand, the operations can be chosen adaptively, the scheme is said to be adaptively-secure.

Definition B.1 (Adaptive security of interactive STE)**.** *Let $\Sigma = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Add}, \mathsf{Remove})$ be an interactive dynamic STE scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator, $\mathcal{L}_{\mathsf{S}}$, $\mathcal{L}_{\mathsf{Q}}$, $\mathcal{L}_{\mathsf{A}}$and $\mathcal{L}_{\mathsf{R}}$ are leakage profiles and $z \in \{0,1\}^*$:*

- $\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$*: given z the adversary $\mathcal{A}$ outputs a structure* DS *and receives* EDS *from the challenger, where* $(st, K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$*. The adversary then* adaptively *chooses a polynomial number of operations* $\mathsf{op}_1, \ldots, \mathsf{op}_m$ *such that* op_i *is either a query* q_i *or an update* u_i^+ *or* u_i^-*. For all* $i \in [m]$*, if* $\mathsf{op}_i = q_i$*, the adversary and the challenger execute the protocol* Query*, and the challenger receives an updated state* st' *and a response* r *while the adversary receives nothing, where* $(st', r; \perp) \leftarrow \mathsf{Query}(st, K, q_i; \mathsf{EDS})$*. If,* $\mathsf{op}_i = u_i^+$*, the adversary and the challenger execute the protocol* Add*, and the challenger receives an updated state* st'*, while the adversary receives an updated encrypted structure* EDS'*, where* $(st'; \mathsf{EDS}') \leftarrow \mathsf{Add}(st, K, u_i^+; \mathsf{EDS})$*. If, on the other hand,* $\mathsf{op}_i = u_i^-$*, the adversary and the challenger execute the protocol* Remove*, and the challenger receives an updated state* st'*, while the adversary receives an updated encrypted structure* EDS'*, where* $(st'; \mathsf{EDS}') \leftarrow \mathsf{Remove}(st, K, u_i^-; \mathsf{EDS})$*. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*
- $\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$*: given z the adversary $\mathcal{A}$ generates a structure* DS *which it sends to the challenger. Given* z *and leakage* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS})$ *from the challenger, the simulator* $\mathcal{S}$ *returns an encrypted structure* EDS *to* $\mathcal{A}$*. The adversary then* adaptively *chooses a polynomial number of operations* $\mathsf{op}_1, \ldots, \mathsf{op}_m$ *such that* op_i *is either a query* q_i *or an update* u_i^+ *or* u_i^-*. For all* $i \in [m]$*, if* $\mathsf{op}_i = q_i$*, the simulator receives the query leakage* $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i)$ *and executes* $\mathsf{Query}(\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, q_i); \mathsf{EDS})$ *with the adversary. The adversary receives nothing as output. If,* $\mathsf{op}_i = u_i^+$*, the adversary the simulator receives the add leakage* $\mathcal{L}_{\mathsf{A}}(\mathsf{DS}, u_i^+)$ *and executes* $\mathsf{Add}(\mathcal{L}_{\mathsf{A}}(\mathsf{DS}, u_i^+); \mathsf{EDS})$ *with the adversary. The adversary receives an updated structure* EDS' *as output. If, on the other hand,* $\mathsf{op}_i = u_i^-$*, the adversary the simulator receives the remove leakage* $\mathcal{L}_{\mathsf{R}}(\mathsf{DS}, u_i^-)$ *and executes* $\mathsf{Remove}(\mathcal{L}_{\mathsf{R}}(\mathsf{DS}, u_i^-); \mathsf{EDS})$ *with the adversary. The adversary receives an updated structure* EDS' *as output. Finally,* $\mathcal{A}$ *outputs a bit* b *that is output by the experiment.*

We say that Σ is adaptively $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}}, \mathcal{L}_{\mathsf{A}}, \mathcal{L}_{\mathsf{R}})$-secure if there exists a PPT *simulator $\mathcal{S}$ such that for all* PPT *adversaries $\mathcal{A}$, for all $z \in \{0,1\}^*$,*

$$\left|\Pr\left[\mathbf{Real}_{\Sigma,\mathcal{A}}(k) = 1\right] - \Pr\left[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1\right]\right| \leq \mathsf{negl}(k).$$

Correctness. We say that a dynamic STE scheme Σ is correct if for all $k \in \mathbb{N}$, for all $\mathsf{poly}(k)$-size structures DS_0, for all $(K, st_0, \mathsf{EDS}_0)$ output by $\mathsf{Setup}(1^k, \mathsf{DS}_0)$, for all sequences of $m = \mathsf{poly}(k)$ operations $\mathsf{op}_1, \ldots, \mathsf{op}_m$ such that $\mathsf{op}_i \in \{q_i, u_i\}$, for all $i \in [m]$, if $\mathsf{op}_i = q_i$, $\mathsf{Query}(st_{i-1}, K, q_i; \mathsf{EDS}_{i-1})$ returns, with all but negligible probability, a response r_i equal to the result of querying DS_{i-1} with q_i; where st_{i-1} is the output of the Add, Remove or Query protocols, while EDS_{i-1} and DS_i result from applying all update operations in the sequence $(\mathsf{op}_1, \cdots, \mathsf{op}_{i-1})$ to EDS_0 and to DS_0, respectively.

C CPX Proofs

C.1 Proof of Theorem 5.1

Theorem 5.1. *For each $t \in [\lambda]$,* CPX *is (α, δ)-useful, where $\alpha = O(\frac{1}{\epsilon} \cdot (\log \lambda)\sqrt{\log \lambda} \cdot \log \frac{1}{\delta})$, and $\epsilon, \delta > 0$.*

Proof. The proof is similar to the proof of utility of Chan et al.'s counter [19]. The main difference is the sensitivity of the "sum" function. Chan et al. consider Add operands from $\{0,1\}$ whereas we consider Add operands from $\{-1,0,1\}$. Because of this the senstivity of our "sum" function is 2 as opposed to 1. In turn, this affects the differential privacy by a multiplicative factor of 2 but this can be factor traded-off for utility by scaling the Laplacian noise appropriately. Note that this factor is subsumed above in the asymptotic notation. □

D HPX Proofs

D.1 Proof of Theorem 6.4

Recall that in our construction, every update operation of the curator *induces* an EAdd operation on *all* the counters. The operand however differs from counter to

counter. For example, a successful ERemove of operand o by the curator leads to a -1 being added to all counters in $\mathsf{Map}_1(o)$, while 0 to the rest of the counters. Similarly, a successful EAdd of operand o leads to a 1 being added to all counters in $\mathsf{Map}_1(o)$, while 0 to the rest of the counters. Therefore, the operation that is induced on the counters is *always* EAdd but the operands come from $\{-1, 0, 1\}$. We therefore slightly abuse the notation and say that a curator's update operation induces an operation in $\{-1, 0, 1\}$ on the counters.

Before introducing our lemma and our main theorem, we set up some notations. Let $\boldsymbol{\sigma} = \big((\mathsf{op}_1, o_1), \ldots, (\mathsf{op}_\lambda, o_\lambda)\big)$ be an update operation sequence, where $\mathsf{op}_i \in \{\mathsf{EAdd}, \mathsf{ERemove}\}$ and $o_i \in \mathbb{U}$. For all $i \in [n]$, we define a new mapping Map_2^i that, intuitively, creates the operation sequences in $\{-1, 0, 1\}^\lambda$ for the ith counter. We call these sequences the *counter mapped sequences*. Formally, for each $j \in [\lambda]$, we have,

$$\mathsf{Map}_2^i[\sigma_j] = \begin{cases} 1 & \text{if } \mathsf{op}_j = \mathsf{EAdd} \text{ and } j \in \mathsf{Map}_1(o_j) \\ -1 & \text{if } \mathsf{op}_j = \mathsf{ERemove} \text{ and } j \in \mathsf{Map}_1(o_j) \\ 0 & \text{otherwise.} \end{cases}$$

Also, for every $i \in [n]$, we define another mapping Map_3^i, that captures whether the jth operation in the update operation sequence is successful or not. We call these sequences the *operation mapped sequences*. Formally,

$$\mathsf{Map}_3^i\Big[\mathsf{Map}_2^i[\sigma_j]\Big] = \begin{cases} \mathsf{Map}_2^i[\sigma_j] & \text{if } o_j \text{ succeeds} \\ 0 & \text{otherwise.} \end{cases}$$

We abuse the notation and we set

$$\mathsf{Map}_2^i[\boldsymbol{\sigma}] = \big(\mathsf{Map}_2^i[\sigma_j]\big)_{j\in[\lambda]}$$

and similarly

$$\mathsf{Map}_3^i\Big[\mathsf{Map}_2^i[\boldsymbol{\sigma}]\Big] = \left(\mathsf{Map}_3^i\Big[\mathsf{Map}_2^i[\sigma_j]\Big]\right)_{j\in[\lambda]}.$$

In the following, we show that any update operation sequences that differ in at most one operation, lead to two operation mapped sequences that differ in at most two operations.

We first start by showing that any two counter mapped sequences that differ in at most one operation, lead to two operation mapped sequences that differ in at most two operations. Note that this result only applies to *set-like* failures.

Lemma D.1. *Let $\boldsymbol{\sigma}$, $\boldsymbol{\sigma}'$ be any neighboring update operation sequences such that for all $i \in [n]$, $\big|\big|\mathsf{Map}_2^i[\boldsymbol{\sigma}] - \mathsf{Map}_2^i[\boldsymbol{\sigma}']\big|\big| \leq 1$, then we have*

$$\Big|\Big|\mathsf{Map}_3^i\big[\mathsf{Map}_2^i[\boldsymbol{\sigma}]\big] - \mathsf{Map}_3^i\big[\mathsf{Map}_2^i[\boldsymbol{\sigma}']\big]\Big|\Big| \leq 2$$

Proof. For simplicity assume that all operations (op_i, o_i) in the update operation sequence $\boldsymbol{\sigma}$ that map to the ith counter operate on the same operand. Formally, for all $i, j \in [\lambda]$, if $\mathsf{Map}_1(o_i) \cap \mathsf{Map}_1(o_j) \neq \perp$, then $o_i = o_j$. [4] We assume the same thing for $\boldsymbol{\sigma}'$ as well.

We now do the rest of the proof for counter i. Let $\mu = \mathsf{Map}_2^i[\boldsymbol{\sigma}]$, $\mu' = \mathsf{Map}_2^i[\boldsymbol{\sigma}']$, $\tilde{\mu} = \mathsf{Map}_3^i(\mu)$ and $\tilde{\mu}' = \mathsf{Map}_3^i(\mu')$, for any $i \in [n]$. We also denote by DB_0 and DB'_0 the underlying plaintext database before applying the sequence of operations in $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$, respectively.

Let $t \leq \lambda$ be the position at which the two sequences μ and μ' differ at. Then $\tilde{\mu}_j = \tilde{\mu}'_j$, for all $j < t$. This simply holds by definition of the operation mapped sequences. Note that this also implies that the databases, after the t first operations, contain the same items and we write $\mathsf{DB}_{t-1} = \mathsf{DB}'_{t-1}$. So in order to proof the lemma we need to show that in the remaining positions, i.e., for all $j \in \{t, \cdots, \lambda\}$, $\tilde{\mu}$ and $\tilde{\mu}'$ differ at most at two places. That is, our proof reduces to a counting problem where we need to count the number of indices in $\{t, \cdots, \lambda\}$ where $\tilde{\mu}$ and $\tilde{\mu}'$ differ.

Consider the tth operation μ_t and μ'_t. There are three possiblities where μ_t and μ'_t are different: (1) $\mu_t = 1, \mu'_t = -1$, (2) $\mu_t = 0, \mu'_t = -1$, or (3) $\mu_t = 1, \mu'_t = 0$. The other three cases are symmetrical and we ignore them without any loss of generality. We now try to infer what the t^{th} operand o_t and o'_t respectively must be in $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ in these three cases. In (1), $o_t = o'_t$ because otherwise either $\mathsf{Map}_2[\sigma_t] \neq i$ or $\mathsf{Map}_2[\sigma'_t] \neq i$ leading to either $\mu_t = 0$ or $\mu'_t = 0$. Let us define $o = o_t = o'_t$. In (2), let $o = o'_t$, while in (3) let $o = o_t$.

Let us now consider case (1) again (when $\mu_t = 1, \mu'_t = -1$) with o defined as above and count the number of indices in $\{t, \cdots, \lambda\}$ where $\tilde{\mu}$ and $\tilde{\mu}'$ differ. The other two cases can be proven on similar lines. Recall that $\mathsf{DB}_{t-1} = \mathsf{DB}'_{t-1}$, therefore there are two cases at this point : (a) $o \notin \mathsf{DB}_{t-1}, \mathsf{DB}'_{t-1}$; and (b) $o \in \mathsf{DB}_{t-1}, \mathsf{DB}'_{t-1}$. Again these cases are symmetrical, so we just describe case (a) in the following.

4 Note that the general case where operations may have different operands can be reduced to the *same operand* case detailed above. To see why, we can simply map different operands to different sub-sequences in such a way that every sub-sequence operates on the same operand. Note that there will be only one sub-sequence that have different operations in this case.

Since $o \notin \mathsf{DB}_{t-1}$ and $\mu_t = 1 \implies \mathsf{op}_t = \mathsf{EAdd}$, the operation must be successful. Therefore $\tilde{\mu}_t = 1$ and the database changes to DB_t such that $o \in \mathsf{DB}_t$. However, since $v \notin \mathsf{DB}'_{t-1}$ and $\mu'_t = -1 \implies \mathsf{ERemove}$, the operation fails. Therefore, $\tilde{\mu}'_t = 0$ and the database becomes DB'_t such that $v \notin \mathsf{DB}'_t$.

We now show that after applying the next operation μ_{t+1} to DB_t and μ'_{t+1} to DB'_t, either $\mathsf{DB}_{t+1} = \mathsf{DB}'_{t+1}$ and $\tilde{\mu}_{t+1} \neq \tilde{\mu}'_{t+1}$ (thereby producing the second differing index); or $\mathsf{DB}_{t+1} = \mathsf{DB}_t$ and $\mathsf{DB}'_{t+1} = \mathsf{DB}'_t$ but $\tilde{\mu}_{t+1} = \tilde{\mu}'_{t+1}$ (at which point the argument can be repeated until the next instance at which the operation mapping differs). Once the databases become the same, they remain similar for the remaining operations as all the operations in μ and μ' are the same. Hence no more different indices are produced. In particular, consider μ_{t+1} and μ'_{t+1} and all the possible values it can take:

1. $\mu_{t+1} = \mu'_{t+1} = 1$. Since $o \in \mathsf{DB}_t$, the operation μ_{t+1} fails. Therefore, $\tilde{\mu}_{t+1} = 0$ and the database does not change making $\mathsf{DB}_{t+1} = \mathsf{DB}_t$ such that $o \in \mathsf{DB}_{t+1}$. However, since $o \notin \mathsf{DB}'_t$ and $\mu'_{t+1} = 1$, the operation is successful. Therefore $\tilde{\mu}'_t = 1$ and the database changes to DB'_{t+1} such that $o \in \mathsf{DB}'_t$. Notice that $\mathsf{DB}_{t+1} = \mathsf{DB}'_{t+1}$.
2. $\mu_{t+1} = \mu'_{t+1} = -1$. This is similar to the above. In the end, $o \notin \mathsf{DB}_{t+1}, \mathsf{DB}'_{t+1}$ and $\mathsf{DB}_{t+1} = \mathsf{DB}'_{t+1}$.
3. $\mu_{t+1} = \mu'_{t+1} = 0$. Since both μ_{t+1} and μ'_{t+1} are 0, the databases do not change but $\tilde{\mu}_{t+1} = \tilde{\mu}'_{t+1}$.

This ends our proof. □

While the previous lemma assumes the counter mapped sequences to be neighbors, we show that the lemma even holds when we assume only the update operation sequences to be neighbors.

Lemma D.2. *Given any two update operation sequences $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ such that $||\boldsymbol{\sigma} - \boldsymbol{\sigma}'|| \leq 1$, and for any $i \in [n]$, we have*

$$\left|\left|\mathsf{Map}_3^i\left[\mathsf{Map}_2^i[\boldsymbol{\sigma}]\right] - \mathsf{Map}_3^i\left[\mathsf{Map}_2^i[\boldsymbol{\sigma}']\right]\right|\right| \leq 2$$

Proof. Given two neighboring update operation sequences $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$, it is clear that the resulting counter mapped sequences, for any $i \in [n]$, will differ in at most 1 position (and that position will be same as the position at which $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ differ). This is because one update operation can map to one and only one operation in the counter mapped sequences. Therefore, we have that for any $i \in [n]$,

$$\left|\left|\mathsf{Map}_2^i(\boldsymbol{\sigma}) - \mathsf{Map}_2^i(\boldsymbol{\sigma}')\right|\right| \leq 1.$$

Based on the result of Lemma D.1, we can then conclude our proof. □

Since an update $u \in \mathbb{U}$ can be mapped to $|\mathsf{Map}_1(u)|$ counters, the overall privacy is degraded by a multiplicative $\max_{u \in \mathbb{U}} |\mathsf{Map}_1(u)|$ factor.

Given the results of the lemma above, we now state our main theorem.

Theorem 6.4. *If Δ_{ctr} is ϵ-differentially-private, then HPX is $2\epsilon \cdot \max_{u \in \mathbb{U}}(|\mathsf{Map}_1(u)|)$-differentially private as per Definition 4.7.*

Proof. In HPX, every counter is implemented using Δ_{ctr} which is an ϵ-differentially private counter on neighboring sequences in $\{-1, 0, 1\}^\lambda$. However, if the sequences are not 1-distance (neighbors) apart but k-distance apart, Δ_{ctr} provides $k\epsilon$-differentially private counters. We already showed in Lemma D.2 that neighboring update operation sequences indeuce operation sequences on individual counters that are 2-distance apart. However, an update can map to multiple counters. In particular, an update $u \in \mathbb{U}$ can map to $|\mathsf{Map}_1(u)|$ counters, implying that we can have a maximum of $\max_{u \in \mathbb{U}}(|\mathsf{Map}_1(u)|$ sequences that are 2-distance apart. Leveraging the union bound we therefore prove our theorem. □