Zachary Espiritu*, Evangelia Anna Markatou, and Roberto Tamassia

# Time- and Space-Efficient Aggregate Range Queries over Encrypted Databases

**Abstract:** We present **ARQ**, a systematic framework for creating cryptographic schemes that handle range aggregate queries (sum, minimum, median, and mode) over encrypted datasets. Our framework does not rely on trusted hardware or specialized cryptographic primitives such as property-preserving or homomorphic encryption. Instead, **ARQ** unifies structures from the plaintext data management community with existing *structured encryption* primitives. We prove how such combinations yield efficient (and secure) constructions in the encrypted setting. We also propose a series of *domain reduction* techniques that can improve the space efficiency of our schemes against sparse datasets at the cost of small leakage. As part of this work, we designed and implemented a new, open-source, encrypted search library called Arca and implemented the **ARQ** framework using this library in order to evaluate **ARQ**'s practicality. Our experiments on real-world datasets demonstrate the efficiency of the schemes derived from **ARQ** in comparison to prior work.

**Keywords:** structured encryption, encrypted search algorithms, aggregate queries, range queries

DOI 10.2478/popets-2022-0095
Received 2022-02-28; revised 2022-06-15; accepted 2022-06-16.

## 1 Introduction

*Structured encryption* (STE) [15, 17] refers to a class of *encrypted search algorithms* (ESA) [29, 42] that allow a client to outsource (and query) an encrypted version of their data to a semi-honest server. Recent research in STE has focused on developing specialized structures that efficiently handle more expressive types of queries, such as *range queries*. For example, the *BlindSeer* construction from [57], the garbled-circuit-based construction from [10], and the range tree constructions from [19, 20, 25, 27, 49, 65, 68] are all examples of *encrypted range query structures*. In these works, range queries are effectively a simple *filter*, applied to a single database attribute, that returns a *set* of matching records.

*Aggregate range queries.* Despite the focus on encrypted range structures in recent years, real-world applications often do not directly require the records in the queried range, but rather the result of an *aggregate function* folded over a second record attribute in the queried range. Consider, for instance, the following SQL query over an employees table, which asks for the median salary of all employees between the ages 30 and 40:

```
SELECT MEDIAN(salary) FROM employees
        WHERE age BETWEEN 30 AND 40;
```

We call the attribute the filter is composed over the *filter attribute* and the attribute the aggregation function is composed over the *aggregate attribute*. In this example, the filter attribute is age and the aggregate attribute is salary. To answer this query, the server only needs to output a single numerical value (the median salary). Integrating aggregate functions directly into queries in this way can minimize bandwidth as the server only needs to send a constant-size value to the client.

*Performance gap for encrypted aggregates.* Using existing record-reporting STE schemes to answer aggregate queries over encrypted data incurs a significant overhead. In many STE structures, the server cannot compute the aggregate server-side and must return the entire set of records satisfying the filter. This incurs linear-size bandwidth and requires the client to spend linear resources to decrypt each of the records before computing the aggregate. Conversely, one may precompute answers to all possible range queries in an encrypted dictionary. This achieves constant-size bandwidth and query time, but prohibitively requires quadratic storage.

Many approaches have been proposed to bridge this performance gap. *Fully homomorphic encryption* (FHE) [30], for example, can allow the server to compute portions of the aggregate before responding to the client. Unfortunately, state-of-the-art FHE schemes have high performance costs and are prohibitive for real-world

***Corresponding Author: Zachary Espiritu:** Brown University, E-mail: zesp@brown.edu
**Evangelia Anna Markatou:** Brown University, E-mail: markatou@brown.edu
**Roberto Tamassia:** Brown University, E-mail: roberto@tamassia.net

applications. As an alternative, *additively homomorphic encryption* (AHE) [56] imposes more acceptable performance costs and allows the server to sum encrypted values prior to sending them to a client. However, AHE does not allow for non-additive aggregates (e.g. min/max) and still requires the server to spend computation time to add AHE ciphertexts.

Alternatively, some encrypted range structures can be augmented with precomputed *sub-aggregates* within the structure. Queries may then be answered by returning a poly-logarithmic set of sub-aggregates which the user processes to recover a single aggregate. These schemes can be somewhat more practical for aggregate queries; however, they may incur higher than necessary storage overhead, especially when the dataset is *sparse*.

Aside from performance concerns, "naive" STE approaches for handling aggregates may incur more leakage than necessary. While the ultimate goal of the end user is to compute a single aggregate value, using standard record-reporting STE schemes as a building block for aggregates often results in *search pattern* leakage (whether the same query is made multiple times) and *access pattern* leakage (whether the same encrypted record is used to respond to multiple queries) that may eventually lead to reconstruction attacks [7, 42]. These performance and security limitations highlight the need for new constructions for encrypted range aggregates.

*Contributions*. We take an *interdisciplinary* approach and demonstrate how *simple* combinations of data structures previously developed in the plaintext data management community with STE primitives can yield efficient (and secure) constructions in the encrypted setting. More generally, our work aims to showcase how cryptographers can leverage prior advances in databases and data structures to develop secure constructions for ESAs. Our contributions are summarized as follows:

- We introduce **ARQ**, a framework for building encrypted aggregate range query indexes which provably captures the leakage of our schemes. We identify a *data-oblivious* (DO) security property which guarantees that data reconstruction attacks are impossible against DO schemes, provided that query and data distributions are independent. (§ 3, § 4)
- Using **ARQ**, we propose novel schemes for encrypted range minimum, approximate mode, and approximate median. Denoting with $m$ the domain size (number of possible values of the filter attribute) and with $n$ the number of records, our minimum scheme improves the previous best $O(m + n \log n)$ storage of Demertzis et al.'s 2-round protocol [20] to an $O(m)$ storage 1-round protocol by prohibiting some *small* queries. To our knowledge, our approximate mode and median schemes are the first in the STE literature and allow for constant time and size queries. (§ 5, § 6, § 7)
- We propose *domain reductions* which can be applied to any **ARQ** scheme to optimize its storage overhead over *sparse* databases in exchange for small performance and leakage. Our reductions improve the performance of the sum and minimum constructions by Demertzis et al. [20] over sparse databases. (§ 8)
- We implement Arca, a new Python library for rapid prototyping of ESAs using STE schemes. Using Arca, we provide a reference implementation of the **ARQ** framework in Python. Using this implementation, we conduct an empirical evaluation of our proposed schemes against real-world datasets. (§ 9)

## 2 Related Work

*Encrypted range aggregate queries in STE*. Demertzis et al. [20] introduced the first study of STE structures for encrypted range sum and minimum queries with constant client-side storage, bandwidth, and computation time. They proposed an encrypted range sum scheme based upon the classic *prefix sums* technique [9]; it requires $O(m)$ storage, where $m$ is the domain size (number of possible values of the filter attribute). They also proposed two schemes for encrypted range minimums based upon the *sparse table* technique by Bender et al. [5]. The first scheme incurs $O(m \log m)$ storage in the size of the domain. The second scheme incurs a smaller $O(m+n \log n)$ storage requirement, where $n$ is the number of records, at the expense of an additional round trip. However, even with the storage optimization of the second scheme, the $O(m)$ factor in the asymptotics still may be prohibitive for sparse databases.

*Encrypted aggregates via other techniques*. Many works have been proposed to support aggregate functions over encrypted databases via various forms of homomorphic encryption and property preserving encryption. A notable example is CryptDB, the first system to support standard SQL operations over encrypted data [60]. CryptDB uses specialized encryption schemes such as *order-preserving encryption* (OPE) [11] and *additively homomorphic encryption* (AHE) [56] to support certain SQL operations, including aggregates. Many similar works use AHE to support aggregates [3, 34, 40, 43, 62].

**Table 1.** Our contributions compared to Demertzis et al. (DPPDGP) [20]. Storage, query time, and communication bandwidth are asymptotic (big-$O$), where $m$ is the domain size, $n$ is the number of records, and $0 < \alpha < 1$ is a tunable parameter. Storage and query time are given separately for the server and the client. We further characterize tradeoffs between the schemes by indicating whether a scheme is *data-oblivious* ("DO") (Definition 3.3) or optimizes storage by taking advantage of sparsity in datasets ("Sparse").

| | Schemes | Server Complexity | | Communication | | Client Complexity | | Tradeoff | |
|---|---|---|---|---|---|---|---|---|---|
| | | Storage | Query | Bandwidth | Rounds | Storage | Query | DO | Sparse |
| Sum | DPPDGP-Sum [20] | $m$ | 1 | 1 | 1 | 1 | 1 | ● | |
| | Sum+DomainBucket | $m^\alpha + n$ | 1 | $n$ | 1 | $n$ | 1 | | ● |
| | Sum+DataBucket | $m^\alpha + n$ | 1 | $\frac{n}{m^\alpha}$ | 1 | $m^\alpha$ | $\frac{n}{m^\alpha}$ | | ● |
| Minimum | DPPDGP-Min1 [20] | $m \log m$ | 1 | 1 | 1 | 1 | 1 | ● | |
| | DPPDGP-Min2 [20] | $m + n \log n$ | 1 | 1 | 2 | 1 | 1 | | ● |
| | LinearMin | $m$ | 1 | 1 | 1 | 1 | 1 | ● | |
| Mode | 1/2-ApproxMode | $m \log m$ | 1 | 1 | 1 | 1 | $d$ | ● | |
| | 1/3-ApproxMode | $m \log \log m$ | 1 | 1 | 1 | 1 | 1 | ● | |
| Median | $\alpha$-ApproxMedian | $\frac{m}{1-\alpha}$ | 1 | 1 | 1 | 1 | 1 | ● | |

However, AHE does not support non-additive aggregate queries, so many of these works do not support operations such as minimum, median, and mode. Additionally, while OPE may be used for rank-based range aggregates (e.g., median), many works have demonstrated that OPE leaks enough information to allow for powerful, practical data-recovery attacks [6, 21, 38, 53].

Other works rely on *trusted hardware*. For instance, Cipherbase supports aggregations using custom, trusted field-programmable gate arrays (FPGAs) [3] that are used to compute sum aggregates. It sends encrypted values to the FPGAs to be decrypted and summed. Then, the result is reencrypted before releasing it to the untrusted environment. Other works use *trusted execution environments* such as Intel SGX. However, the security guarantees of such hardware are complex and often are memory-limited. We do not rely on specialized hardware or hardware security assumptions in this work.

*Leakage-abuse attacks on encrypted range structures.* An important property of a structured encryption scheme is its *leakage*, or what information is revealed by the scheme's operations. Islam et al. introduced the first study of STE leakage-abuse attacks [41]; they showed how to perform *query-recovery* attacks by exploiting *access pattern* leakage. Follow-up work such as Cash et al. [13] and Zhang et al. [67] exploited access pattern leakage to launch query-recovery attacks with similar impact under different assumptions. Recent work has also shown how to combine access *and* search pattern leakage to perform query-recovery attacks [55].

The increased expressiveness of range queries may allow for more severe *data-recovery* attacks. Many works have demonstrated powerful one-dimensional attacks under varying assumptions (see. e.g., [36, 37, 39, 45–48]), and a recent line of work has yielded reconstruction attacks in higher dimensions [26, 27, 50]. The impact of these kinds of attacks can vary when pitted against real-world datasets and query distributions [42].

# 3 Preliminaries

*Notation.* $\{0,1\}^\ell$ denotes the set of all binary strings of length $\ell$. $\{0,1\}^*$ denotes the set of all finite binary strings. $\perp$ represents the empty string. $x \leftarrow \mathcal{A}$ represents the output $x$ of procedure $\mathcal{A}$. Given a set $S$, the cardinality of $S$ is denoted $\#S$. The set of numbers $\{0, 1, \ldots, x-1\}$ is denoted $[x]$.

*Basic structures.* Our protocols use several basic data structures whose syntax we define here. A *dictionary* $\mathsf{DX}$ of size $s$ is a collection of $s$ key-value pairs. $v_i := \mathsf{DX}[\ell_i]$ denotes the retrieval of the value $v_i$ associated with the label $\ell_i$. $\mathsf{DX}[\ell_i] := v$ denotes the assignment of value $v_i$ to label $\ell_i$. A *multimap* $\mathsf{MM}$ of size $s$ is a collection of $s$ key-tuple pairs. $\mathbf{t}_i := \mathsf{MM}[\ell_i]$ denotes the retrieval of the tuple $\mathbf{t}_i$ associated with the label $\ell_i$. $\mathsf{MM}[\ell_i] := \mathbf{t}_i$ denotes the assignment of tuple $\mathbf{t}_i$ to label $\ell_i$. The length of each tuple $\mathbf{t}_i$ may vary within the multimap.

Given a data structure $\mathsf{DS}$, we refer to the act of retrieving the value associated with a label in $\mathsf{DS}$ as *querying* $\mathsf{DS}$. We refer to the set of labels that can be used to query $\mathsf{DS}$ as the *query space* of $\mathsf{DS}$ and the set of possible outputs as the *response space* of $\mathsf{DS}$. We write $\mathsf{DS} : \mathbf{Q} \rightarrow \mathbf{R}$ to denote that $\mathbf{Q}$ is $\mathsf{DS}$'s query space and $\mathbf{R}$ is $\mathsf{DS}$'s response space. We denote the number of key-value pairs in $\mathsf{DS}$, or $\mathsf{DS}$'s *size*, as $|\mathsf{DS}|$.

*Tables.* A *table* $\mathsf{DB}$ is a two-dimensional array where each row is a *record* and each column is an *attribute*. We assume that each attribute has a finite *domain* of possible values. Every record $\mathbf{r} \in \mathsf{DB}$ is a *tuple* indexed by each of DB's attributes. In this work, we primarily consider one-dimensional queries, so we assume that every record has one filter attribute and one aggregate attribute for simplicity. Given a domain value $x_i$, we use $\mathsf{DB}(x_i)$ to denote the set of records with filter attribute value $x_i$. We also use $\mathsf{DB}^{\leftarrow}(x_i)$ to denote the record with filter attribute value closest or equal to, but not greater than, $x_i$; similarly, $\mathsf{DB}^{\rightarrow}(x_i)$ denotes the record with filter attribute value closest to, but not less than, $x_i$.

Throughout the paper, we use $n$ to refer to the number of records in DB and $m$ to refer to the domain size of DB's filter attribute.

*Structured encryption.* A *structured encryption* (STE) scheme encrypts a data structure $\mathsf{DS}$ so that a *client* can outsource it and privately query it using a secret key $K$. We exclusively use *response-hiding* schemes where query responses are not revealed to the server.

**Definition 3.1** (Response-hiding STE [15]). *A STE scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ *consists of four polynomial-time algorithms that work as follows:*

- $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$ *is a probablistic algorithm run by the client. It takes as input a security parameter* $1^k$ *and a plaintext data structure* $\mathsf{DS}$*. It then outputs a key* $K$ *and an encrypted structure* $\mathsf{EDS}$*.*
- $\mathsf{tk} \leftarrow \mathsf{Token}(K, Q)$ *is a deterministic algorithm run by the client when it issues a query. It takes as input a key* $K$ *and a query* $Q$ *and outputs a token* $\mathsf{tk}$*.*
- $\mathsf{ct} \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$ *is a deterministic algorithm run by the server to respond to queries. It takes as input the encrypted structure* $\mathsf{EDS}$ *and a query token* $\mathsf{tk}$ *and outputs a response* $\mathsf{ct}$ *(which may be* $\perp$*).*
- $R \leftarrow \mathsf{Resolve}(K, \mathrm{ct})$ *is a deterministic algorithm that takes as input the secret key* $K$ *and a ciphertext* $\mathrm{ct}$ *and outputs a plaintext response* $R$*.*

*We say* $\Sigma$ *is correct if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \rightarrow \mathbf{R}$*, for all* $\mathsf{poly}(k)$*-size sequences of queries* $Q_1, \ldots, Q_s$ *where* $Q_i \in \mathbf{Q}$*, for all* $\mathsf{tk}_i$ *output by* $\mathsf{Token}(K, Q_i)$*,* $\mathsf{Resolve}(K, \mathsf{Query}(\mathsf{EDS}, \mathsf{tk}_i)) = \mathsf{DS}[Q_i]$ *with all but negligible probability.*

Definition 3.1 applies to *non-interactive* STE schemes. Some STE protocols are *interactive*, where queries client involve more than one round of communication.

**Definition 3.2** (Interactive STE). *A interactive STE scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Query})$ *consists of two polynomial-time algorithms where* $\mathsf{Setup}$ *is the same as it was in Definition 3.1 and* $\mathsf{Query}$ *is as follows:*

- $(R, \perp) \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}((K, Q), \mathsf{EDS})$ *is a two party protocol algorithm run between the client and the server. It takes as input from the client a key* $K$ *and a query* $Q$ *and as input from the server an encrypted structure. It outputs a plaintext result* $R$ *to the client and nothing to the server.*

*We say* $\Sigma$ *is correct if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \rightarrow \mathbf{R}$*, for all* $\mathsf{poly}(k)$*-size sequences of queries* $Q_1, \ldots, Q_m$ *where* $Q_i \in \mathbf{Q}$*,* $\mathsf{Query}_{\mathbf{C},\mathbf{S}}((K, Q_i), \mathsf{EDS}) = \mathsf{DS}[Q_i]$ *with all but negligible probability.*

*Security definitions.* To prove security of an STE protocol $\Sigma$, we define *leakage functions* that capture what information is revealed by the different operations of $\Sigma$: $\mathcal{L}_{\mathsf{S}}$, the *setup leakage*, or what is leaked by the $\mathsf{Setup}$ operation, and $\mathcal{L}_{\mathsf{Q}}$, the *query leakage*, or what is leaked by the $\mathsf{Query}$ algorithm. We then prove that a semi-honest, adaptive adversary can only distinguish between two *experiments* with *negligible probability*: the *real* world, in which the actual STE protocol is used against the adversary, and the *ideal* world, which attempts to *simulate* the real world only based on the leakage $\mathcal{L}_{\mathsf{S}}$ and $\mathcal{L}_{\mathsf{Q}}$. We use the standard definitions for adaptive semantic security from [15], which we recap in Appendix A.

## 3.1 Data-Oblivious STE Schemes

In our work, we find that the reduced expressiveness of aggregate structures (they do not return the records in the queried range) means that aggregate structures do not necessarily suffer from the same types of leakage as SSE structures (i.e., volume pattern, search pattern, and access pattern to individual records). As such, we are interested in precisely capturing when such a scheme avoids these kinds of *data-dependent* leakage patterns, as this would imply that the scheme is secure against all previously known *data-reconstruction* attacks.

We call this property *data-obliviousness* (DO). This property is similar to the standard definition used in the *data-oblivious algorithm* literature (e.g., [8, 32]). However, our definition differs in that it makes no statement about the information leaked by different query transcripts. At a high-level, a STE scheme is DO if its setup leakage $\mathcal{L}_{\mathsf{S}}$ reveals nothing about the plaintext structure other than (potentially) the size of the structure *and* if its query leakage $\mathcal{L}_{\mathsf{Q}}$ reveals nothing other than (potentially) the size of the structure and information about the queries themselves.

**Definition 3.3** (Data-oblivious STE scheme). *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ *be an adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-semantically secure, response-hiding, structured encryption scheme for the data structure* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$. *We say that* $\Sigma$ *is* data-oblivious (DO) *if there exists functions* $f$ *and* $g$ *such that* $\mathcal{L}_{\mathsf{S}}(\mathsf{DS}) = f(|\mathsf{DS}|)$ *and* $\mathcal{L}_{\mathsf{Q}}(\mathsf{DS}, Q) = g(|\mathsf{DS}|, Q)$.

The DO definition for interactive STE schemes is identical to Definition 3.3 except $\Sigma = (\mathsf{Setup}, \mathsf{Query})$. The DO property guarantees that data reconstruction attacks are impossible if the queries to $\Sigma$ are independent of the underlying data distribution (a standard assumption in the reconstruction attacks literature). This is because the information leaked by $\Sigma$ is solely a function of the size of the encrypted data structure and the queries.

# 4 ARQ: A General Framework

A primary goal of this work is to identify features of existing plaintext aggregate range query schemes that make them suitable for the encrypted setting. To do this, we introduce a general syntax for *plaintext* aggregate range query schemes. We then show how to derive a framework for building provably-secure, encrypted aggregate range supporting schemes from this syntax. Our framework allows us to provably characterize the leakage of any aggregate range scheme that falls within our syntax. Finally, we introduce a *data independence* definition that captures a set of plaintext aggregate range query schemes that may produce a provably DO scheme.

Before presenting our framework, we first define our syntax for *plaintext* aggregate range query schemes. All of the plaintext schemes we consider in this work (and those from Demertzis et al. [20]) fit into this syntax.

**Definition 4.1** (Aggregate range query scheme). *An* aggregate range query scheme $\Pi = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *for a database* $\mathsf{DB}$ *consists of three polynomial-time, deterministic algorithms that work as follows:*

- $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$ *takes as input a database* $\mathsf{DB}$ *and outputs an index structure* $\mathsf{DS} : \mathbf{U} \to \mathbf{S}$.
- $U \leftarrow \mathbb{Q}(m, Q)$ *takes as input the domain size* $m$ *and a query* $Q$. *It outputs a initial set of subqueries* $U \subseteq \mathbf{U}$ *to be issued to* $\mathsf{DS}$.
- $(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)$ *aggregates a set of responses from* $\mathsf{DS}$. *It takes as input prior state* $\mathrm{st}$ *(which may be* $\perp$*), the domain size* $m$*, the initial query* $Q$*, and a set of responses* $S \subseteq \mathbf{S}$. *It then outputs new state* $\mathrm{st}'$*, an aggregate* $R$ *(which may be* $\perp$*), and a set of additional subqueries to be issued* $U$ *(which may be* $\perp$*).*

We now describe $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$, our *framework for encrypted aggregate range query schemes.* $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ is a (possibly) interactive STE scheme which is parameterized by an aggregate range query scheme $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ and a response-hiding STE scheme $\Sigma$ on data structure $\mathbb{S}(\mathsf{DB})$. The full **ARQ** framework is defined in Figure 1. (For brevity, we refer to $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ as **ARQ** in the paper when the parameters are implied from context.) In **ARQ**, the client converts their database $\mathsf{DB}$ into a aggregate query index $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$. Then, $\Sigma.\mathsf{Setup}$ is used to encrypt $\mathsf{DS}$. When the client performs a range query, they use $\mathbb{Q}$ and $\Sigma.\mathsf{Token}$ to determine which search tokens to send to the server. Finally, the client decrypts the responses and passes the responses to $\mathbb{R}$. $\mathbb{R}$ may output another set of subqueries to issue to the server; otherwise, $\mathbb{R}$ outputs the final aggregate to the client.

While the general form of **ARQ** is written in the interactive STE syntax in order to support interactive plaintext schemes, we emphasize that schemes derived from **ARQ** do not necessarily require multiple rounds of communication—in fact, most any of the **ARQ** schemes that we derive in this work only require a single round of communication between the client and server.

**Theorem 4.2.** *If* $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *is an aggregate range query scheme, and* $\Sigma$ *is an adaptively* $(\mathcal{L}_{\mathsf{S}}^{\Sigma}, \mathcal{L}_{\mathsf{Q}}^{\Sigma})$*-secure, response-hiding STE scheme on data structure* $\mathbb{S}(\mathsf{DB})$*, then* $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ *is adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$*-secure, where*

$$\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = \mathcal{L}_{\mathsf{S}}^{\Sigma}(\mathsf{DS}), \qquad \text{and}$$

$$\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = \left(\mathcal{L}_{\mathsf{Q}}^{\Sigma}(\mathsf{DS}, u)\right)_{u \in \Lambda}.$$

*Here,* $\Lambda$ *is the union of* $U_i$*'s, where* $U_i$ *is the instantiation of* $U$ *on the* $i^{th}$ *loop of* $\mathsf{Query}_{\mathbf{C},\mathbf{S}}$.

*Proof.* Let $\mathcal{S}^{\Sigma}$ be the simulator guaranteed to exist by the adaptive security of $\Sigma$ and consider the **ARQ** simulator $\mathcal{S}$ that works as follows. Given $\mathcal{L}_{\mathsf{S}}(\mathsf{DB})$, $\mathcal{S}$ simulates the encrypted index $\mathsf{EDS}$ by computing $\mathsf{EDS} \leftarrow \mathcal{S}^{\Sigma}(\mathcal{L}_{\mathsf{S}}(\mathsf{DB}))$. Given $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q)$, for each $\mathcal{L}_{\mathsf{Q}}^{\Sigma}(\mathsf{DS}, u)$ in $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q)$, $\mathcal{S}$ outputs $\mathcal{S}^{\Sigma}(\mathcal{L}_{\mathsf{Q}}^{\Sigma}(\mathsf{DS}, u))$.

We now show that, for all PPT adversaries $\mathcal{A}$,

$$\underset{break}{\Pr} = |\Pr[\mathbf{Real}_{\mathbf{ARQ},\mathcal{A}} = 1] - \Pr[\mathbf{Ideal}_{\mathbf{ARQ},\mathcal{A},\mathcal{S}} = 1]|$$

Let $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ be a response-hiding STE scheme, $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ be an aggregate range query scheme, $\mathbf{S}$ be a server, and $\mathbf{C}$ be a client. Consider the interactive encrypted aggregate range query scheme $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}} = (\mathsf{Setup}, \mathsf{Query})$ defined as follows:

```
– $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DB})$:
    1. compute $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$;
    2. compute $(K, \mathsf{EDS}) \leftarrow \Sigma.\mathsf{Setup}(1^k, \mathsf{DS})$;
    3. output $(K, \mathsf{EDS})$;
– $R \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}(K, Q)$:
    1. $\mathbf{C}$ sets $\mathrm{st} \leftarrow \perp$ and $R \leftarrow \perp$;
    2. $\mathbf{C}$ computes $U \leftarrow \mathbb{Q}(m, Q)$;
    3. while $U \neq \perp$,
        (a) for all $u_i \in U$,
                i. $\mathbf{C}$ computes $\mathsf{stk}_i \leftarrow \Sigma.\mathsf{Token}(K, u_i)$;
                ii. $\mathbf{C}$ sends $\mathsf{stk}_i$ to server;
        (b) for all $\mathsf{stk}_i$,
                i. $\mathbf{S}$ computes $\mathrm{ct}_i \leftarrow \Sigma.\mathsf{Query}(\mathsf{stk}_i, \mathsf{EDS})$;
                ii. $\mathbf{S}$ sends $\mathrm{ct}_i$ to client;
        (c) initialize set $S$;
        (d) for all $\mathrm{ct}_i$,
                i. $\mathbf{C}$ computes $s_i \leftarrow \Sigma.\mathsf{Resolve}(K, \mathrm{ct}_i)$;
                ii. $\mathbf{C}$ adds $s_i$ to $S$;
        (e) $\mathbf{C}$ computes and sets $(\mathrm{st}, R, U) \leftarrow \mathbb{R}(\mathrm{st}, Q, S)$;
    4. $\mathbf{C}$ outputs $R$;
```

**Fig. 1.** The $\mathbf{ARQ}$ framework.

is negligible. The only difference between $\mathbf{Real}_{\mathbf{ARQ},\mathcal{A}}$ and $\mathbf{Ideal}_{\mathbf{ARQ},\mathcal{A},\mathcal{S}}$ is that all applications of $\Sigma.\mathsf{Setup}$ and $\Sigma.\mathsf{Token}$ have been replaced with invocations of $\Sigma$'s simulator $\mathcal{S}^{\Sigma}$. Thus, in order for the adaptive semantic security of $\Sigma$ to hold, $\Pr_{break}$ must be negligible. □

## 4.1 Data Independence

Motivated by the security guarantees of Definition 3.3, we now identify properties of aggregate range query schemes and STE schemes that result in an instantiation of $\mathbf{ARQ}$ that satisfies the DO requirements. We first introduce a notion of *data independence* over a plaintext aggregate range query scheme. Intuitively, if $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ is data independent, then the size of all responses to queries is identical given input databases $\mathsf{DB}$ of the same size and the subqueries generated by $\mathbb{Q}$ and $\mathbb{R}$ do not depend on the original table $\mathsf{DB}$ passed to $\mathbb{S}$.

**Definition 4.3** (Data independence). *Let $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ be a plaintext aggregate range query scheme, $(\mathsf{DB}_0, \mathsf{DB}_1)$ be a pair of tables of size $m = |\mathsf{DB}_0| = |\mathsf{DB}_1|$, and $T = Q_1, \ldots, Q_s$ be a query transcript. Then, for each $Q \in T$, let $U \leftarrow \mathbb{Q}(m, Q)$. Then, for $i \in \{0, 1\}$, let*

$$\begin{aligned} \mathsf{DS}_i &\leftarrow \mathbb{S}(\mathsf{DB}_i), \\ S_{i:0} &\leftarrow \{\mathsf{DS}_i[u] \mid u \in U\}, \quad \text{and} \\ (\mathrm{st}_{i:0}, R_{i:0}, U_{i:0}) &\leftarrow \mathbb{R}(\perp, m, Q_i, S_{i:0}). \end{aligned}$$

*Then, for $j > 0$,*

$$(\mathrm{st}_{i:j}, R_{i:j}, U_{i:j}) \leftarrow \begin{cases} \mathbb{R}(\mathrm{st}_{i:j-1}, m, Q_i, S_{i:j-1}) & \text{if } U_{i:j-1} \neq \perp \\ (\perp, R_{i:j-1}, \perp) & \text{if } U_{i:j-1} = \perp \end{cases}$$

*We say that $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ is a* data independent *aggregate range query scheme if all of these conditions are true:*

(C1) $|\mathsf{DS}_0| = |\mathsf{DS}_1|$ *and*

(C2) $U_{0:j} = U_{1:j}$ *for all $j > 0$.*

*Implications of data independence.* We first outline one interesting implication of the data independence definition that implies that any data independent scheme requires at most one round of communication.

**Corollary 4.4** (sketch). *Let $\Pi$ be an interactive plaintext aggregate range query scheme. If $\Pi$ satisfies data independence, $\Pi$ may be converted into a non-interactive scheme with identical functionality.*

The formalization of Corollary 4.4 is in Appendix B. The contrapositive of Corollary 4.4 provides an important insight—plaintext aggregate range query schemes that cannot be converted into a equivalent, non-interactive scheme will not satisfy data independence. As we describe later, this observation allows us to prune our search for plaintext schemes that can produce DO-satisfying, $\mathbf{ARQ}$-based encrypted schemes.

From a security perspective, our notion of *data independence* captures a class of plaintext aggregate range query schemes that can be combined with the $\mathbf{ARQ}$ framework and a suitable choice of $\Sigma$ to produce a encrypted aggregate range scheme that satisfies our DO security property. Specifically, the following theorem states that, if $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ is data independent and $\Sigma$ is a DO, response-hiding STE encryption scheme over the data structure output by $\mathbb{S}$, then $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ is DO.

**Theorem 4.5.** *Let $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ be a data independent aggregate range query scheme where $\mathbb{S}$ outputs data structure $\mathsf{DS}$ with query space $\mathbf{U}$ and response space $\mathbf{S}$ given some input database $\mathsf{DB}$. Also, let $\Sigma$ be a DO, response-hiding STE scheme for data structure $\mathsf{DS} : \mathbf{U} \to \mathbf{S}$. Then, the interactive STE scheme $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ is DO.*

*Proof.* We observe that the DO property of $\Sigma$ implies there exist functions $f$ and $g$ such that $\mathcal{L}_{\mathsf{S}}^{\Sigma}(\mathsf{DS}) = f(|\mathsf{DS}|)$, $\mathcal{L}_{\mathsf{Q}}^{\Sigma}(\mathsf{DS}, Q) = g(|\mathsf{DS}|, Q)$, and $\Sigma$ is $(\mathcal{L}_{\mathsf{S}}^{\Sigma}, \mathcal{L}_{\mathsf{Q}}^{\Sigma})$-secure. We now show that there exist functions $h$ and $j$ such that $\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = h(|\mathsf{DB}|)$, $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = j(|\mathsf{DB}|, Q)$, and $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ is adaptively $(\mathcal{L}_{\mathsf{S}}^{\Sigma}, \mathcal{L}_{\mathsf{Q}}^{\Sigma})$-secure.

- $\mathcal{L}_{\mathsf{S}}$: By (C1) of Definition 4.3, we know that every database of the same size $|\mathsf{DB}|$ results in an aggregate index structure of the same size $|\mathsf{DS}|$. Thus, there exists some function $f'$ such that $|\mathsf{DS}| = f'(|\mathsf{DB}|)$. By Theorem 4.2, $\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = \mathcal{L}_{\mathsf{S}}^{\Sigma}(\mathsf{DS}) = f(|\mathsf{DS}|) = f(f'(|\mathsf{DB}|))$. Thus, defining $h = f \circ f'$ gives us $\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = h(|\mathsf{DB}|)$ as desired.
- $\mathcal{L}_{\mathsf{Q}}$: By definition of $\Sigma$, $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = (g(|\mathsf{DS}|, Q))_{q \in \mathbf{U}}$. We just showed that there exists some function $f'$ such that $|\mathsf{DS}| = f'(|\mathsf{DB}|)$, so we know that $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = (g(f'(|\mathsf{DB}|), Q))_{q \in \mathbf{U}}$. Also, by (C2) of Definition 4.3, we know that $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ always generates the same subqueries for the aggregate index $\mathsf{DS}$ for a given query $Q$ regardless of the underlying contents of $\mathsf{DB}$; in other words, the $\Lambda$ term of the leakage is a function of $Q$ and nothing else. Thus, we know that there must exist a function $j$ such that $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = (g(f'(|\mathsf{DB}|), Q))_{q \in \Lambda} = j(|\mathsf{DB}|, Q)$. □

We now present several plaintext aggregate range query schemes using our syntax from Definition 4.1, then use the **ARQ** framework to instantiate encrypted variants of those schemes. In Sections 5, 6, and 7, we also prove the data independence of our chosen plaintext aggregate range query schemes, which shows that the resulting **ARQ** schemes are DO when using a standard STE scheme, such as $\Pi_{\text{bas}}$ from [14] (Theorem 4.5).

In each section, we also perform a complexity analysis on concrete instantiations of **ARQ**. Since **ARQ** requires a concrete $\Sigma$, our analyses assume that $\Sigma$ requires storage $O(|\mathsf{DS}|)$ and each query requires constant time and space in the size of $\mathsf{DS}$. An example of such a $\Sigma$ that satisfies these assumptions and the leakage profile constraints in Theorem 4.5 is $\Pi_{\text{bas}}$ from [14]. While we use $\Pi_{\text{bas}}$ for our analysis, we emphasize that different choices of $\Sigma$ may be selected, which may result in different complexity and leakage tradeoffs.

# 5 Range Minimum Query

We now consider the *range minimum query* problem. Solutions to the range minimum query problem can be used to answer other types of range queries in exchange for a constant factor increase in time and space. We provide examples of such *query type transformation* techniques for encrypted databases in Appendix C in lieu of considering specialized structures for those types.

**Definition 5.1** (Range minimum query)**.** *Given an array $A$ of $n$ numbers and indices $i$ and $j$ where $0 \le i \le j < n$, the* range minimum query *returns the smallest element of $A$ in the range $[i, j]$.*

*Previous results.* Demertzis et al. [20] proposed two approaches to the RMQ problem which we refer to as DPPDGP-Min1 and DPPDGP-Min2. DPPDGP-Min1 is built directly on top of the *sparse table* (ST) technique by Bender et al. [5]. Starting from every element in $A$, ST precomputes the answer for all queries whose range length is a power of 2 and stores it in an two-dimensional array $M$. This produces $O(\log m)$ answers for each of the $m$ elements of $A$. Thus, the total space of $M$ is $O(m \log m)$. To answer a query for an interval $[\ell, r]$, ST accesses $M$ for the precomputed answers for the two overlapping intervals that exactly cover $[\ell, r]$. It then returns the minimum of those two answers.

DPPDGP-Min1 encrypts the array generated by ST to achieve $O(m \log m)$ space with constant-size and time queries. DPPDGP-Min2 is similar to DPPDGP-Min1, but reduces the overhead on sparse databases to $O(m + n \log n)$ in exchange for an additional round of communication. It does this by first accessing an additional encrypted index that maps query domain values to the identifier of the record nearest to that domain value. This allows the ST to be built over the record identifiers, resulting in smaller storage when $n < m$.

It can be easily shown that DPPDGP-Min1 is DO. However, DPPDGP-Min2 is not DO—the scheme used by DPPDGP-Min2 is interactive (and cannot be easily changed into a non-interactive scheme) and is thus not data independent as implied by Corollary 4.4.

*Existing plaintext structures.* Several $O(m)$-space solutions to for range minimums exist in the data management literature. However, achieving this storage bound appears to necessitate the use of various compaction techniques that make them unsuitable for our security goals. For example, some schemes [18, 22] use *bit-packing* techniques that condense $O(m \log m)$ bits to $O(m)$ words through the combination of different components of the structure into the same word. While these structures, in theory, could be implemented in the STE setting, practical symmetric encryption algorithms' use of padding would prevent the use of bit-packing and thus $O(m \log m)$ space would be required.

Similarly, other schemes [4, 5, 28] use a related *lookup table* technique. At a high-level, this class of techniques saves space by storing the answers to all possible queries in a compact lookup table. Then, queries to the structure return a reference to a part of the lookup table, which must be separately queried to retrieve the actual answer. These kinds of structures are also unsuitable for our DO security goals for the following reasons: first, Corollary 4.4 implies the multi-round nature of the lookup table technique makes it impossible for the plaintext scheme to be data independent, and thus the resulting **ARQ** scheme will not be DO. More specifically, if two queries to the structure require the user to query the same component of the lookup table, then the access pattern leakage on the lookup table reveals that both queries had the same answer. Thus, the access pattern leakage on the lookup table can change when the contents of the underlying database changes, which would make the resulting **ARQ** scheme not DO.

## 5.1 Our Approach

*Modified Fischer-Heun (FH) algorithm.* In light of the aforementioned concerns, we describe how to adapt the scheme of Fischer and Heun [28] (FH) in such a way that the resulting *plaintext* scheme satisfies the constraints given by Corollary 4.4 and maintains the $O(m)$ space and $O(1)$ query time requirements in exchange for the restriction that clients are not allowed to issue range queries *with length that is less than some small size* $s$. Our modifications result in a DO scheme.

We first present a slightly simplified version of FH which will be suitable for the description of our encrypted approach. FH works by first partitioning the table $\mathsf{DB}$ into blocks $B_1, \ldots, B_{m/s}$ of size $s = \log m$. Then, an array $\mathsf{A}$ of size $m/s = m/\log m = O(m)$ is generated, where $\mathsf{A}[i] = \langle \min(B_i), \mathsf{indexof}(\mathsf{DB}, \min(B_i)) \rangle$. The ST technique is then applied over $\mathsf{A}$ to produce the sparse table $M$ of size $O(\frac{m}{\log m} \log \frac{m}{\log m}) = O(m)$. Finally, a normalization technique is applied to generate a *lookup table* of size $O(m)$ that stores the answers to all possible queries on arrays of size $s$. (We elide the details of this part of the scheme, since it does not affect our approach.) Queries are answered by taking the minimum of the minima for the following three ranges:

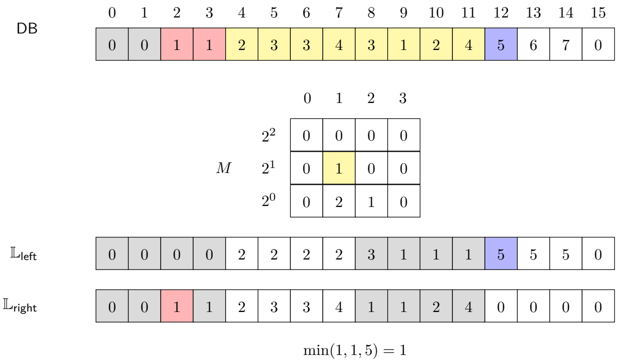
```
Let SparseTable(A) denote an application of the ST algorithm [5] on array A. Consider the aggregate range query scheme $\Pi_{\textsc{LinearMin}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ defined as follows:

– $\underline{\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})}$:
    1. initialize arrays $\mathsf{A}$, $\mathbb{L}_{\mathsf{left}}$, and $\mathbb{L}_{\mathsf{right}}$;
    2. compute $s = \log m/4$;
    3. partition $\mathsf{DB}$ into blocks $B_1, \ldots, B_{m/s}$ of size $s$;
    4. for $i \in [m/s]$, set $\mathsf{A}[i] := \min(B_i)$;
    5. compute $M \leftarrow \mathtt{SparseTable}(\mathsf{A})$;
    6. for $i \in [m]$,
        (a) compute block index $b$ of index $i$;
        (b) set $\mathbb{L}_{\mathsf{left}}[i] := \min_{sb \le k \le i} \mathsf{DB}[k]$;
        (c) set $\mathbb{L}_{\mathsf{right}}[i] := \min_{i \le k \le s(b+1)-1} \mathsf{DB}[k]$;
    7. output $\mathsf{DS} \leftarrow (M, \mathbb{L}_{\mathsf{left}}, \mathbb{L}_{\mathsf{right}})$;
– $\underline{U \leftarrow \mathbb{Q}(m, Q)}$:
    1. parse $(\ell, r) \leftarrow Q$;
    2. if $r - \ell < s$, abort;
    3. compute $b_\ell \leftarrow \lceil \frac{\ell}{s} \rceil$ and $b_r \leftarrow \lfloor \frac{r}{s} \rfloor$;
    4. set $q_1 \leftarrow \ell$ and $q_2 \leftarrow r$;
    5. if $b_r - b_\ell > 1$,
        (a) compute $h \leftarrow \lfloor \log(b_r - b_\ell + 1) \rfloor$;
        (b) compute $q_3 \leftarrow (b_\ell, h)$ and $q_4 \leftarrow (b_r - 2^h + 1, b_\ell)$;
        (c) output $(q_1, q_2, q_3, q_4)$;
    6. else, output $U \leftarrow (q_1, q_2)$;
– $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
    1. output $(\bot, \min(S), \bot)$;
```

**Fig. 2.** The plaintext $\Pi_{\textsc{LinearMin}}$ scheme used to instantiate the encrypted LinearMin scheme.

1. From $\ell$ to the end of $\ell$'s block using the normalization table to find the index of the minimum, then using $\mathsf{A}$ to retrieve the actual minimum.
2. The range spanning all blocks between $\ell$'s block and $r$'s block using $M$ (does not require accessing $\mathsf{A}$).
3. From the start of $r$'s block to $r$ using the normalization table, then using $\mathsf{A}$ for the actual minimum.

However, directly transforming the FH scheme into an STE structure results in problematic leakage from the lookup table invoked in subproblems 1 and 3.

Thus, we now introduce $\Pi_{\textsc{LinearMin}}$, our modified FH scheme. Figure 2 outlines the algorithm for the scheme, and Figure 3 provides an example of the scheme. Our scheme is exactly the same as in the FH algorithm except with the following modifications. First, the user is prevented from making some queries that are smaller than $s$—in particular, those queries that lie entirely within a block $B_i$. (For example, a database of size $m = 2^{64}$ has block size $s = \log 2^{64} = 64$, and so some queries of size smaller than 64 are not permissible.) This allows us to avoid using the lookup tables from the original FH scheme. However, with a simple



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DB | 0 | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 3 | 1 | 2 | 4 | 5 | 6 | 7 | 0 |

| $M$ | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| | $2^2$ | 0 | 0 | 0 | 0 |
| | $2^1$ | 0 | 1 | 0 | 0 |
| | $2^0$ | 0 | 2 | 1 | 0 |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbb{L}_{\mathsf{left}}$ | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 3 | 1 | 1 | 1 | 5 | 5 | 5 | 0 |
| $\mathbb{L}_{\mathsf{right}}$ | 0 | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 1 | 1 | 2 | 4 | 0 | 0 | 0 | 0 |

$\min(1, 1, 5) = 1$

**Fig. 3.** Minimum scheme on database of size $m = 16$. The highlighted regions show how to answer the example minimum query $\min(\mathsf{DB}, [2, 12]) = \min(\{1, 1, 5\}) = 1$ using the index.

modification, we can still answer arbitrary queries of size at least $s$. To do this, we create two arrays $\mathbb{L}_{\mathsf{left}}$ and $\mathbb{L}_{\mathsf{right}}$ of size $m$ which we use to precompute *one-sided queries* within each block $B_i$. Specifically, given index $i$ in block index $b$, we assign $\mathbb{L}_{\mathsf{left}}[i] := \min_{sb \leq k \leq i} \mathsf{DB}[k]$ and $\mathbb{L}_{\mathsf{right}}[i] := \min_{i \leq k \leq s(b+1)-1} \mathsf{DB}[k]$. We can then answer a query $[\ell, r]$ by retrieving the answers for at most three of the following subproblems:

1. From $\ell$ to the end of $\ell$'s block using $\mathbb{L}_{\mathsf{right}}$.
2. The range spanning all blocks between $\ell$'s block and $r$'s block using $M$.
3. From the beginning of $r$'s block to $r$ using $\mathbb{L}_{\mathsf{left}}$.

Since any query of size $s$ or greater must touch or overlap at least one block boundary, we may answer any such query using the structures above. Thus, all queries are answerable in constant time. Queries that are smaller than size $s$ also may be answered provided that they touch or overlap at least one block boundary.

LinearMin*: A New Linear-Space Scheme.* Combination of $\Pi_{\textsc{LinearMin}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ with the **ARQ** framework results in a provably secure scheme, LinearMin $= \mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$, for the encrypted range minimum problem with the leakage profile given in Theorem 4.2.

**Lemma 5.2.** $\Pi_{\textsc{LinearMin}}$ *is data independent.*

*Proof (sketch).* Given any pair of databases $(\mathsf{DB}_0, \mathsf{DB}_1)$, $\mathbb{S}$ always outputs $M$, $\mathbb{L}_{\mathsf{left}}$, and $\mathbb{L}_{\mathsf{right}}$ with the same respective size regardless of whether or not $\mathsf{DB}_0$ or $\mathsf{DB}_1$ is chosen. Thus, $|\mathsf{DS}_0| = |\mathsf{DS}_1|$. Additionally, given a query $Q$, $\mathbb{Q}$ always outputs the same set of subqueries $U_0$ for $Q$ (as it is deterministic and has no knowledge of the underlying $\mathsf{DB}$). This implies that the size of the initial subresponse set $S_0$ is identical for both databases. Also, $\mathbb{R}$ always outputs $\perp$ for $U_j$ for $j > 0$, so we know that $U_{0:j} = U_{1:j}$ for all $j > 0$. □

**Theorem 5.3.** *Given* $\Pi_{\textsc{LinearMin}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *and a response-hiding STE scheme* $\Sigma$ *on data structure* $\mathsf{DS}$ *with storage space* $O(|\mathsf{DS}|)$ *and constant query time and space, the scheme* LinearMin $= \mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ *is* DO, *supports queries on ranges of size at least* $\log m$, *and requires* $O(m)$ *storage,* $O(1)$ *query time at both the client and server, and* $O(1)$ *bandwidth.*

Theorem 5.3 follows by Lemma 5.2 and from [28].

*Workarounds for query size limitation.* One simple solution to minimize the number of queries of size $s$ that are blocked by the scheme is to recursively apply the scheme again within blocks of size $s$. This maintains the asymptotics of the construction (at the expense of a larger coefficient on the storage overhead) and decreases the threshold where certain queries cannot be answered from $s = \log m$ to $s' = \log \log m$.

Additionally, in practical deployments, the encrypted minimum structure may be stored alongside an record-reporting encrypted range structure (e.g., [20, 25, 27]). In such settings, the client may use a *split execution* strategy [62] where, when faced with an unanswerable query, the client can send a range query to the encrypted range structure. Then, the client can compute the minimum by decrypting the returned records client-side and taking the minimum of the query attribute. Given that this is required only for some *small* queries (of size less than $s$), the client still may enjoy the performance and leakage benefits of the LinearMin scheme on larger queries while incurring minimal performance overhead for the small queries.

# 6 Range Mode Query

In this section, we propose a novel scheme for the *encrypted approximate range mode query* problem, which asks for the element with the maximum number of occurrences in a given range of an array. The current state-of-the-art storage bound for constant-time range mode queries over static arrays is $O(n^2 \log \log n / \log^2 n)$ [59]. Given this near-quadratic storage requirement, we instead consider an *approximate* version of the range mode query problem. Similar approximations are available in several database implementations to improve performance (e.g., the `APPROXIMATE_MEDIAN` and `APPROXIMATE_MODE` functions in Vertica [63]). Introduced by Bose et al. [12], the *approximate range mode query* problem asks us to return an element of the queried range whose frequency is at least $0 < \alpha < 1$ times that of the frequency of the actual mode of $A'$.

**Definition 6.1** ($\alpha$-approximate mode query). *Given an array $A$ of size $n$ and indices $i$ and $j$ where $0 \leq i \leq j < n$, let $A'$ be the multiset of elements comprised of all elements of $A$ between the indices $i$ and $j$ (inclusive). We say that an element $x \in A'$ is an $\alpha$-*approximate mode *of $A'$ if* $\mathsf{freq}(x, [i, j]) \geq \alpha \cdot \mathsf{freq}(x, [i, j])$ *where* $\mathsf{freq}(x, [i, j])$ *returns the frequency of $x$ in the range $[i, j]$ and $0 < \alpha < 1$. The $\alpha$-*approximate range mode query *returns an $\alpha$-approximate mode of $A'$.*

Observe that the accuracy of the approximation increases as $\alpha$ tends to 1.

*Other existing plaintext structures.* Several solutions have been proposed that solve the approximate range mode problem with varying degrees of efficiency. Many of these schemes use bit-packing techniques that, as explained previously, we can not directly translate to STE structures (e.g., [33, 52]) or rely on theoretical data structures with no practical implementation (e.g., [23]). Additionally, while some recent schemes achieve lower storage requirements than our chosen approach, their query algorithms require non-constant query time (e.g., [23]). In all such cases, queries required at least 2 round trips to the structure and subsequent accesses to the query structure depended on the result of the initial access. These features meant that the schemes would not be data independent as changes to the underlying database would result in different access patterns; additionally, as shown by Corollary 4.4, the interactivity of the scheme makes the schemes non-data-independent.

## 6.1 Our Approach

*The BKMT scheme.* Given the above constraints, we focus our attention on the structures developed by Bose, Kranakis, Morin, and Tang (BKMT) [12]. The BKMT scheme relies on the following technical theorem.

**Theorem 6.2** (Mode partition [12]). *If $\{B_1, \ldots, B_k\}$ is a $k$-partition of the range $[\ell, r]$, then* $\arg\max_p \mathsf{mode}(B_p) \geq \frac{\mathsf{mode}([\ell,r])}{k}$*, where* $\mathsf{mode}$ *is the function that returns the mode of the input range on $A$. Then, of the $k$ submodes* $\mathsf{mode}(B_1), \ldots, \mathsf{mode}(B_k)$*, the submode with the highest frequency is an $1/k$-approximate mode of $[\ell, r]$.*

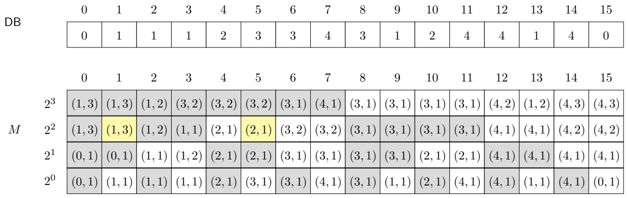
Consider the aggregate range query scheme $\Pi_{1/2\text{-}\textsc{ApproxMode}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ defined as follows:

```
– $\underline{\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})}$:
    1. compute $k \leftarrow \lceil \log m \rceil$;
    2. for $p \in [0, k]$,
        (a) for $b$ in $[0, \lceil m/2^p \rceil]$,
            i. compute $s \leftarrow b \cdot 2^p$;
            ii. compute $h \leftarrow \min(s + 2^{p-1}, m)$;
            iii. for $1 \leq i \leq h$, compute the $(\mathsf{mode}, \mathsf{freq})$
                 pair for the range $[1, i]$ and set in $M[(p, i)]$;
            iv. for $h + 1 \leq j \leq m$, compute the
                $(\mathsf{mode}, \mathsf{freq})$ pair for the range $[h{+}1, j]$ and
                set in $M[(p, j)]$;
    3. output $\mathsf{DS} \leftarrow M$;
– $\underline{U \leftarrow \mathbb{Q}(m, Q)}$:
    1. parse $(\ell, r) \leftarrow x$;
    2. compute $\mathsf{level} \leftarrow 1 + \lfloor \log(\ell \oplus r) \rfloor$;
    3. output $((\mathsf{level}, \ell), (\mathsf{level}, r))$;
– $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
    1. compute $(R, f) \leftarrow \arg\max_{(x, f_x) \in S} f_x$;
    2. output $(\perp, R, \perp)$;
```

**Fig. 4.** The plaintext $\Pi_{1/2\text{-}\textsc{ApproxMode}}$ scheme used to instantiate the encrypted 1/2$-$ApproxMode scheme.

Yao [66] and Alon and Schieber [2] provide the necessary $k$-partitioning scheme for Theorem 6.2. We refer to this technique as the *AS technique*. AS is similar to the previously mentioned ST technique except that, in AS, the $k$-intervals in a given partition do *not* overlap. Here, we describe the AS partitioning algorithm for the case when $k = 2$. For $k = 2$, AS splits the array at point $h = 2^{\lfloor \log m \rfloor}$ into two blocks. Then, for $1 \leq i \leq h$, AS precomputes the answers to all intervals $[i, h]$. Similarly, for $h + 1 \leq j \leq m$, AS precomputes the answers to all intervals $[h + 1, j]$. These intervals allow us to answer any query $[\ell, r]$ that intersects the halfway point $h$. To answer queries that fall entirely within one half of the array (i.e. where $r \leq h$ or $\ell \leq h+1$), AS recursively processes each half of the array with the same algorithm. Each level of the resulting table represents $m$ intervals computed at one of the $O(\log m)$ recursive steps.

The BKMT solution follows as a combination of Theorem 6.2 and the AS technique. At setup time, we generate a $k$-interval AS table $M$ that stores the mode of each interval and the mode's frequency. Then, to answer a query $[\ell, r]$, we query the $k$ non-overlapping intervals in $M$ that exactly cover $[\ell, r]$ to retrieve their modes. By Theorem 6.2, the mode with the highest frequency of all $k$ modes is a $1/k$-approximate mode of $[\ell, r]$. This scheme is described in Figure 4.



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DB | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 4 | 3 | 1 | 2 | 4 | 4 | 1 | 4 | 0 |

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $2^3$ | (1,3) | (1,3) | (1,2) | (3,2) | (3,2) | (3,2) | (3,1) | (4,1) | (3,1) | (3,1) | (3,1) | (3,1) | (4,2) | (1,2) | (4,3) | (4,3) |
| $M$ | $2^2$ | (1,3) | (1,3) | (1,2) | (1,1) | (2,1) | (2,1) | (3,2) | (3,2) | (3,1) | (3,1) | (3,1) | (3,1) | (4,1) | (4,1) | (4,2) | (4,2) |
| | $2^1$ | (0,1) | (0,1) | (1,1) | (1,2) | (2,1) | (2,1) | (3,1) | (3,1) | (3,1) | (3,1) | (2,1) | (2,1) | (4,1) | (4,1) | (4,1) | (4,1) |
| | $2^0$ | (0,1) | (1,1) | (1,1) | (1,1) | (2,1) | (3,1) | (3,1) | (4,1) | (3,1) | (1,1) | (2,1) | (4,1) | (4,1) | (1,1) | (4,1) | (0,1) |

**Fig. 5.** $1/2$-approximate mode scheme on database of size $m = 16$. Adjacent shaded and non-shaded areas in the same row of $M$ denote which domain values of DB are used to compute the intervals at that level. The first item of each tuple in $M$ is a mode (element); the second item of the tuple is the mode's frequency. The highlighted regions show which entries are queried to answer the example query $[1, 5]$—picking the highlighted tuple with the larger second element gives us $(1, 3)$, and the first element of the tuple $(1)$ is the $1/2$-approximate mode.

1/2-ApproxMode*: A New Encrypted Mode Scheme.* Combination of $\Pi_{1/2\text{-ApproxMode}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ with the **ARQ** framework results in a provably secure scheme, 1/2-ApproxMode $=$ $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$, for the encrypted range mode problem with the leakage profile given in Theorem 4.2.

**Lemma 6.3.** $\Pi_{1/2\text{-ApproxMode}}$ *is data independent.*

Lemma 6.3 can be proved similarly to Lemma 5.2.

**Theorem 6.4.** *Given* $\Pi_{1/2\text{-ApproxMode}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *and a response-hiding STE scheme* $\Sigma$ *on data structure* DS *with storage space* $O(|\mathsf{DS}|)$ *and constant query time and space, the scheme* 1/2-ApproxMode $=$ $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ *is* DO *and requires* $O(m \log m)$ *storage,* $O(1)$ *query time at both the client and server, and* $O(1)$ *bandwidth.*

Theorem 6.4 follows by Lemma 6.3 and from [12].

1/3-ApproxMode*.* 1/3-ApproxMode, a scheme for answering encrypted 1/3-approximate range mode queries, is equivalent to 1/2-ApproxMode except we modify $\Pi_{1/2\text{-ApproxMode}}$ to use a 3-interval AS table. Queries then involve three intervals instead of two. This improves the storage requirement to $O(m \log \log m)$ in exchange for a wider approximation. We omit the full description of this approach and direct the reader to [2] for the $k$-interval partition algorithm for $k > 2$.

# 7 Range Median Query

In this section, we propose a novel encrypted scheme for the *encrypted approximate range median query* problem.

Like the range mode problem, the exact range median query problem has near-quadratic storage overhead for constant-time queries, with the best known solution requiring $O(n^2 \log^{(k)} n/ \log n)$ storage, where $k$ is an arbitrary constant and $\log^{(k)}$ is the iterated logarithm function [58]. We thus concentrate on solutions for the *approximate* version of the problem introduced by Bose et al. [12]. Given $0 < \alpha < 1$, the $\alpha$-approximate median query asks for an element of the queried range whose rank is within $\pm\alpha/2$ of the rank of the true median.

**Definition 7.1** (Approximate median query)**.** *Given an array* $A$ *of* $n$ *numbers and indices* $i$ *and* $j$ *where* $0 \leq i \leq j < n$*, let* $A'$ *be the multiset of elements comprised of all elements of* $A$ *between the indices* $i$ *and* $j$ *(inclusive). Given* $0 < \alpha < 1$*, we say that an element* $x \in A'$ *is an* $\alpha$-approximate median *of* $A'$ *if the percentile rank of* $x \in A'$*, denoted* $r_x$*, satisfies* $r_x \in \left[\frac{1}{2} - \frac{1-\alpha}{2}, \frac{1}{2} + \frac{1-\alpha}{2}\right]$. *The* $\alpha$-approximate range median query *returns an* $\alpha$*-approximate median of* $A'$*.*

Note that in the above definition, the accuracy of the approximation increases as $\alpha$ tends to 1.

*Other existing plaintext structures.* We know of one other constant time and bandwidth approximate median scheme [23]. It uses fusion trees, a theoretical data structure with no practical implementation, so we do not consider it in this work.

## 7.1 Our Approach

*A modified version of the BKMT scheme.* We use a scheme based on the $\alpha$-approximate range median index from Bose, Kranakis, Morin, and Tang (BKMT) [12]. In our testing, we found that the original BKMT query algorithm sometimes produced queries to indices that were outside the bounds of the structure; this was the case whether or not the domain size matched the paper's assumption that it was a strict power of two. However, we found that the setup algorithm from BKMT was correct. The scheme we present here is identical to the BKMT scheme with a modified query algorithm that fixes the out-of-bounds issues but preserves the constant time and size query overhead.

We refer to our modified scheme as $\Pi_{\alpha\text{-ApproxMedian}}$. Figure 6 defines the scheme and Figure 7 illustrates an example. The algorithm relies on the following observation: given a sufficiently long query interval, the effects of the elements in a small prefix and suffix of the interval are minimal on the median and thus can be

Let BKMTApproxMed($\alpha$, DB) denote an application of the BKMT $\alpha$-approximate median algorithm over table DB [5]. Consider the aggregate range query scheme $\Pi_{\alpha\text{-ApproxMedian}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ defined as follows:

```
– DS ← $\mathbb{S}$(DB):
    1. initialize dictionary DX;
    2. compute $k \leftarrow \lceil \log m \rceil$ and $p \leftarrow \lceil \frac{2\cdot(1+\alpha)}{1-\alpha} \rceil$;
    3. for $1 \le i \le k$,
        (a) compute $b \leftarrow 2^{k-i}$;
        (b) for $1 \le j \le \lceil \frac{m}{b} \rceil$,
            i. initialize list medians;
            ii. for $1 \le x \le p$,
                A. compute $s \leftarrow \min((j-1)\cdot b, m-1)$;
                B. compute $e \leftarrow \min(s + (x \cdot b), m)$;
                C. compute the median from $s$ to $e$ in DB
                   and append it to medians;
            iii. set DX$[(i,j)]$ := medians;
    4. output DS ← (DX);
– $U \leftarrow \mathbb{Q}(m, Q)$:
    1. parse $(\ell, r) \leftarrow x$;
    2. compute max_blocks ← $2 \cdot \lceil \frac{2\cdot\alpha}{1-\alpha} \rceil$;
    3. compute $k \leftarrow \lceil \log m \rceil$;
    4. compute initial_level ← $k - \lfloor \log(r-\ell) \rfloor + 1$;
    5. compute offset ← $\lfloor \log(\text{max\_blocks} + 2) \rfloor - 2$
    6. compute $i \leftarrow \min(\text{initial\_level} + \text{offset}, k)$;
    7. compute $b \leftarrow 2^{k-i}$;
    8. compute $b_\ell \leftarrow \lceil \frac{\ell}{b} \rceil + 1$ and $b_r \leftarrow \lfloor \frac{r}{b} \rfloor$;
    9. output $U \leftarrow (\langle i, b_\ell \rangle)$;
– $(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)$:
    1. recompute $b_\ell$ and $b_r$ using $\mathbb{Q}$;
    2. compute $x \leftarrow b_r - b_\ell$;
    3. output $(\bot, S[x], \bot)$;
```

**Fig. 6.** The plaintext $\Pi_{\alpha\text{-ApproxMedian}}$ scheme used to instantiate the encrypted $\alpha$-ApproxMedian scheme.

| DB | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 1 | 2 | 3 | 3 | 4 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $T_1$ | $1,1,\bot$ | $3,\bot,\bot$ | | | | | | |
| $T_2$ | $0,1,1$ | $1,1,2$ | $2,3,\bot$ | $3,\bot,\bot$ | | | | |
| $T_3$ | $0,0,1$ | $1,1,1$ | $1,1,1$ | $1,1,2$ | $2,2,3$ | $3,3,3$ | $3,3,\bot$ | $4,\bot,\bot$ |

**Fig. 7.** Median scheme on example DB of size $m = 16$, $\alpha = \frac{1}{5}$.

ignored. Thus, we can avoid precomputing some sub-medians while still achieving the desired approximation.

Given a table DB with domain size $m = 2^k$ for some $k \geq 1$, BKMT creates a hierarchical set of arrays $T_1, \ldots, T_k$ as follows. For $1 \leq i \leq k$, we partition DB into $2^i$ blocks of size $m/2^i$. Then, $T_i$ contains $2^i$ entries $T_i[j]$, each of which corresponds to the $j$th block of size $m/2^i$, denoted $B_{i:j}$. Then, each $T_i[j]$ is a list containing $p = \lfloor \frac{2(1+\alpha)}{1-\alpha} \rfloor$ elements of DB, where, for all $1 \leq x \leq p$, $T_i[j][x] := \mathsf{median}(\bigcup_{0 \leq q \leq x-1} B_{i:(j+q)})$.

Each array $T_i$ contains $2^i$ lists, each containing $\lfloor \frac{2(1+\alpha)}{1-\alpha} \rfloor$ elements. Thus, each $T_i$ is of size $O(\frac{2^i(1+\alpha)}{1-\alpha})$. There are $\log m$ arrays, so the total space needed to store them is $\sum_{i=1}^{\log m} O(\frac{2^i(1+\alpha)}{1-\alpha}) = O(\frac{m(1+\alpha)}{1-\alpha}) = O(\frac{m}{1-\alpha})$. Answering a query $[\ell, r]$ can be done in $O(1)$ by accessing a single element in $T$ using the following algorithm:

1. Determine which $T_i$ was generated from blocks that are as large as possible but still fit within the query length $r - \ell$ (lines 2 through 6 in Figure 7).
2. Given the block size $b \leftarrow 2^{k-i}$ used to generate $T_i$, compute block indices of $\ell$ and $r$ as $b_\ell = \lceil \frac{\ell}{b} \rceil + 1$ and $b_r = \lfloor \frac{r}{b} \rfloor$.
3. Output $T_i[b_\ell][b_r - b_\ell] = \mathsf{median}(\bigcup_{0 \leq q \leq b_r} B_{i:(b_\ell+q)})$.

$\alpha$-ApproxMedian*: A New Encrypted Median Scheme.* Combination of $\Pi_{\alpha\text{-ApproxMedian}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ with the **ARQ** framework results in a provably secure scheme, $\alpha$-ApproxMedian $= \mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$, for the encrypted range median problem with the leakage profile given in Theorem 4.2.

**Lemma 7.2.** $\Pi_{\alpha\text{-ApproxMedian}}$ *is data independent.*

Lemma 7.2 can be proved similarly to Lemma 5.2.

**Theorem 7.3.** *Given* $\Pi_{\alpha\text{-ApproxMedian}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *and a response-hiding STE scheme* $\Sigma$ *on data structure* DS *with storage space* $O(|\mathsf{DS}|)$ *and constant query time and space, the scheme* $\alpha$-ApproxMedian $= \mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ *is* DO *and requires* $O(\frac{m}{1-\alpha})$ *storage,* $O(1)$ *query time at both the client and server, and* $O(1)$ *bandwidth.*

Theorem 7.3 follows by Lemma 7.2 and from [12].

## 8 Exploiting Sparsity

We now present a set of storage optimizations for encrypted aggregate range query structures over *sparse* databases. We refer to our techniques as *domain reductions*, as they allow the client to reduce the size of the domain that the aggregate structure is built over. Our reductions themselves are plaintext aggregate range query schemes that convert the query space of an encrypted data structure to a smaller domain space, while still allowing the client to make queries in the original

```
Let $\Pi' = (\mathbb{S}', \mathbb{Q}', \mathbb{R}')$ be a plaintext aggregate range query scheme. Consider the aggregate range query scheme $\Pi_{\textsc{DomainBucket}(\mathbb{S}',\mathbb{Q}',\mathbb{R}')} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ defined as follows:

– $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$:
    1. initialize multimap $\mathsf{MM}$;
    2. for all $0 \leq i \leq m^{1-\alpha}$,
        (a) compute $\ell \leftarrow im^{\alpha}$ and $r \leftarrow ((i+1)m^{\alpha}) - 1$;
        (b) compute $l_1 \leftarrow \mathsf{DB}^{\leftarrow}(\ell - 1)$ and $l_2 \leftarrow \mathsf{DB}^{\rightarrow}(r+1)$;
        (c) compute $\mathsf{bucket}_i \leftarrow \{\langle j, \mathsf{DB}(j)\rangle \mid j \in [\ell, r]\} \cup \{l_1, l_2\}$;
        (d) set $\mathsf{MM}[\langle \ell, r\rangle] := \mathsf{bucket}_i$;
    3. compute $\mathsf{DS}' \leftarrow \mathbb{S}'(\mathsf{DB})$;
    4. output $\mathsf{DS} \leftarrow (\mathsf{MM}, \mathsf{DS}')$;
– $U \leftarrow \mathbb{Q}(m, Q)$:
    1. parse $(\ell, r) \leftarrow Q$;
    2. compute $i_\ell \leftarrow \lfloor \frac{\ell}{m^{1/\alpha}} \rfloor$ and $i_r \leftarrow \lfloor \frac{r}{m^{1/\alpha}} \rfloor$;
    3. compute $L_\ell \leftarrow i_\ell m^{\alpha}$ and $R_\ell \leftarrow ((i_\ell + 1)m^{\alpha}) - 1$;
    4. compute $L_r \leftarrow i_r m^{\alpha}$ and $R_r \leftarrow ((i_r + 1)m^{\alpha}) - 1$;
    5. output $(\langle L_\ell, R_\ell\rangle, \langle L_r, R_r\rangle)$;
– $(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)$:
    1. if $\mathrm{st} = \perp$,
        (a) parse $(B_\ell, B_r) \leftarrow S$;
        (b) compute the closest element $\ell$ to $x$ in bucket $B_\ell$ in the right direction;
        (c) compute the closest element $r$ to $x$ in bucket $B_r$ in the left direction;
        (d) compute $U \leftarrow \mathbb{Q}(m, \langle \ell, r\rangle)$;
        (e) set $\mathrm{st}' \leftarrow (\langle \ell, r\rangle, \perp)$;
        (f) output $(\mathrm{st}', \perp, U)$;
    2. otherwise,
        (a) parse $(\langle \ell, r\rangle, \mathrm{st}_{\mathbb{R}}) \leftarrow \mathrm{st}$;
        (b) compute $(\mathrm{st}'_{\mathbb{R}}, R, U) \leftarrow \mathbb{R}'(\mathrm{st}_{\mathbb{R}}, m, \langle \ell, r\rangle, S)$;
        (c) set $\mathrm{st}' \leftarrow (\langle \ell, r\rangle, \mathrm{st}'_{\mathbb{R}})$;
        (d) output $(\mathrm{st}', R, U)$;
```

**Fig. 8.** The domain reduction $\Pi_{\textsc{DomainBucket}(\mathbb{S}',\mathbb{Q}',\mathbb{R}')}$.

query space. In general, our reductions enable the client to efficiently find the ID of the record whose query attribute is closest to the client's desired query. Our reductions significantly reduce storage with different *tradeoffs* between query performance and leakage.

Our domain reduction schemes are presented in terms of the plaintext aggregate range query syntax from Definition 4.1, though the reductions themselves are parameterized by a secondary scheme. Defining our reductions in this way allows us to immediately apply the **ARQ** framework to our reductions to derive encrypted aggregate range query schemes.

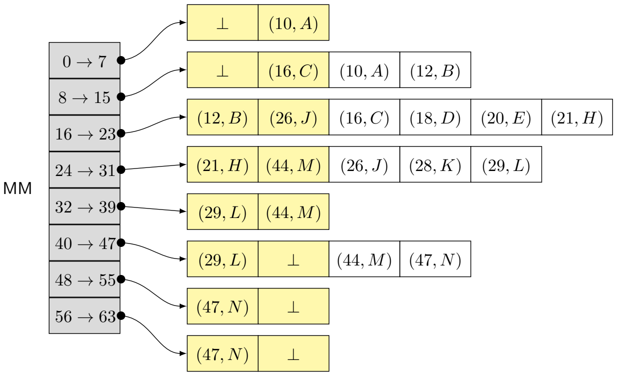

MM
0 → 7
8 → 15
16 → 23
24 → 31
32 → 39
40 → 47
48 → 55
56 → 63
⊥ | (10, A)
⊥ | (16, C) | (10, A) | (12, B)
(12, B) | (26, J) | (16, C) | (18, D) | (20, E) | (21, H)
(21, H) | (44, M) | (26, J) | (28, K) | (29, L)
(29, L) | (44, M)
(29, L) | ⊥ | (44, M) | (47, N)
(47, N) | ⊥
(47, N) | ⊥

**Fig. 9.** DomainBucket example on a database of size $m = 64$. Highlighted cells represent the lookup portion of each bucket.

*The* DomainBucket *reduction.* Figure 9 illustrates the DomainBucket reduction; the algorithm is defined in Figure 8. The reduction takes as input the domain size $m$, the database DB, and a tunable constant $0 < \alpha < 1$. We partition the domain of DB into non-overlapping "buckets" where each bucket has equal size. More precisely, we instantiate a multimap MM with keys $\{[im^{\alpha}, ((i+1)m^{\alpha}) - 1] \mid i \in [m^{1-\alpha}]\}$, where the $i$th label holds tuples corresponding to the all of the records in the range $[im^{\alpha}, ((i+1)m^{\alpha}) - 1]$. Each tuple has two elements: the original domain value and the ordered ID of the record (i.e., the transformed domain).

We also store two additional entries in each bucket that we call "backwards/forwards lookup" entries. The first entry corresponds to the record that is closest to the left endpoint of the bucket, but is not within the bucket range itself; the second record is for the right endpoint of the bucket. This allows the client to receive a valid answer to a query even if their original query falls in a bucket with no records.

When making a query to the structure, the client sends a search token corresponding to the *bucket* containing the domain point. The server responds with the entire bucket. The client then does a linear pass to find the record ID of the closest element to $x$.

*Complexity of* DomainBucket. DomainBucket requires only a constant-size search token corresponding to the desired bucket range of size $m^{\alpha}$. However, the response bandwidth from the server to the client is potentially linear—consider a database where all of the records are in a single bucket range, and thus a query to that range will return $O(n)$ records. Thus, the worst-case client computation complexity is $O(n)$, as such a database would require the client to sort through all $O(n)$ records to recover the nearest populated point.

*The* DataBucket *reduction.* The volume leakage and the worst-case-linear bandwidth of the DomainBucket scheme motivates our second domain reduction con-

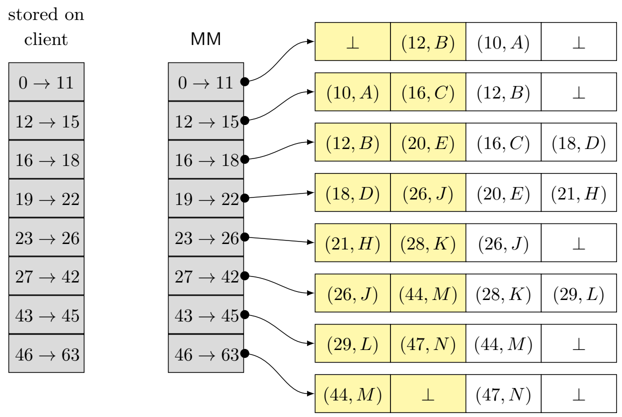

stored on client
MM
0 → 11
12 → 15
16 → 18
19 → 22
23 → 26
27 → 42
43 → 45
46 → 63
⊥ | (12, B) | (10, A) | ⊥
(10, A) | (16, C) | (12, B) | ⊥
(12, B) | (20, E) | (16, C) | (18, D)
(18, D) | (26, J) | (20, E) | (21, H)
(21, H) | (28, K) | (26, J) | ⊥
(26, J) | (44, M) | (28, K) | (29, L)
(29, L) | (47, N) | (44, M) | ⊥
(44, M) | ⊥ | (47, N) | ⊥

**Fig. 10.** DataBucket example on a database of size $m = 64$. Highlighted cells represent the lookup portion of each bucket.

struction, DataBucket (Figure 10). We elide the formal protocol description due to the scheme's similarities with the DomainBucket scheme. DataBucket is identical to the DomainBucket scheme except that each bucket contains the *same* number of records. Then, the client maintains $O(m^\alpha)$ storage to restore these buckets in future queries. Buckets are padded with $\perp$ to ensure that their volumes are indistinguishable. This brings the bandwidth in the worst case from $O(m)$ to $O(m^\alpha)$ (i.e. the maximum size of a bucket).

*Example: range sum queries.* With our domain reduction schemes in hand, we use this section as an example of how our domain reductions can be used to derive optimizations to the encrypted range sum scheme presented by Demertzis et al. [20], which require $O(m)$ storage.

**Definition 8.1** (Range sum query)**.** *Given an array $A$ of $n$ numbers and indices $i$ and $j$ where $0 \le i \le j < n$, the* range sum query *computes $\sum_{k=i}^{j} A[k]$, the sum of all $A[k]$ where $i \le k \le j$.*

Solutions to the range sum query problem can be used to answer *count*, *average*, and *variance* queries in exchange for a constant factor increase in time and space. We provide examples of such *query type transformation* techniques for encrypted databases in Appendix C in lieu of considering specialized structures for those types.

*Previous results.* Demertzis et al. [20] presented the first and only encrypted range sum query STE scheme, DPPDGP-Sum, which was based on the classic *prefix sums* technique [9]. This technique allows for constant-size and constant-time queries in exchange for $O(m)$ storage. In this technique, an array $A$ of size $m$ is computed such that, for all $x \in [m]$, $A[x] := \sum_{i=0}^{x} \mathsf{DB}(x)$. Then, a range sum query $[\ell, r]$ may be answered in constant time by accessing $A[r]$ and $A[\ell-1]$, then computing

$$A[r] - A[\ell-1] = \sum_{i=0}^{r} \mathsf{DB}(x) - \sum_{i=0}^{\ell-2} \mathsf{DB}(x) = \sum_{i=\ell-1}^{r} \mathsf{DB}(x).$$

The DPPDGP-Sum scheme translates prefix sums to the encrypted range sum problem by encrypting $A$ with an array encryption scheme. (We note that **ARQ** can be used to derive this scheme and its leakage.)

*Our derived schemes.* Our reductions may be compiled directly on top of DPPDGP-Sum to reveal the asymptotics detailed in Table 1. To do this, we generate the prefix sums array $A$ over the $n$ sorted record IDs in $\mathsf{DB}$ as the domain values. (Observe that the "domain" of this prefix sums array is over the $O(n)$ record IDs of $\mathsf{DB}$.) Then, we apply our choice of reduction to the array $A$. When the client wants to make a query $[\ell, r]$, they make *two* queries to the reduction structure—one to find the nearest record ID $\mathsf{id}_\ell$ on the left of $\ell$, and another to find the nearest record ID $\mathsf{id}_r$ on the right of $r$. The client then queries $A$ using $[\mathsf{id}_\ell, \mathsf{id}_r]$.

Under our general domain reduction framework, the added structure normally incurs an additional round of communication. However, for prefix sums, we may avoid the additional round of communication due to the following: since every record ID has a one-to-one mapping with an entry in the prefix sums array, we can replace every instance of the record ID in the domain reduction structure with its entry in the prefix sums array. Now, when the client makes a query to the domain reduction structure, the structure's response is the desired sum.

*Tradeoffs and potential attacks.* While our reductions result in substantially lower storage overhead on sparse datasets, the leakage changes result in schemes that are not provably DO. However, the lack of the DO property does not mean that the schemes are immediately vulnerable to attacks. To provide more context on this tradeoff, we now describe starting points for attacks against the domain reductions and what assumptions may be necessary for such attacks to produce practical impact.

Consider some instantiation of $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ where $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ is derived from a domain reduction scheme. Given that the reduction index translates coordinates in the original query space $\mathbf{Q}^*$ into a new query space $\mathbf{Q}$ and, generally, $|\mathbf{Q}^*| < |\mathbf{Q}|$, at least two queries to the domain reduction index will be transformed into the same query to the aggregate index structure. In the specific reductions we propose in this work, this duplication may reveal information about the density distribution of the underlying database.

For example, in DataBucket or the implicit domain reduction used in DPPDGP-Min2, if the adversary knows that the client is issuing every original domain query exactly once and assuming that $\Sigma$ has search pattern leakage, they may count how many queries each bucket receives. More queries issued to some buckets imply that the dataset has different density levels, with the buckets receiving more queries corresponding to a sparser domain range. We note that we are currently not aware of any attack that allows the adversary to *order* these encrypted buckets, so while the adversary knows that some areas of the domain are sparser than others, they may not be able to determine the ordering of these densities without more assumptions. Attacks that leverage knowledge of the query distribution (e.g., [47, 55]) can be mitigated using frequency smoothing (e.g., Pancake [35]), leakage suppression frameworks (e.g., [31, 44]), or wrap-around query buffers [51]. In Appendix D, we discuss more mitigations and extensions.

# 9 Empirical Evaluation

We now evaluate how the schemes derived from the **ARQ** framework perform in practice.

*Arca: a new structured encryption library.* As part of this work, we designed and implemented a new, open-source, encrypted search library called Arca [24]. (Arca's code is available at https://github.com/cloudsecuritygroup/arca.) Arca is a Python package designed to allow researchers to easily and rapidly prototype systems that use encrypted search algorithms. It provides simple cryptographic primitives (which themselves are based on those provided by the Python `cryptography` package [61]) and implementations of various structured encryption schemes. Arca v0.1.0 consists of 1558 lines of code (excluding 1121 lines of tests) as counted by CLOC v1.92 [1]. Arca has complete type annotations and passes all strict type checks provided by MyPy.

We implemented all of the plaintext schemes that we chose in this work as well as the plaintext schemes used by Demertzis et al. [20]. Then, using Arca, we implemented the **ARQ** framework. Combining our implementation of the **ARQ** framework and our implementations of the plaintext schemes produced the LinearMin, 1/2-ApproxMode, $\alpha$-ApproxMedian, DomainBucket, and DataBucket encrypted schemes as well as the encrypted schemes from [20]. The plaintext implementations of these schemes and the **ARQ** framework are included within the Arca 0.1 library.

*Cryptographic primitives.* We used the cryptographic primitives provided by Arca. For symmetric encryption, we used AES-256 in CBC mode; for PRFs, we used SHA512. For each **ARQ** instantiation, we used Arca's implementation of the $\Pi_{\text{bas}}$ scheme from [14] as $\Sigma$.

*Experimental setup.* We ran our experiments single-threaded on a Slurm computing cluster consisting of Intel Xeon E5-2670 and E5-2600 processors. All experiments were performed in-memory, with each process allotted 300 GB of RAM. (Our experiment code is at https://github.com/cloudsecuritygroup/arq-experiments.)

*Datasets.* We use two real-world datasets in our evaluation. **Gowalla** [16] was a location-based social networking website in which users could share their locations. This dataset contains a total of 6,442,892 check-ins collected from users between February 2009 and October 2010, with 5,561,630 unique records. The date and time of the check-ins were converted to Unix time integers, then shifted to domain $\{0, \ldots, 54083068\}$. These normalized times were used as the query attribute. Using **Gowalla**, we demonstrate the effects of increasing the number of records on the scheme costs by randomly partitioning **Gowalla** into 12 sets of 500,000 points. We ran our schemes on one partition, then formed a new dataset by adding another partition to the previous partitions and benchmarked the costs again with the increased number of records.[1] We use this dataset (and this partitioning scheme) to replicate Demertzis et al.'s [20] evaluation of their schemes.

**Amazon** [54] contains 51,311,621 item ratings from reviews left in the *Books* section of Amazon between May 1996 and October 2018. There are 7,837 unique timestamps in the dataset. We normalized the timestamp of the reviews to the domain $\{0, \ldots, 7058880\}$.

*Quantitative evaluation.* Figure 11 demonstrates the effectiveness of our domain reductions in reducing the index size, construction time, and server query time of

[1] We were unable to run the DPPDGP-Min1 scheme on any partition set of **Gowalla** within 8 hours due to the number of encryptions needed to encrypt the ST. Thus, we instead computed the number of bytes needed to store the encrypted form of each entry of ST (162 B) and the average time for computing a hash of each entry's label and encrypting its value (15356 ns). Then, we computed the # of cells in the ST and extrapolated the index size and runtime of DPPDGP-Min1 using the previous two metrics. Note that our encryption time estimate is conservative, as it does not take into account the time it takes to access elements from the large (plaintext) ST array.

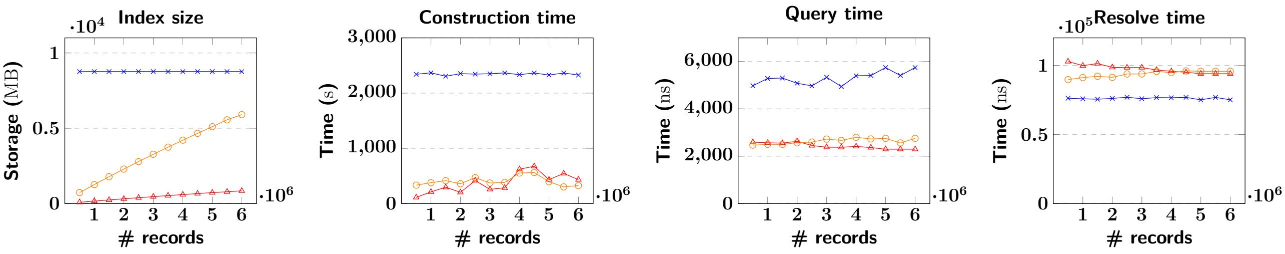


**Fig. 11.** Scheme costs on the **Gowalla** dataset with DPPDGP-Sum [20] (—×—), DomainBucket (—○—), and DataBucket (—△—).

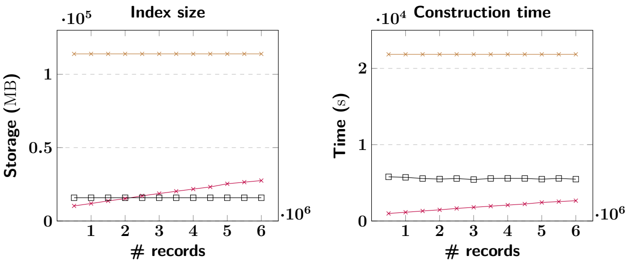


**Fig. 12.** Scheme costs on the **Gowalla** dataset with DPPDGP-Min1 [20] (—×—), DPPDGP-Min2 [20] (—×—), and LinearMin (—□—). There were no significant changes in Query time as the number of records increased (average Query times—DPPDGP-Min1: 1656 ns; DPPDGP-Min2: 2033 ns; LinearMin: 3014 ns). There were no significant changes in Resolve time as the number of records increased (average Resolve times—DPPDGP-Min1: 41508 ns; DPPDGP-Min2: 42058 ns; LinearMin: 121939 ns).

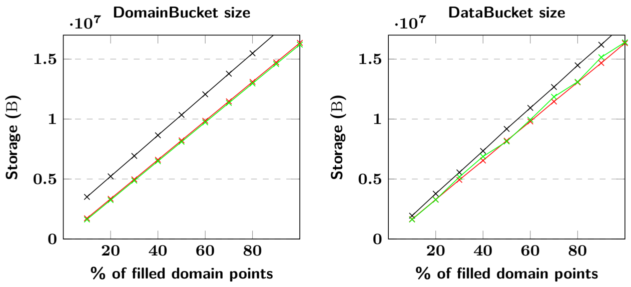


**Fig. 13.** Density evaluation of DomainBucket (left) and DataBucket (right) on synthetic datasets ($m = 100000$) with $\alpha = 0.25$ (—×—), $\alpha = 0.50$ (—×—), and $\alpha = 0.75$ (—×—).

DPPDGP-Sum, while only slightly increasing the resolve time at the client. In particular, the DataBucket scheme results in significantly lower storage and construction overhead than DomainBucket—since $m$ and $\alpha$ are public parameters and each bucket is the same size, the entire bucket may be encrypted as a single value which substantially reduces the amount of extra padding incurred by each encryption operation. The client-side performance tradeoffs with the reductions are made evident in the results for Resolve, but the runtime is minimally greater than that of DPPDGP-Sum.

For the minimum schemes, Figure 12 demonstrates that our LinearMin scheme (designed for *dense* databases) performs significantly better than DPPDGP-Min1 (its direct *dense* scheme competitor) *and* better than DPPDGP-Min2 (the scheme designed for "sparse" databases) starting at 2.5 million records in **Gowalla**. For the approximate schemes, Table 2 and Table 3 in the Appendix provide baseline performance benchmarks for the 1/2-ApproxMode and $\alpha$-ApproxMedian schemes for $\alpha = 0.5$. We can see that the storage and build time overhead of both schemes is comparable to that of DPPDGP-Min1.

In Figure 13, we plot the index size of our DomainBucket and DataBucket schemes under synthetic databases of different densities, with uniformly distributed data. We observe that the index size increases as the sparsity decreases, as expected in both cases. Due to the data distribution, we observe similar performance from the DomainBucket and DataBucket schemes.

All in all, our experiments and analysis demonstrate that our approach's theoretical constant-size and time query overhead is very small in practice. In particular, the overhead will be significantly smaller than the linear-time overhead of the strategies used in prior work mentioned in Section 2.

## Acknowledgements

Work supported in part by the National Science Foundation, the NetApp University Research Fund, and the Randy Pausch Undergraduate Summer Research Award at Brown University. Part of this research was conducted using computational resources of the Center for Computation and Visualization at Brown University.

# References

[1] AlDanial and other contributors, "AlDanial/cloc: 1.92," Zenodo, Dec. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5760077

[2] N. Alon and B. Schieber, "Optimal preprocessing for answering on-line product queries," The Moise and Frida Eskenasy Institute of Computer Sciences, Tel-Aviv University, Tech. Rep. 71/87, 1987.

[3] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal Security with Cipherbase," in *CIDR*, 2013.

[4] M. A. Bender and M. Farach-Colton, "The LCA Problem Revisited," in *LATIN 2000: Theoretical Informatics*, G. H. Gonnet and A. Viola, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 88–94.

[5] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *Journal of Algorithms*, 2005.

[6] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov, "The Tao of Inference in Privacy-Protected Databases," *Proc. VLDB Endow.*, vol. 11, no. 11, p. 1715–1728, jul 2018.

[7] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting Leakage Abuse Attacks," in *27th Annual Network and Distributed System Security Symposium (NDSS 2020)*. The Internet Society, 2020.

[8] M. Blanton, A. Steele, and M. Alisagari, "Data-Oblivious Graph Algorithms for Secure Computation and Outsourcing," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. Association for Computing Machinery, 2013.

[9] G. Blelloch, "Prefix Sums and Their Applications," in *Synthesis of Parallel Algorithms*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, ch. 1, pp. 35–60.

[10] T. Boelter, R. Poddar, and R. A. Popa, "A Secure One-Roundtrip Index for Range Queries," Cryptology ePrint Archive, Report 2016/568, 2016, https://eprint.iacr.org/2016/568.

[11] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-Preserving Symmetric Encryption," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 224–241.

[12] P. Bose, E. Kranakis, P. Morin, and Y. Tang, "Approximate Range Mode and Range Median Queries," in *STACS 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 377–388.

[13] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-Abuse Attacks Against Searchable Encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. Association for Computing Machinery, 2015, p. 668–679.

[14] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[15] M. Chase and S. Kamara, "Structured Encryption and Controlled Disclosure," in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 577–594.

[16] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and Mobility: User Movement in Location-Based Social Networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2011.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 79–88.

[18] P. Davoodi, R. Raman, and S. R. Satti, "Succinct Representations of Binary Trees for Range Minimum Queries," in *Computing and Combinatorics*, J. Gudmundsson, J. Mestre, and T. Viglas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 396–407.

[19] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical Private Range Search Revisited," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. Association for Computing Machinery, 2016, p. 185–198.

[20] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, M. Garofalakis, and C. Papamanthou, "Practical Private Range Search in Depth," *ACM Trans. Database Syst.*, 2018.

[21] F. B. Durak, T. M. DuBuisson, and D. Cash, "What Else is Revealed by Order-Revealing Encryption?" in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2016.

[22] S. Durocher and R. Singh, "A simple linear-space data structure for constant-time range minimum query," *Theoretical Computer Science*, 2019.

[23] H. El-Zein, M. He, J. I. Munro, Y. Nekrich, and B. Sandlund, "On Approximate Range Mode and Range Selection," in *30th International Symposium on Algorithms and Computation (ISAAC 2019)*, 2019.

[24] Z. Espiritu, "Arca," 2022. [Online]. Available: https://github.com/cloudsecuritygroup/arca

[25] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich Queries on Encrypted Data: Beyond Exact Matches," in *Computer Security – ESORICS 2015*, 2015.

[26] F. Falzon, E. A. Markatou, Akshima, D. Cash, A. Rivkin, J. Stern, and R. Tamassia, "Full Database Reconstruction in Two Dimensions," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2020.

[27] F. Falzon, E. A. Markatou, Z. Espiritu, and R. Tamassia, "Attacks on Encrypted Range Search Schemes in Multiple Dimensions," Cryptology ePrint Archive, Report 2022/090, 2022, https://ia.cr/2022/090.

[28] J. Fischer and V. Heun, "Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE," in *Combinatorial Pattern Matching*, 2006.

[29] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. N. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunning-

ham, “SoK: Cryptographically Protected Database Search,” *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 172–191, 2017.

[30] C. Gentry, “A Fully Homomorphic Encryption Scheme,” Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2009.

[31] M. George, S. Kamara, and T. Moataz, “Structured Encryption and Dynamic Leakage Suppression.” Springer-Verlag, 2021.

[32] M. T. Goodrich, O. Ohrimenko, and R. Tamassia, “Graph Drawing in the Cloud: Privately Visualizing Relational Data Using Small Working Storage,” in *Graph Drawing*, W. Didimo and M. Patrignani, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 43–54.

[33] M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen, “Cell Probe Lower Bounds and Approximations for Range Mode,” in *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 605–616.

[34] P. Grofig, M. Härterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert, “Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data,” *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, 2014.

[35] P. Grubbs, A. Khandelwal, M.-S. Lacharité, L. Brown, L. Li, R. Agarwal, and T. Ristenpart, “Pancake: Frequency Smoothing for Encrypted Data Stores,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 2451–2468.

[36] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, “Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2018.

[37] ——, “Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks,” in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[38] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, “Leakage-Abuse Attacks against Order-Revealing Encryption,” in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[39] Z. Gui, O. Johnson, and B. Warinschi, “Encrypted Databases: New Volume Attacks against Range Queries,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2019.

[40] H. Hacıgümüş, B. Iyer, and S. Mehrotra, “Efficient Execution of Aggregation Queries over Encrypted Relational Databases,” in *Database Systems for Advanced Applications*, 2004.

[41] M. S. Islam, M. Kuzu, and M. Kantarcioglu, “Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation,” in *NDSS*, 2012.

[42] S. Kamara, A. Kati, T. Moataz, T. Schneider, A. Treiber, and M. Yonli, “SoK: Cryptanalysis of encrypted search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data,” in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[43] S. Kamara and T. Moataz, “SQL on Structurally-Encrypted Databases,” in *Advances in Cryptology – ASIACRYPT*, 2018.

[44] S. Kamara, T. Moataz, and O. Ohrimenko, “Structured Encryption and Leakage Suppression,” in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 339–370.

[45] G. Kellaris, G. Kollios, K. Nissim, and A. O’Neill, “Generic Attacks on Secure Outsourced Databases,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[46] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, “The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution,” *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[47] ——, “Response-Hiding Encrypted Ranges: Revisiting Security via Parametrized Leakage-Abuse Attacks,” in *IEEE Symp. on Security and Privacy*, ser. S&P, 2021.

[48] M.-S. Lacharité, B. Minaud, and K. G. Paterson, “Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage,” in *2018 IEEE Symposium on Security and Privacy*, 2018.

[49] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, “Fast and Scalable Range Query Processing With Strong Privacy Protection for Cloud Computing,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2305–2318, 2016.

[50] E. A. Markatou, F. Falzon, R. Tamassia, and W. Schor, “Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2243–2261.

[51] E. A. Markatou and R. Tamassia, “Mitigation Techniques for Attacks on 1-Dimensional Databases That Support Range Queries,” in *Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16–18, 2019, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2019, p. 231–251.

[52] G. Navarro and S. V. Thankachan, “Encodings for Range Majority Queries,” in *Combinatorial Pattern Matching*. Cham: Springer International Publishing, 2014, pp. 262–272.

[53] M. Naveed, S. Kamara, and C. V. Wright, “Inference Attacks on Property-Preserving Encrypted Databases,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2015.

[54] J. Ni, J. Li, and J. McAuley, “Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics, 2019.

[55] S. Oya and F. Kerschbaum, “Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption,” in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 127–142.

[56] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” in *Advances in Cryptology — EUROCRYPT ’99*, 1999.

[57] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin, “Blind Seer: A Scalable Private DBMS,” in *2014 IEEE Symposium on*

*Security and Privacy*, 2014.

[58] H. Petersen, "Improved Bounds for Range Mode and Range Median Queries," in *SOFSEM 2008: Theory and Practice of Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 418–423.

[59] H. Petersen and S. Grabowski, "Range Mode and Range Median Queries in Constant Time and Sub-Quadratic Space," *Inf. Process. Lett.*, vol. 109, no. 4, p. 225–228, Jan. 2009.

[60] R. A. Popa, C. M. S. 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*. Association for Computing Machinery, 2011.

[61] Python Cryptographic Authority, "pyca/cryptography," 2018, version 3.4.7. [Online]. Available: https://cryptography.io/

[62] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing Analytical Queries over Encrypted Data," *Proc. VLDB Endow.*, vol. 6, no. 5, p. 289–300, Mar. 2013.

[63] Vertica, 2021. [Online]. Available: https://www.vertica.com

[64] C. Wang, J. Bater, K. Nayak, and A. Machanavajjhala, "DP-Sync: Hiding Update Patterns in Secure Outsourced Databases with Differential Privacy," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1892–1905.

[65] J. Wang and S. S. M. Chow, "Forward and Backward-Secure Range-Searchable Symmetric Encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 28–48, 2022.

[66] A. C. Yao, "Space-Time Tradeoff for Answering Range Queries (Extended Abstract)," in *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '82. Association for Computing Machinery, 1982, p. 128–136.

[67] Y. Zhang, J. Katz, and C. Papamanthou, "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption," in *25th USENIX Security Symposium (USENIX Security '16)*, 2016.

[68] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic Searchable Symmetric Encryption Schemes Supporting Range Queries with Forward (and Backward) Security," in *Computer Security*, 2018.

| **Scheme** | **Index size** (MB) | **Build time** (s) |
|---|---|---|
| **1/2-ApproxMode** | 219 025.72 | 20 761.48 |
| **$\alpha$-ApproxMedian** | 130 459.63 | 13 603.23 |

**Table 2.** Scheme costs on **Gowalla**, which have effectively constant index size and construction time in the number of records.

| **Scheme** | **Index size** (MB) | **Build time** (s) |
|---|---|---|
| **DPPDGP-Sum** | 1143.54 | 258.01 |
| **DomainBucket** | 2.13 | 1.30 |
| **DataBucket** | 2.01 | 1.11 |
| **DPPDGP-Min1** | 26 301.39 | 9345.69 |
| **DPPDGP-Min2** | 1160.04 | 555.01 |
| **LinearMin** | 3231.74 | 818.05 |
| **1/2-ApproxMode** | 25 157.85 | 7384.72 |
| **$\alpha$-ApproxMedian** | 16 307.45 | 4711.28 |

**Table 3.** Scheme costs on the **Amazon** dataset.

# A Adaptive Semantic Security

**Definition A.1** (Adaptive semantic security [15]). *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Query}, \mathsf{Resolve})$ *be a response-hiding structured encryption scheme for the data structure* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$. *Also, let* $\mathcal{A}$ *be a stateful adversary,* $\mathcal{S}$ *be a simulator,* $\mathcal{L}_\mathsf{S}$ *and* $\mathcal{L}_\mathsf{Q}$ *be leakage functions, and* $z \in \{0,1\}^*$. *Given the following probabilistic experiments:*

$\mathbf{Real}_{\Sigma,\mathcal{A}}(k)$**:** *Given* $z$, $\mathcal{A}$ *outputs a (plaintext) structure* $\mathsf{DS}$. *The challenger executes* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$ *and outputs* $\mathsf{EDS}$ *to* $\mathcal{A}$. $\mathcal{A}$ *then* adaptively *chooses* $m = \mathsf{poly}(k)$ *queries* $Q_1, \ldots, Q_m$. *For each query* $Q_i$, *the challenger executes* $\mathsf{tk}_i \leftarrow \mathsf{Token}(K, Q_i)$ *and outputs token* $\mathsf{tk}_i$ *to* $\mathcal{A}$. *Finally,* $\mathcal{A}$ *returns a bit* $b$ *that is output by the experiment.*

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$**:** *Given* $z$, $\mathcal{A}$ *outputs a (plaintext) structure* $\mathsf{DS}$. *The challenger outputs* $z$ *and the setup leakage* $\mathcal{L}_\mathsf{S}(\mathsf{DS})$ *to the simulator* $\mathcal{S}$. $\mathcal{S}$ *returns an encrypted data structure* $\mathsf{EDS}$ *to* $\mathcal{A}$. $\mathcal{A}$ *then* adaptively *chooses* $m = \mathsf{poly}(k)$ *queries* $Q_1, \ldots, Q_m$. *For each query* $Q_i$, *the challenger gives* $\mathcal{L}_\mathsf{Q}(\mathsf{DS}, Q_i)$ *to* $\mathcal{S}$, *and* $\mathcal{S}$ *outputs a token* $\mathsf{tk}_i$ *to* $\mathcal{A}$. *Finally,* $\mathcal{A}$ *returns a bit* $b$ *that is output by the experiment.*

*We say that* $\Sigma$ *is adaptively* $(\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q})$*-semantically secure if, for all* PPT *adversaries* $\mathcal{A}$, *there exists a* PPT *simulator* $\mathcal{S}$ *where*

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

The definition of adaptive semantic security for interactive STE schemes is identical to Definition A.1 except that in the real experiment, for each query $Q_i$, the challenger $\mathcal{C}$ executes the two-party protocol $\mathsf{Query}_{\mathcal{C},\mathcal{A}}((K, Q_i), \mathsf{EDS})$ with $\mathcal{A}$, then outputs $\perp$ to $\mathcal{A}$. Additionally, in the ideal experiment, $\mathcal{S}$ may output a sequence of tokens to $\mathcal{A}$ for each query $Q_i$.

# B Formalization and Proof of Corollary 4.4

We first define a technical definition of *equivalence* between two plaintext aggregate range query schemes. Our equivalence definition captures when two plaintext aggregate range query schemes output exactly the same *final* aggregate result when given the same database and the same query.

**Definition B.1** (Equivalence). *Let* $\Pi_1 = (\mathbb{S}_1, \mathbb{Q}_1, \mathbb{R}_1)$ *and* $\Pi_2 = (\mathbb{S}_2, \mathbb{Q}_2, \mathbb{R}_2)$ *be two plaintext aggregate range query schemes. We say that* $\Pi_1$ *and* $\Pi_2$ *are* equivalent *plaintext aggregate range query schemes if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size tables* $\mathsf{DB}$*, for all* $\mathsf{poly}(k)$*-size sequences of queries* $Q_1, \ldots, Q_s$*, for all queries* $Q \in Q_1, \ldots, Q_s$*, given*

$$\begin{aligned}\mathsf{DS}_i &\leftarrow \mathbb{S}_i(\mathsf{DB}),\\ U_i &\leftarrow \mathbb{Q}_i(m, Q),\\ S_{i:0} &\leftarrow \{\mathsf{DS}_i[u] \mid u \in U_i\},\\ (\mathrm{st}_{i:0}, R_{i:0}, U_{i:0}) &\leftarrow \mathbb{R}_i(\perp, m, Q, S_{i:0}),\end{aligned}$$

*and, for all* $j > 0$,

$$\begin{aligned}&(\mathrm{st}_{i:j}, R_{i:j}, U_{i:j})\\ &\quad \leftarrow \begin{cases}\mathbb{R}(\mathrm{st}_{i:j-1}, m, Q, S_{i:j-1}) & \text{if } U_{i:j-1} \neq \perp\\ (\perp, R_{i:j-1}, \perp) & \text{if } U_{i:j-1} = \perp\end{cases}\end{aligned}$$

*then there exists an integer* $k \geq 0$ *where* $R_{0:j} = R_{1:j}$ *for* $j \geq k$.

We can then show that if a plaintext aggregate range query scheme is data independent, there exists an equivalent plaintext aggregate range query scheme that requires only one round of queries to the aggregate index structure.

**Corollary B.2.** *Let* $\Pi = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ *be a plaintext aggregate range query scheme. If* $\Pi$ *is data independent, then there exists a scheme* $\Pi' = (\mathbb{S}', \mathbb{Q}', \mathbb{R}')$ *such that* $\Pi$ *is equivalent to* $\Pi'$ *and* $\mathbb{R}'$ *always outputs* $(\mathrm{st}, R, U)$ *where* $\mathrm{st} = \perp$ *and* $U = \perp$.

Due to space restrictions, we defer the full proof of Corollary B.2 to the full version.

# C Query Type Transformations

In this section, we briefly describe how to transform certain aggregate range query schemes into another scheme that can answer a different type of aggregate query. Due to space restrictions, we omit the $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ formalization and analysis of these transformations and defer those definitions to the full version.

*Transformations from range sum query.* Aggregate range query schemes for the encrypted range sum query problem may be used to answer *count*, *average*, and *variance* queries with a constant factor increase in storage, bandwidth, and time.

- *Count.* We add a “fake” attribute to each record with value 1, and apply the range sum query technique over this new attribute.
- *Average.* The client divides the result of a sum query over the desired range by the result of a count query over the same range.
- *Variance.* We add a new attribute to each record that holds the *square* of the desired query attribute, apply the range sum query technique over this new attribute, and then answer queries by taking the result of a sum query over the square attribute and subtracting the square of the result of an average query over the non-squared attribute.

*Transformations from range minimum query.* Aggregate range query schemes for the encrypted range minimum query problem may be used to answer *maximum*, *bottom-k*, and *top-k* queries.

- *Maximum.* We negate the value in the aggregate attribute and apply the range minimum query technique over the negated attribute. When receiving the result of a query, the client negates the returned value to return it to its original sign.
- *Bottom-k.* The technique we use is a generalization of the technique presented by Demertzis et al. [20]. Instead of creating one aggregate structure, we create $k$ aggregate structures, where the $i$th structure stores the element of rank-$i$ (e.g., the first structure stores the true minimum for each preprocessed range, the second structure stores the value above the minimum for each preprocessed range, etc.). Additionally, each minimum is stored as a tuple, where the first element is the minimum value and the second element is an identifier that uniquely identifies the record that this minimum was associated with. During queries, the client generates $O(k)$ search tokens for all of the $k$

structures, and $O(k)$ bandwidth is sent back to the client in response. The client then sorts the returned tokens, removes duplicate record IDs, and picks the lowest $k$ values.

- *Top-k.* Follows from a combination of the maximum and bottom-$k$ transformations.
- *Range.* We use a minimum and maximum structure, which gives us the difference between the maximum and minimum.

# D Extensions

## D.1 Higher Dimensions

The majority of the aggregate query schemes discussed in this paper easily generalize to $d$-dimensional databases of domain size $m_1 \times \cdots \times m_d$.

- **Sum:** For DPPDGP-Sum, a $d$-dimensional index can be created by using the $d$-dimensional prefix sums technique of [9], where an $d$-dimensional array of size $m_1 \times \cdots \times m_d$ is generated, and the index $(c_1, \ldots, c_d)$ stores the sum of all of the records in the $d$-dimensional hypercube defined by $(0, \ldots, 0)$ and $(c_1, \ldots, c_d)$. This requires the client to access $2^d$ entries in the structure.
- **Minimum and Mode:** For DPPDGP-Min1, DPPDGP-Min2, and LinearMin, we can construct $d$ indices, one for each dimension, and then constructing a new index where each entry stores the minimum of the "product" of the ranges represented by each entry of the $d$ dimension-specific indices. This requires the client to access $2^d$ entries in the final structure. (The same technique applies to the ApproxMode schemes due to the similarities between the ST and AS structures.)

These generalizations do not work for the $\alpha$-ApproxMedian and domain reduction schemes. The "bucketing" approach of both the median and the domain reduction schemes does not easily extend to higher dimensions since all such schemes operate by accessing a single entry in the encrypted structure. One may approximate these constructions using a *single range cover*-like approach used by Demertzis et al. [20] in their one-dimensional range search schemes; we leave further exploration of such extensions to future work.

## D.2 Record-Reporting

Applications that desire *record-reporting* for aggregate functions (specifically min/max and median) can instead (or additionally) encode the record identifier associated with the given plaintext value. For example, an encrypted search engine may be interested in identifying the top-$k$ documents with the highest word count that were generated within a particular date range. One can implement this via a combination of the query transformations detailed in Section C to reduce the top-$k$ problem to the minimum scheme; then encoding the record identifier of the document alongside each precomputed sub-aggregate in the structure. (To retrieve the actual record, a separate, $O(n)$-size index can be stored alongside the aggregate structure to map the record ID to the actual record.)

## D.3 Updates

We handle updates by combining a *client-side update cache* (e.g., [64]) with periodic intermediate Setup and "rebuild" operations. At a high-level, we initialize the cache to have a size of $\delta_1$ (a tunable parameter). Then, when the client wants to add or change a record, they add the change to their client-side cache without propagating it to the server immediately. On subsequent range queries, the client is responsible for performing a linear scan of the cache to detect if any of the updates fall within the queried range and adjusting the computed aggregate accordingly. Once the cache size reaches $\delta_1$, the client triggers a Setup operation where they generate a new encrypted structure *over only the records in the cache* and then send the new structure to the server, which accumulates multiple such *update structures* over time. The client stores the keys for the new structure alongside the previously stored keys and empties the cache. On subsequent queries, the client generates multiple search tokens for both structures. To avoid a continual increase in the number of stored structures and search tokens, the client periodically *rebuilds* the server-side structure when the number of update structures reaches $\delta_2$ by constructing a new, single index based off of queries to the existing structures on the server. The client then sends the new structure to the server, which then deletes the previous structures and replaces it with the new index.