Time- and Space-Efficient Aggregate Range Queries on Encrypted Databases

Zachary Espiritu*
Brown University

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science, *with honors*

Advisor: Roberto Tamassia
Reader: Vasileios P. Kemerlis

Abstract

Much of the recent *searchable symmetric encryption* (SSE) and *structured encryption* (STE) literature has centered on the development of *encrypted range structures.* Such structures solve the traditional problem where the client queries the server with a range predicate, and the server responds with the *set* of records satisfying the given predicate. However, users are often not directly interested in the records within the queried range, but rather in the result of an *aggregate function* applied to some attribute of the records. In these cases, the bandwidth, client-side computation, and potentially unnecessary leakage incurred from reporting the whole range makes such schemes problematic to use in settings that necessitate high aggregate computation performance.

In this thesis, we tackle this challenge by developing **ARQ**, a systematic framework for creating cryptographic schemes that handle range aggregate queries (sum, minimum, median, and mode) over encrypted datasets. Our schemes do not rely on trusted hardware or specialized cryptographic primitives such as order-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. Our techniques yield more space-efficient encrypted structures for sum and minimum queries than previous structures presented by Demertzis et al. (*ACM Trans. Database Syst. '18*); they also yield the first encrypted range query schemes for approximate median and mode queries. 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 the **ARQ** framework in comparison to prior work.[1]

*`zespirit@cs.brown.edu`

[1] This thesis contains joint work with Evangelia Anna Markatou and Roberto Tamassia [EMT22] which is currently under double-blind review.

Contents

Thesis statement: *Simple* combinations of data structures previously developed in the plaintext data management community with structured encryption primitives yields efficient (and secure) constructions for privacy-preserving databases.

1 Introduction

Many computer applications today are outsourced to third-party providers via a deployment paradigm known as *cloud computing*. Under this model, a client outsources parts of (or all of) their application's storage and computation to a third-party service provider. This strategy has immense benefits for the client—it alleviates the burden of maintaining one's own infrastructure and hardware, and it also allows the client to take advantage of the economies of scale enjoyed by the service provider. However, this model places a significant amount of trust in the service provider. In particular, the client must trust the server to not snoop or tamper with the application's data. This assumption can be problematic for applications that utilize highly sensitive datasets such as medical data, government records, or data requiring high regulatory oversight (i.e. due to FERPA, HIPPA, PCI DSS, etc.). Such concerns have spurred the development of *privacy-preserving databases* that reduce the trust assumptions involved in outsourcing control to a third-party.

One such class of techniques known as *structured encryption* (STE) [CGKO06; CK11] allows a client to outsource (and later, query) an encrypted version of their data to a semi-honest server. Recent research has focused on improving the practicality of STE by developing specialized structures that efficiently handle more expressive types of queries, such as *range queries*. For example, the *BlindSeer* construction from [Pap+14], the garbled-circuit-based construction from [BPP16], and the range tree constructions from [DPPDG16; DPPDGP18; LLWB16; FJKNRS15; ZSLSP18; WC19; FMET22] are all examples of *encrypted range query structures*. In these works, range queries are effectively presented as a simple *filter* operation, where a predicate is applied to a single database attribute and the *set* of records matching the predicate is returned.

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 aggregates over encrypted data. Using record-reporting STE schemes to answer aggregate queries over encrypted data incurs a significant overhead over traditional "plaintext" data management approaches. 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

Table 1: Our contributions compared to Demertzis et al. (DPPDGP) [DPPDGP18]. Asymptotics are in terms of big-O, where m denotes the size of the domain and n is the number of records, and $0 < \alpha < 1$ is a tunable parameter. "Query" refers to query handling runtime at the server and the client independently. We denote the main tradeoff between our analyzed schemes—whether the schemes are *data-oblivious* ("DO") (Definition 3.4) or optimize 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 [DPPDGP18]	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^α	$\frac{n}{m^\alpha}$		•
Minimum	DPPDGP-Min1 [DPPDGP18]	$m \log m$	1	1	1	1	1	•	
	DPPDGP-Min2 [DPPDGP18]	$m + n \log n$	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	α-**ApproxMedian**	$\frac{m}{1-\alpha}$	1	1	1	1	1	•	

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 and store the answer to all possible range queries in an encrypted dictionary. This naive approach achieves constant-size bandwidth and query time, but prohibitively requires quadratic storage.

Many approaches have been proposed to bridge this performance gap for encrypted databases. *Fully homomorphic encryption* (FHE) [Gen09], 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 applications. As an alternative, *additively homomorphic encryption* (AHE) [Pai99] 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 (e.g., [FJKNRS15; DPPDG16; DPPDGP18; FMET22]) may be used to achieve a balance between both naive approaches by storing precomputed *sub-aggregates* within the structure. Then, aggregate queries may be answered by returning a poly-logarithmic subset of sub-aggregates which the user can process 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 underlying 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 to respond to multiple queries) that can eventually lead to data reconstruction attacks [BKM19]. All in all, the performance and security limitations of existing approaches for handling encrypted range aggregate queries highlights the need for new constructions.

Contributions. We defend the following thesis statement:

> *Simple* combinations of data structures previously developed in the plaintext data management community with structured encryption primitives yields efficient (and secure) constructions for privacy-preserving databases.

To argue our statement, we develop the following contributions:

- We introduce a **general framework** for building encrypted aggregate range query schemes, **ARQ**, 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 any of our schemes that satisfy this property provided that the query distribution is independent of the data distribution. (Sections 3, 4)
- Using **ARQ**, we propose **novel schemes for encrypted range minimum, approximate range mode, and approximate range median**. Our minimum scheme improves the previously-best-known $O(m+n\log n)$ storage overhead of Demertzis et al.'s 2-round protocol [DPPDGP18] to a $O(m)$ 1-round protocol at the cost of prohibiting some *small* queries. To our knowledge, our approximate mode and median schemes are the **first** schemes considering these problems in the STE literature and allow for constant-time and constant-size queries. (Sections 5, 6, 7)
- We propose a series of ***domain reductions*** which can be applied to any **ARQ** scheme to optimize its storage overhead for *sparse* databases in exchange for small performance and leakage tradeoffs. Our reduction techniques can significantly improve the performance of the sum and minimum constructions by Demertzis et al. [DPPDGP18] on sparse databases. (Section 8)
- 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 in terms of storage, query time, and bandwidth. (Section 9)

Finally, we discuss extensions to this work such as generalizing the schemes to multiple dimensions and handling updates. Table 1 compares our schemes with prior work. Our work showcases how cryptographers can leverage advances in databases and data structures to develop secure constructions for aggregate range queries.

2 Related Work

Encrypted range aggregate queries in STE. Demertzis et al. [DPPDGP18] introduced the first study of specific STE structures for encrypted range aggregate queries, proposing encrypted schemes for *range sum queries* (RSQ) and *range minimum queries* (RMQ). Their range sum scheme is based upon the classic *prefix sums* technique [Ble93] and achieves constant query size and time. However, the scheme incurs linear server-side storage in the size of the *domain* (not the number of records), which can be prohibitive for *sparse* databases, such as high-precision geospatial databases that operate over latitude and longitude.

For the range minimum problem, Demertzis et al. proposed two schemes which were based upon the *sparse table* technique by Bender et al. [BFPSS05]. Like the RSQ scheme, both RMQ schemes achieve constant client-side storage, constant bandwidth queries, and constant time queries. The first RMQ scheme incurs linearithmic storage in the size of the domain. The second RMQ scheme incurs a smaller $O(m + n \log n)$ storage requirement, where m is the size of the domain and n is the number of records, at the expense of an additional round trip query. However, even with the storage optimization of the second scheme, the $O(m)$ factor in the storage asymptotics still may be prohibitive for sparse databases. To our knowledge, we propose the first STE schemes for encrypted median and mode queries.

Encrypted SQL. Many works have been proposed to support aggregate functions over encrypted SQL databases via various forms of homomorphic encryption and property preserving encryption. A notable example is Popa et al.'s CryptDB, the first system to support standard SQL operations over encrypted data [PRZB11]. CryptDB uses several specialized encryption schemes such as order-preserving encryption (OPE) tailored to each of the SQL operators; of interest to this work in particular is their use of *additively homomorphic encryption* (AHE) to support aggregation operations [Pai99]. Applications of AHE for encrypted sum aggregates have appeared in many other "secure" database works [Ara+13; Gro+14; HIM04; KM18; TKMZ13]. However, as previously mentioned, AHE does not support aggregate queries that cannot be computed additively. As such, many of these works do not support fundamental aggregate queries such as minimum, median, and mode. Additionally, while OPE may be used to derive simple solutions to rank-based range aggregates (e.g., median queries), many works have demonstrated that OPE leaks enough information to allow for powerful, yet practical data-recovery attacks [NKW15; DDC16; GSBNR17; BGCRS18].

Other encrypted database works rely on *trusted hardware* to answer more types of aggregates. For instance, the stronger security modes of Arasu et al.'s Cipherbase supports aggregations using custom, trusted field-programmable gate arrays (FPGAs) [Ara+13]. Cipherbase computes sum aggregates over columns encrypted with AES by sending the 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 enclaves* such as Intel SGX. However, the security guarantees of such hardware are complex and often such systems are memory-limited. Furthermore, in the specific case of Intel SGX, significant leakage attacks have been discovered against SGX (e.g., [SKGY20]) that have led to the deprecation of SGX on certain Intel processors [Int21]. In this work, we do not rely on specialized hardware or hardware security assumptions.

Leakage-abuse attacks on encrypted range structures. An important property of a structured encryption scheme is its *leakage*, or what information is leaked to the adversary when the scheme's operations are invoked. Islam et al. introduced the first study of leakage-abuse attacks in the searchable encryption setting [IKK12]. In their work, the authors demonstrated how to perform *query-recovery* attacks by exploiting *access pattern* leakage, where the adversary detects when two queries access the same (encrypted) file identifier. Follow-up work such as Cash et al. [CGPR15] and Zhang et al. [ZKP16] exploited access pattern leakage to launch query-recovery attacks with similar impact under different assumptions. Recent works have also demonstrated how to combine access *and* search pattern leakage to perform query-recovery [OK21].

Range queries, in particular, may allow for more severe *data-recovery* attacks. Kellaris et al. [KKNO16] were the first to develop generic reconstruction attacks against one-dimensional range query schemes. Many follow-up works demonstrated more powerful one-dimensional attacks under varying assumptions (see. e.g., [GLMP18; LMP18; GLMP19; GJW19; KPT20; KPT21]), and a recent line of work has yielded reconstruction attacks in higher dimensions [Fal+20; MFST20; FMET22].

3 Preliminaries

Notation. $\{0,1\}^\ell$ denotes the set of all binary strings of length ℓ. $\{0,1\}^*$ 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 make use of several basic data structures whose notation and syntax we define here. A *dictionary* 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 ℓ_i. $\mathsf{DX}[\ell_i] := v$ denotes the assignment of value v_i to label ℓ_i. A *multimap* 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 ℓ_i. $\mathsf{MM}[\ell_i] := \mathbf{t}_i$ denotes the assignment of tuple $\mathbf{t}_i$ to label ℓ_i. The length of each tuple $\mathbf{t}_i$ may vary within the multimap.

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

Tables. We use *tables* as the main input to our STE schemes. A *table* DB is a two-dimensional array where each row is a *record* and each column is an *attribute* represented by a unique positive integer. We assume that each attribute has a finite *domain* of possible values. Every record $\mathbf{r} \in \mathsf{DB}$ is a *tuple* indexed by the integers representing each of the attributes in DB. 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 or equal to, but not less than, x_i. Finally, we use n to refer to the number of records in a table 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 DS (e.g., dictionary) so that a *client* can outsource an encrypted form of the structure to an untrusted *server* and privately query it using a secret key K. We adopt our STE definitions from the work of Chase and Kamara [CK11], which generalized the SSE definitions of Curtmola et al. [CGKO06].

An important distinction is between *response-revealing* STE schemes, where the response to the query is revealed to the server in plaintext (but not directly the query itself), and *response-hiding* schemes, where the response to the query is not revealed to the server. (Note that [CK11] refers to these two characteristics as *ciphertext-output* schemes and *plaintext-output* schemes, though we use the more commonly used "response-revealing" / "response-hiding" terminology in this work.) In this work, we exclusively refer to *response-hiding STE schemes*, defined below.

Definition 3.1 (Response-hiding STE [CK11])**.** *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* DS*. It then outputs a key* K *and an encrypted data structure* 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* tk*.*

- $\mathrm{ct} \leftarrow \mathsf{Query}(\mathsf{EDS}, \mathsf{tk})$ *is a deterministic algorithm run by the server to respond to queries. It takes as input the encrypted data structure* EDS *and a query token* tk *and outputs a response* 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* ct *and outputs a plaintext response* R*.*

A response-hiding STE scheme Σ *is correct if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$*, for all* $\mathsf{poly}(k)$*-size sequences of queries* $Q_1, \ldots, Q_s$ *where* $Q_i \in \mathbf{Q}$*, for all* 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, which can be queried by sending the server a single search token. Some STE protocols are *interactive*, which means that executing a query requires the server and the client to engage in a two-party protocol involving 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* Setup *is the same as it was in Definition 3.1 and* 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.*

An interactive STE scheme Σ *is correct if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size structures* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$*, 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 Σ, we define *leakage functions* that precisely define what information is leaked to the adversary by the different operations of Σ: $\mathcal{L}_{\mathsf{S}}$, the *setup leakage*, or what is leaked by the Setup operation, and $\mathcal{L}_{\mathsf{Q}}$, the *query leakage*, or what is leaked by the Query algorithm. We show that no information beyond these functions is leaked by proving that an 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 $\mathcal{L}_{\mathsf{Q}}$.

In this work, the adversary is *semi-honest*, meaning that the adversary does not deviate from the execution of the protocol, and is *adaptive*, meaning that the adversary may choose query operations at will and can attempt to learn information from or modify their query strategy based on the query transcript.

Definition 3.3 (Adaptive semantic security [CK11]). *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 setup and query 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* DS*. The challenger executes* $(K, \mathsf{EDS}) \leftarrow \mathsf{Setup}(1^k, \mathsf{DS})$ *and outputs* 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* 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* 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* 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* tk_i *to* $\mathcal{A}$*. Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

We say that Σ 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*

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

The definition of adaptive semantic security for interactive STE schemes is identical to Definition 3.3 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.

Data-oblivious structure (DO). Throughout our work, we observe 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 thus formulate a definition tailored to the encrypted aggregate range problem that we call *data-obliviousness* (DO). This property is similar to the standard definition used in the the *data-oblivious algorithm* literature (e.g., [BSA13; GOT12]). 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 3.4 (Data-oblivious STE structure)**.** *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 Σ 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 3.4 except $\Sigma = (\mathsf{Setup}, \mathsf{Query})$. The DO property guarantees that data reconstruction attacks are impossible if the queries to Σ are independent of the underlying data distribution (a standard assumption in the reconstruction attacks literature). This is because the information leaked by Σ is solely a function of the size of the encrypted data structure and the queries.

4 ARQ: A General Framework

In addition to defining new, secure schemes for encrypted databases, another 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 be combined with any standard STE scheme to 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 that we analyze in this work, including those previously analyzed by Demertzis et al. [DPPDGP18], fit into this syntax.

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, EDS);
- $R \leftarrow \mathsf{Query}_{\mathbf{C},\mathbf{S}}(K, Q)$:
  1. $\mathbf{C}$ sets 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 stk_i to server;
     (b) for all stk_i,
        i. $\mathbf{S}$ computes $\mathrm{ct}_i \leftarrow \Sigma.\mathsf{Query}(\mathsf{stk}_i, \mathsf{EDS})$;
        ii. $\mathbf{S}$ sends ct_i to client;
     (c) initialize set S;
     (d) for all 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;

Figure 1: The **ARQ** framework.

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

- $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$ *takes as input a database* 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* DS*.*
- $(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)$ *aggregates a set of responses from* DS*. It takes as input prior state* 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* st'*, an aggregate R (which may be $\perp$), and a set of additional subqueries to be issued U (which may be $\perp$).*

We note that $\mathbb{R}$ takes Q as input since the method for aggregating subresults may differ based on the initial query. Additionally, $\mathbb{R}$ is stateful in order to support interactive protocols requiring multiple rounds of communication.

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 an interactive STE scheme parameterized by an aggregate range query scheme $(\mathbb{S}, \mathbb{Q}, \mathbb{R})$ and a response-hiding STE scheme Σ on data structure $\mathbb{S}(\mathsf{DB})$, and the full framework is defined in Figure 1. (For brevity, we refer to $\mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$ as $\mathbf{ARQ}$ in the paper when the parameters are implied from context.) In $\mathbf{ARQ}$, the client converts their database DB into a aggregate query index $\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})$. Then, $\Sigma.\mathsf{Setup}$ is used to encrypt 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.

Theorem 4.2. *If $(\mathbb{S},\mathbb{Q},\mathbb{R})$ is an aggregate range query scheme, and Σ 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*

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

and

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

Here, Λ 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 Σ and consider the $\mathbf{ARQ}$ simulator $\mathcal{S}$ that works as follows. Given $\mathcal{L}_{\mathsf{S}}(\mathsf{DB})$, $\mathcal{S}$ simulates the encrypted index 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 $\mathcal{A}$,

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

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 Σ's simulator $\mathcal{S}^{\Sigma}$. Thus, in order for the adaptive semantic security of Σ to hold, $\Pr_{break}$ must be negligible. □

4.1 Data Independence

Motivated by the strong guarantees of our DO property (Definition 3.4), we would like to identify properties of aggregate range query schemes and STE schemes that result in an instantiation of $\mathbf{ARQ}$ that satisfies the DO requirements. To do this, 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 DB of the same size and the subqueries generated by $\mathbb{Q}$ and $\mathbb{R}$ do not depend on the original table 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* Π *be an interactive plaintext aggregate range query scheme. If* Π *satisfies data independence,* Π *may be converted into a non-interactive scheme with identical functionality.*

We provide a formal definition and proof of this property in Appendix A. The contrapositive of Corollary 4.4 provides an important insight—namely, that plaintext aggregate range query schemes that cannot be converted into a non-interactive, equivalent scheme will not satisfy data independence. As we describe later in this work, this observation allows us to prune several plaintext aggregate range query schemes in our search for plaintext schemes that can produce DO-satisfying, **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 **ARQ** framework and a suitable choice of Σ 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 Σ 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* DS *with query space* $\mathbf{U}$ *and response space* $\mathbf{S}$ *given some input database* DB. *Also, let* Σ *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. First, we observe that the DO property of Σ implies there exist functions f and g such that $\mathcal{L}^{\Sigma}_{\mathsf{S}}(\mathsf{DS}) = f(|\mathsf{DS}|)$, $\mathcal{L}^{\Sigma}_{\mathsf{Q}}(\mathsf{DS}, Q) = g(|\mathsf{DS}|, Q)$, and Σ is adaptively $(\mathcal{L}^{\Sigma}_{\mathsf{S}}, \mathcal{L}^{\Sigma}_{\mathsf{Q}})$-secure. We now need to 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 $(\mathcal{L}^{\Sigma}_{\mathsf{S}}, \mathcal{L}^{\Sigma}_{\mathsf{Q}})$-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, $\mathcal{L}_{\mathsf{S}}(\mathsf{DB}) = \mathcal{L}^{\Sigma}_{\mathsf{S}}(\mathsf{DS}) = f(|\mathsf{DS}|) = f(f'(|\mathsf{DB}|))$. 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 Σ, $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = (g(|\mathsf{DS}|, Q))_{q \in \mathbf{U}}$. 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 DS for a given query Q regardless of the underlying contents of DB; in other words, the Λ term of the leakage is a function of Q and nothing else. Thus, we know that there must exist a function j such that $\mathcal{L}_{\mathsf{Q}}(\mathsf{DB}, Q) = (g(f'(|\mathsf{DB}|), Q))_{q \in \Lambda} = j(|\mathsf{DB}|, Q)$. □

In the following sections, we 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 Π_{bas} from [Cas+14] (Theorem 4.5).

In each section, we also perform a complexity analysis on concrete instantiations of **ARQ**. Since **ARQ** requires a concrete Σ, our complexity analyses assume that Σ requires storage $O(|\mathsf{DS}|)$ and each query requires constant time and space in the size of DS. An example of such a Σ that satisfies these complexity assumptions and the leakage profile constraints in Theorem 4.5 is Π_{bas} from [Cas+14]. While we use Π_{bas} for our complexity analysis, we emphasize that different choices of Σ 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 $0 \leq i \leq j < n$, the* range minimum query *returns the smallest element of A in the range $[i, j]$.*

Previous results. Demertzis et al. [DPPDGP18] 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. [BFPSS05]. 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 with an encrypted array scheme 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 as the combinations of access patterns between the first encrypted index and the second encrypted index differ when the coordinate of points are changed in the input database. Additionally, the scheme used by DPPDGP-Min2 is interactive and is thus not data independent as implied by Corollary 4.4.

Survey of existing plaintext structures. Several $O(m)$-space solutions to the range minimum query problem 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 [DRS12; DS19] 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 [BF00; BFPSS05; FH06] 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 [FH06] (FH) in such a way that the resulting *plaintext* scheme does *not* require bit-packing techniques, 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*. (We propose workarounds for the query size limitations in Appendix B.) We then show how our modifications allow us to easily derive a provably DO encrypted range minimum query 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 DB into blocks $B_1, \ldots, B_{m/s}$ of size $s = \log m$. Then, an array 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 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:

1. From ℓ to the end of ℓ's block using the normalization table to find the index of the minimum, then using A to retrieve the actual minimum.

2. The range spanning all blocks between ℓ's block and r's block using M (does not require accessing A).

3. From the beginning of r's block to r using the normalization table, then using 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 $\Pi_{\text{LinearMin}}$, our modified FH scheme. Figure 2 outlines the algorithm for the scheme, and Figures 3 and 4 provide 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.[2] This allows us to avoid using the lookup tables from the original FH scheme. However, with a simple modification, we can still answer arbitrary queries of size at least s. To do this, we

[2]As an example, a database of size $m = 2^{64}$ has block size $s = 64$, and so some (but not all) queries of size smaller than 64 are not permissible with this scheme.

Let $\texttt{SparseTable}(\mathsf{A})$ denote an application of the ST algorithm [BFPSS05] 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 A, $\mathbb{L}_{\mathsf{left}}$, and $\mathbb{L}_{\mathsf{right}}$;
  2. compute $s = \log m/4$;
  3. partition 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 $M \leftarrow \texttt{SparseTable}(\mathsf{A})$;
  6. for $i \in [m]$,
     (a) compute block index b of index i;
     (b) set $\mathbb{L}_{\mathsf{left}}[i] := \min_{sb \leq k \leq i} \mathsf{DB}[k]$;
     (c) set $\mathbb{L}_{\mathsf{right}}[i] := \min_{i \leq k \leq s(b+1)-1} \mathsf{DB}[k]$;
  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 $b_\ell \leftarrow \left\lceil \frac{\ell}{s} \right\rceil$ and $b_r \leftarrow \left\lfloor \frac{r}{s} \right\rfloor$;
  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 $q_3 \leftarrow (b_\ell, h)$;
     (c) compute $q_4 \leftarrow (b_r - 2^h + 1, b_\ell)$;
     (d) output (q_1, q_2, q_3, q_4);
  6. else,
     (a) output $U \leftarrow (q_1, q_2)$;
- $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
  1. output $(\perp, \min(S), \perp)$;

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

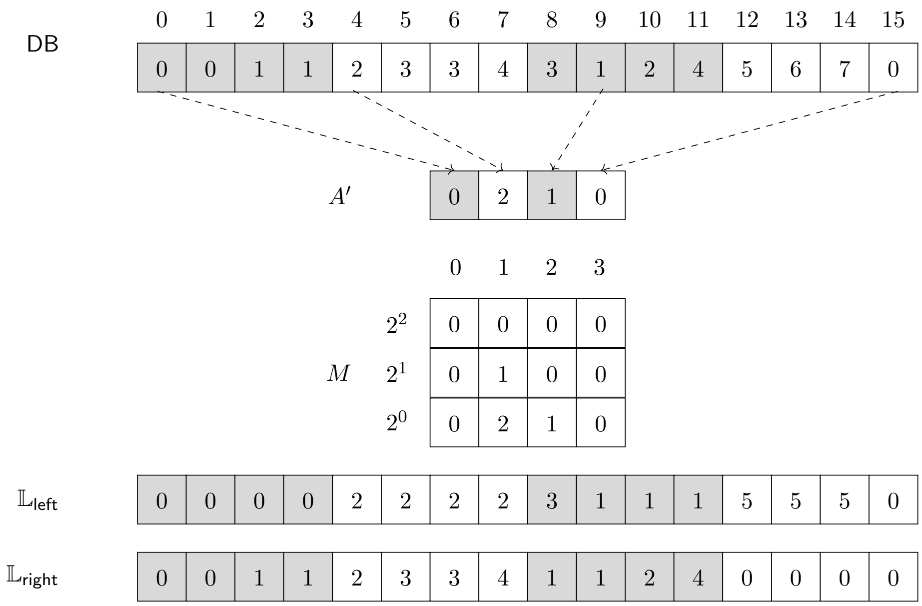

Figure 3: Minimum scheme on database of size $m = 16$.

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 ℓ to the end of ℓ's block using $\mathbb{L}_{\mathsf{right}}$.

2. The range spanning all blocks between ℓ'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 query of size s or greater using the precomputed structures defined above and 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, $\textsc{LinearMin} = \mathbf{ARQ}_{\Sigma,\mathbb{S},\mathbb{Q},\mathbb{R}}$, 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 DB_0 or 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 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 $j > 0$. $\square$

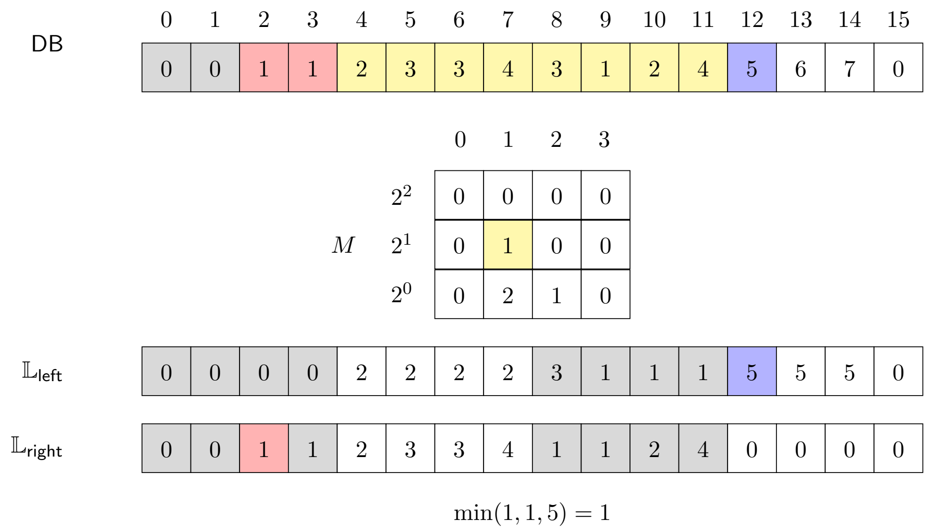

Figure 4: Example of how to answer the minimum query $[2, 12]$ using the structure in Figure 3.

Theorem 5.3. *Given the plaintext minimum aggregate range query scheme $\Pi_{\text{LINEARMIN}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$, and a response-hiding STE scheme Σ on data structure* 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 *and requires $O(m)$ storage, $O(1)$ query time at both the client and server, and $O(1)$ bandwidth in the size of the database domain m.*

Theorem 5.3 follows by Lemma 5.2 and from a similar argument to that of Fischer and Heun [FH06].

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 $O(n^2 \log\log n / \log^2 n)$ [PG09]. Given the near-quadratic storage requirement of exact range mode query schemes, we instead consider an *approximate* version of the range mode query problem. Such approximations are available in several database implementations to improve performance (e.g., the `APPROXIMATE_MEDIAN` and `APPROXIMATE_MODE` functions in Vertica [Foc21]). Introduced by Bose et al. [BKMT05], 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 (α-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 α-*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 α-*approximate range mode query *returns an α-approximate mode of A'.*

Observe that the accuracy of the approximation increases as α tends to 1.

Let $\texttt{BKMTApproxMin}(k, \mathsf{DB})$ denote an application of the BKMT algorithm with k intervals [BFPSS05] and let $\texttt{IntervalSelect}(\mathsf{A})$ denote an application of the AS interval selection algorithm [AS87] on query Q. 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 $M \leftarrow \texttt{BKMTApproxMin}(\frac{1}{2}, \mathsf{DB})$;
  2. output $\mathsf{DS} \leftarrow (M)$;
- $\underline{U \leftarrow \mathbb{Q}(m, Q)}$:
  1. parse $(\ell, r) \leftarrow x$;
  2. compute $(q_1, q_2) \leftarrow \texttt{IntervalSelect}(\ell, r)$;
  3. output (q_1, q_2);
- $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
  1. compute $(R, f) \leftarrow \arg\max_{(x, f_x) \in S} f_x$;
  2. output $(\perp, R, \perp)$;

Figure 5: The plaintext $\Pi_{1/2\text{-}\textsc{ApproxMode}}$ scheme used to instantiate the encrypted $1/2-$ApproxMode scheme.

Survey of existing plaintext structures. Several solutions have been proposed that deal with the approximate range mode problem with varying degrees of efficiency. Many of these schemes use bit-packing techniques [GJLT10; NT14; EHMNS19] that, as explained previously, we can not directly translate to STE structures. Additionally, while some recent schemes achieve lower storage requirements than our chosen approach, their query algorithms require non-constant query time [EHMNS19]. In all cases we examined, non-constant time queries were due to 1) queries requiring at least 2 round trips to the structure and 2) 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) [BKMT05]. The BKMT scheme relies on the following technical theorem.

Theorem 6.2 (Mode partition [BKMT05])**.** *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 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 mode with the highest frequency is an $1/k$-approximate mode of $[\ell, r]$.*

Yao [Yao82] and Alon and Schieber [AS87] 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.

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

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

Figure 6: The BKMT 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. As an example, consider the query $[1, 5]$. This query falls entirely within the first half of the array, but not entirely within the halves of the first half, so we will retrieve entries from row 2^2. Then, we retrieve the entries from row 2^2 that correspond to our query's endpoints—in this case, $M[2^2][1] = (1, 3)$ and $M[2^2][5] = (2, 1)$. We then pick the tuple whose second element is greater, which gives us $(1, 3)$. The first element of this tuple is the 1/2-approximate mode.

This change allows for the partitioning scheme to be generic for any value of k (as opposed to the ST technique, which is only defined over 2 intervals).

Here, we describe the partitioning algorithm for the case when $k = 2$ to highlight the differences between the AS approach and the ST approach. 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 $h + 1 \leq j \leq n$, 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. This process produces a hierarchical table where each level represents m intervals computed at one of the $O(\log m)$ recursive steps.

The BKMT solution for $1/k$-approximate range mode queries 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 5.

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 the plaintext mode aggregate range query scheme* $\Pi_{1/2\text{-ApproxMode}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$, *and a response-hiding STE scheme* Σ *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 in the size of the database domain m.

Theorem 6.4 follows by Lemma 6.3 and from the analysis in [BKMT05].

1/3-ApproxMode. The description of 1/3-ApproxMode, a scheme for answering encrypted 1/3-approximate range mode queries, is equivalent to that of 1/2-ApproxMode except we instantiate **ARQ** with a modified variant of $\Pi_{1/2\text{-ApproxMode}}$ that uses a 3-interval AS table and queries involve computing three interval covers instead of two. (We direct the reader to [AS87] for the full description of the k-interval partition algorithm for $k > 2$.) This change improves the storage requirement to $O(m \log \log m)$ in exchange for a wider approximation. Given the similarity between 1/3-ApproxMode and 1/2-ApproxMode, we omit the full description of this approach.

7 Range Median Query

In this section, we consider and 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. The current state-of-the-art solution for constant-time range mode queries on static datasets by Petersen requires $O(n^2 \log^{(k)} n / \log n)$ storage, where k is an arbitrary constant and $\log^{(k)}$ is the iterated logarithm function [Pet08]. Thus, just as in our discussion of the range mode query problem, we instead concentrate on solutions for the *approximate* version of the problem introduced by Bose et al. [BKMT05]. Given $0 < \alpha < 1$, the α-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* α-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 α-approximate range median query *returns an α-approximate median of A'.*

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

7.1 Our Approach

The BKMT scheme. We describe the *tunable* α-approximate median query structure from Bose, Kranakis, Morin, and Tang (BKMT) [BKMT05] which we refer to as $\Pi_{\alpha\text{-ApproxMedian}}$. Figure 7 describes the scheme and Figure 8 illustrates an example. The BKMT algorithm relies on the following observation about the median: 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 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 jth block of size $m/2^i$, denoted

Let $\texttt{BKMTApproxMed}(\alpha, \mathsf{DB})$ denote an application of the BKMT α-approximate median algorithm over table DB [BFPSS05]. Consider the aggregate range query scheme $\Pi_{\alpha\text{-ApproxMedian}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ defined as follows:

- $\underline{\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})}$:
  1. initialize dictionary DX;
  2. compute $k \leftarrow \lceil \log m \rceil$ and $p \leftarrow \left\lfloor \frac{2(1+\alpha)}{1-\alpha} \right\rfloor$;
  3. compute $(T_1, \ldots, T_k) \leftarrow \texttt{BKMTApproxMed}(\alpha, \mathsf{DB})$;
  4. for all $1 \leq i \leq k$, for all $1 \leq j \leq 2^i$, for all $1 \leq x \leq p$, set $\mathsf{DX}[\langle i, j, x \rangle] := T_i[j][x]$;
  5. output $\mathsf{DS} \leftarrow (\mathsf{DX})$;
- $\underline{U \leftarrow \mathbb{Q}(m, Q)}$:
  1. parse $(\ell, r) \leftarrow x$;
  2. compute $L \leftarrow r - \ell + 1$;
  3. compute $i \leftarrow \left\lceil \log \frac{(1+\alpha)m}{(1-\alpha)L} \right\rceil$;
  4. compute $b_\ell = \left\lceil \frac{2^i \ell}{m} \right\rceil$ and $b_r = \left\lfloor \frac{2^i r}{m} \right\rfloor$;
  5. output $U \leftarrow (\langle i, b_\ell, b_r - b_\ell \rangle)$;
- $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
  1. output $(\perp, S, \perp)$;

Figure 7: The plaintext $\Pi_{\alpha\text{-ApproxMedian}}$ scheme used to instantiate the encrypted α-ApproxMedian scheme.

$B_{i:j}$. Then, each $T_i[j]$ is a list containing $p = \left\lfloor \frac{2(1+\alpha)}{1-\alpha} \right\rfloor$ elements of DB, where, for all $1 \leq x \leq p$, $T_i[j][x] := \mathsf{median}\left(\bigcup_{0 \leq q \leq x-1} B_{i:(j+q)} \right)$.

Each array T_i contains 2^i lists, each containing $\left\lfloor \frac{2(1+\alpha)}{1-\alpha} \right\rfloor$ 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 $\sum_{i=1}^{\log m} O\left(\frac{2^i(1+\alpha)}{1-\alpha} \right) = O\left(\frac{m(1+\alpha)}{1-\alpha} \right) = O\left(\frac{m}{1-\alpha} \right)$. Answering a query $[\ell, r]$ can be done in $O(1)$ by accessing a single element in T using the following algorithm:

1. Given the length of the query $L = r - \ell + 1$, locate array T_i by computing $i = \left\lceil \log \frac{(1+\alpha)m}{(1-\alpha)L} \right\rceil$.
2. Compute block indices of ℓ and r as $b_\ell = \left\lceil \frac{2^i \ell}{m} \right\rceil$ and $b_r = \left\lfloor \frac{2^i r}{m} \right\rfloor$.
3. Output $T_i[b_\ell][b_r - b_\ell] = \mathsf{median}\left(\bigcup_{0 \leq q \leq b_r} B_{i:(b_\ell+q)} \right)$.

It easily follows that answering queries with the above algorithm takes $O(1)$ time.

α-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, α-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.

DB

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

T_1

	0	1
1	1.5	2.5
2	2	$\perp$

T_2

	0	1	2	3
1	1	3	2.5	2.5
2	1.5	3	2.5	$\perp$

T_3

	0	1	2	3	4	5	6	7
1	0.5	1	2.5	3.5	2	3	2.5	2
2	1	1.5	3	3	2.5	3	4	$\perp$

T_4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	1	1	2	3	3	4	3	1	2	4	4	1	4	0
2	0.5	1	1	1.5	2.5	3	3.5	3.5	2	1.5	3	4	2.5	2.5	2	$\perp$

Figure 8: α-ApproxMedian scheme on DB of size $m = 16$, $\alpha = \frac{1}{8}$.

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 the plaintext median aggregate range query scheme* $\Pi_{\alpha\text{-ApproxMedian}} = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$, *and a response-hiding STE scheme* Σ *on data structure* DS *with storage space* $O(|\mathsf{DS}|)$ *and constant query time and space, the scheme* α-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 in the size of the database domain* m.

Theorem 7.3 follows by Lemma 7.2 and from the analysis in [BKMT05].

8 Exploiting Sparsity via Domain Reductions

We now present several techniques that optimize the space requirements of encrypted aggregate range query structures for *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 act as *compilers* 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 query space. In general, all of 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 domain reductions result in significantly reduced storage, with different *tradeoffs* between query performance and leakage.

Definition 8.1 (Domain reduction)**.** *Given a data structure* $\mathsf{DS} : \mathbf{Q} \to \mathbf{R}$*, a* domain-transforming data structure from $\mathbf{Q}$ to $\mathbf{Q}^*$ *is a data structure* $\mathsf{DS}^* = (\mathsf{DS}_1, \mathsf{DS}_2)$ *where* $\mathsf{DS}_1 : \mathbf{Q} \to \mathbf{Q}^*$ *and* $\mathsf{DS}_2 : \mathbf{Q}^* \to \mathbf{R}$*. If* $|\mathbf{Q}^*| < |\mathbf{Q}|$*, we then say that* DS^* *is a* domain-reducing data structure*.*

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 aggregate range 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.

The DomainBucket reduction. Figure 10 illustrates an example of the DomainBucket reduction, and the algorithm is defined in Figure 9. The reduction takes as input the domain size m, the database DB, and a tunable constant $0 < \alpha < 1$. Given a database DB with domain $\mathcal{D}$, we partition the domain space 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 ith 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 record's original domain value and the ordered ID of the record (i.e., the transformed domain).

We also store two additional records in each bucket that we refer to as the "backwards/forwards lookup" portion of each bucket. The first such record 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 similar, but for the right endpoint of the bucket. This allows the client to receive a valid answer to a query in a *single* round of communication, 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. DomainBucket requires only a constant-size search token corresponding to the desired bucket range of size m^α. 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 domain point.

The DataBucket reduction. The volume leakage and the worst-case-linear bandwidth of the DomainBucket scheme motivates our second domain reduction construction, DataBucket (Figure 11). 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. Each bucket volume is at most one less than the largest bucket volume, and so the remaining buckets are padded with $\perp$ to ensure that the bucket volumes are indistinguishable. This brings the bandwidth complexity in the worst case from $O(m)$ to $O(m^\alpha)$ (i.e. the maximum size of a bucket).

8.1 Example: Range Sum Queries

In this section, we discuss the encrypted range sum query problem. With our domain reduction schemes in hand, we use this section as an example of how our domain reductions can be used to de-

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

- $\underline{\mathsf{DS} \leftarrow \mathbb{S}(\mathsf{DB})}$:
  1. initialize multimap 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 $\mathsf{DS} \leftarrow (\mathsf{MM}, \mathsf{DS}')$;
- $\underline{U \leftarrow \mathbb{Q}(m, Q)}$:
  1. parse $(\ell, r) \leftarrow Q$;
  2. compute $i_\ell \leftarrow \left\lfloor \frac{\ell}{m^{1/\alpha}} \right\rfloor$ and $i_r \leftarrow \left\lfloor \frac{r}{m^{1/\alpha}} \right\rfloor$;
  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 $(\langle L_\ell, R_\ell\rangle, \langle L_r, R_r\rangle)$;
- $\underline{(\mathrm{st}', R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S)}$:
  1. if $\mathrm{st} = \perp$,
     (a) parse $(B_\ell, B_r) \leftarrow S$;
     (b) compute the closest element ℓ to x in bucket B_ℓ 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 (st', R, U);

Figure 9: The domain reduction $\Pi_{\text{DomainBucket}(\mathbb{S}',\mathbb{Q}',\mathbb{R}')}$.

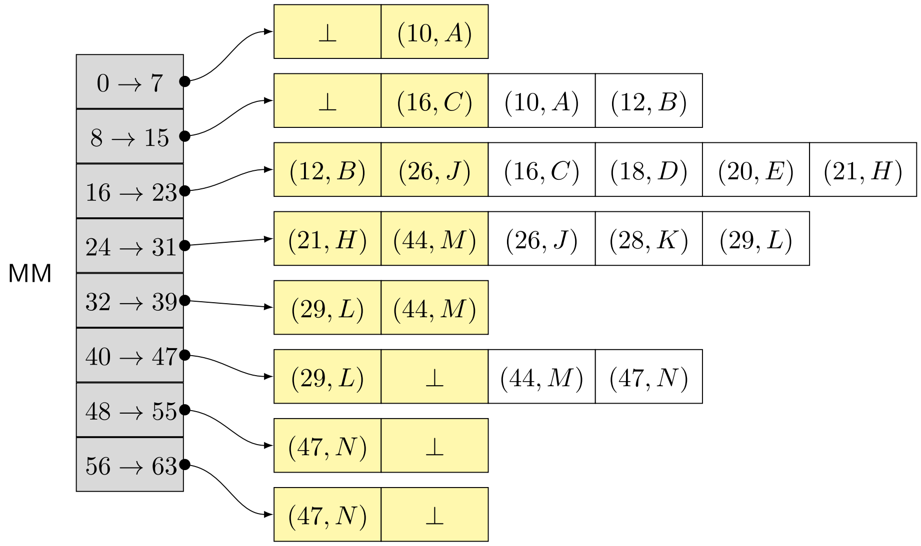

Figure 10: DomainBucket example on a database of size $m = 64$. Highlighted cells represent the lookup portion of each bucket.

rive optimizations to the encrypted range sum scheme presented by Demertzis et al. [DPPDGP18], which require $O(m)$ storage.

Definition 8.2 (Range sum query)**.** *Given an array A of n numbers and indices i and j where $0 \leq i \leq j < n$, the* range sum query *computes*

$$\sum_{k=i}^{j} A[k],$$

the sum of all $A[k]$ where $i \leq k \leq j$.

As explained in prior plaintext database works [Ska13], 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. [DPPDGP18] presented the first and only encrypted range sum query STE scheme, DPPDGP-Sum, which was based on the classic *prefix sums* technique [Ble93]. 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 the prefix sums technique to the encrypted range sum problem by simply encrypting A with an array encryption scheme.

We note that our **ARQ** framework can be used to derive this scheme and its leakage.

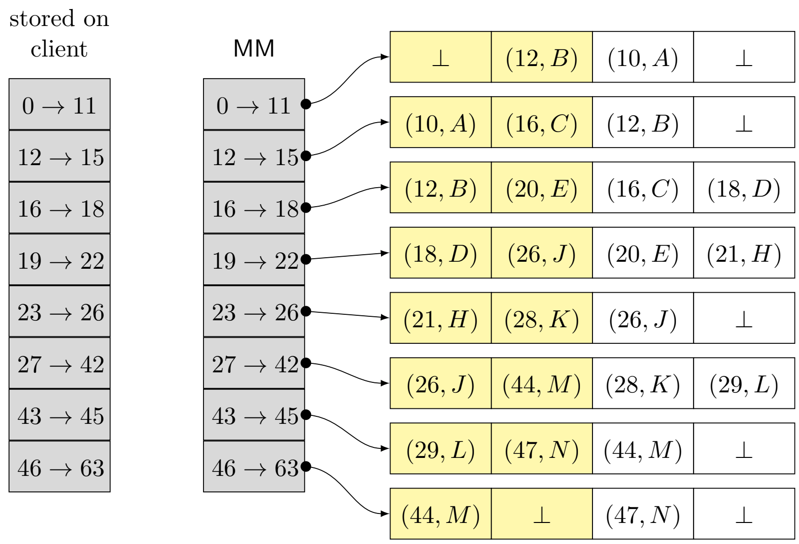

Figure 11: DataBucket example. Highlighted cells represent the lookup portion of each bucket.

Our derived schemes. Our domain reduction schemes 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 over the n sorted record IDs in DB as the domain values. (Observe that the "domain" of this prefix sums array is over the $O(n)$ record IDs of DB.) Then, we apply our choice of domain reduction scheme to the prefix sums array, which maps from the original domain of size m to the record ID domain of size $O(n)$. When the client wants to make a query $[\ell, r]$, they make *two* queries to the domain reduction structure—one to find the nearest record ID id_ℓ on the left of ℓ, and another to find the nearest record ID id_r on the right of r. The client then queries the actual prefix sums array 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.

8.2 Tradeoffs and Attacks

Our domain reductions primarily trade increased leakage in exchange for substantially less storage overhead. In particular, under DataBucket, the client-side storage is significantly smaller than $O(m)$. For example, when given a dataset with domain size $m = 54000000$, the client-side storage is $O(7348)$ for $\alpha = 0.5$.

On the other hand, 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 surrounding this tradeoff, we now describe potential starting points for attacks against the domain reductions and what threat model 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. Definition 8.1 requires that the domain reduction index translates coordinates in the original query space $\mathbf{Q}^*$ into the query space $\mathbf{Q}$ of the aggregate index structure such that $|\mathbf{Q}^*| <$

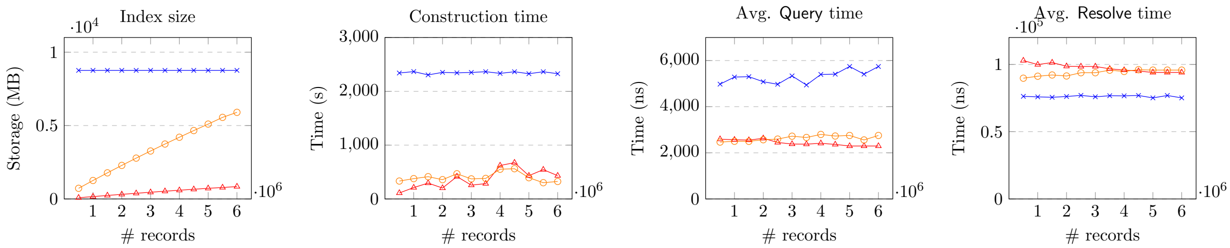

Figure 12: Scheme costs on the **Gowalla** dataset with DPPDGP-Sum [DPPDGP18] (—×—), DomainBucket (—○—), and DataBucket (—△—).

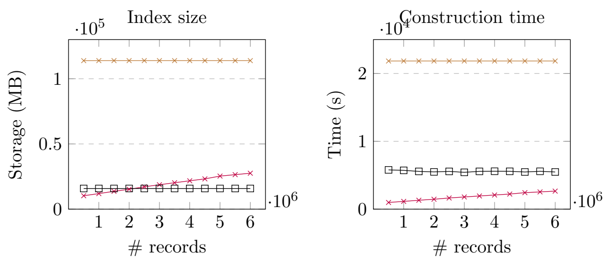

Figure 13: Scheme costs on the **Gowalla** dataset with DPPDGP-Min1 [DPPDGP18] (—×—), DPPDGP-Min2 [DPPDGP18] (—×—), and LinearMin (—□—). There were no significant changes in Query time as the number of records increased (average Query times—DPPDGP-Min1: 1656 ns; DPPDGP-Min2: 2033 ns; LinearMin: 3014 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).

$|\mathbf{Q}|$. By the Pigeonhole Principle, 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 Σ has search pattern leakage, they may count how many (encrypted) queries each (encrypted) bucket receives via the search pattern leakage. Larger numbers of queries issued to the same bucket (in comparison to the number of queries issued to other buckets) implies that the dataset has different density levels, with the buckets holding larger numbers of queries corresponding to a less-dense range in the domain. 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 there are some less-dense areas of the domain than others, the adversary may not be able to determine the ordering of these density levels without additional assumptions on the dataset. Furthermore, attacks that leverage knowledge of the query distribution (e.g., [KPT21; OK21]) can be mitigated using frequency smoothing techniques (e.g., Pancake [Gru+20]). In Section 10, we discuss more mitigations and extensions to our work.

9 Empirical Evaluation

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

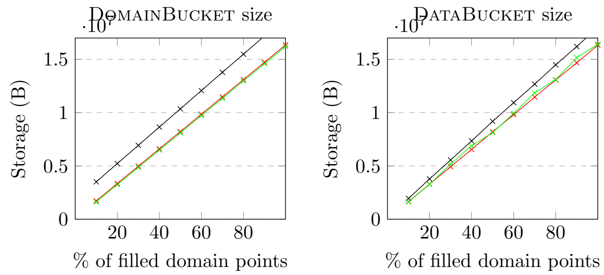

Figure 14: Density evaluation of DomainBucket (left) and DataBucket (right) on synthetic datasets ($m = 100000$) with $\alpha = 0.25$ (—×—), $\alpha = 0.50$ (—×—), and $\alpha = 0.75$ (—×—).

Arca: a new structured encryption library. As part of this work, we designed and implemented a new, open-source, encrypted search library called Arca [Esp22]. Arca is an 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 [Aut18]) and implementations of various structured encryption schemes. Arca has complete type annotations and passes all strict static type checks provided by the MyPy type checker. Arca's code and its associated documentation are available at `https://github.com/cloudsecuritygroup/arca`.

We implemented all of the plaintext schemes that we chose in this work as well as the plaintext schemes used by Demertzis et al. [DPPDGP18]. Then, using Arca, we implemented the **ARQ** framework. Combining our implementation of the **ARQ** framework and our implementations of the plaintext schemes produced LinearMin, 1/2-ApproxMode, α-ApproxMedian, DomainBucket, DataBucket, and the encrypted schemes from [DPPDGP18]. These schemes are included within the Arca library.

Cryptographic primitives. We used the cryptographic primitives provided by the Arca library. 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 Π_{bas} scheme from [Cas+14] as Σ.

Experimental setup. We ran our experiments as independent, single thread, single process tasks on a Slurm computing cluster consisting of Intel Xeon E5-2670 and E5-2600 processes. All experiments were performed in-memory, with each process allotted a maximum of 300 GB of RAM.

Datasets. We use two real-world datasets in our evaluation. **Gowalla** [CML11] 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 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.[3] We use this dataset (and this partitioning scheme) to replicate Demertzis et al.'s [DPPDGP18] evaluation of their schemes.

[3]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

Scheme	Index size (MB)	Build time (s)
1/2-ApproxMode	219 025.72	20 761.48
α-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
α-ApproxMedian	16 307.45	4711.28

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

Amazon [NLM19] 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 date and time of the reviews so that the domain of the times was $\{0, \ldots, 7058880\}$.

Quantitative evaluation. Figure 12 demonstrates the effectiveness of our domain reductions in reducing the index size, construction time, and server query time of 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 α 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 13 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 provide baseline performance benchmarks for the 1/2-ApproxMode and α-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 14, 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.

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.

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.

10 Extensions

10.1 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 [Ble93], 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 α-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 an *single range cover*-like approach used by Demertzis et al. [DPPDGP18] in their one-dimensional range search schemes; we leave further exploration of such extensions to future work.

10.2 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.)

10.3 Updates

We handle updates by combining a *client-side update cache* (e.g., [WBNM21]) with periodic intermediate Setup and "rebuild" operations. At a high-level, we initialize the cache to have a size of δ_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 propogating 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 δ_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 δ_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.

References

[Yao82] A. C. Yao. "Space-Time Tradeoff for Answering Range Queries (Extended Abstract)". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. 1982, pp. 128–136.

[AS87] N. Alon and B. Schieber. *Optimal preprocessing for answering on-line product queries*. Tech. rep. 71/87. The Moise and Frida Eskenasy Institute of Computer Sciences, Tel-Aviv University, 1987.

[Ble93] G. Blelloch. "Prefix Sums and Their Applications". In: *Synthesis of Parallel Algorithms*. 1st. 1993. Chap. 1, pp. 35–60.

[Pai99] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. 1999.

[BF00] M. Bender and M. Farach-Colton. "The LCA problem revisited". In: vol. 1776. 2000, pp. 88–94. DOI: 10.1007/10719839_9.

[HIM04] H. Hacıgümüş, B. Iyer, and S. Mehrotra. "Efficient Execution of Aggregation Queries over Encrypted Relational Databases". In: *Database Systems for Advanced Applications*. 2004.

[BFPSS05] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. "Lowest common ancestors in trees and directed acyclic graphs". In: *Journal of Algorithms* (2005). DOI: 10.1016/j.jalgor.2005.08.001.

[BKMT05] P. Bose, E. Kranakis, P. Morin, and Y. Tang. "Approximate Range Mode and Range Median Queries". In: *STACS 2005*. 2005, pp. 377–388. DOI: 10.1007/978-3-540-31856-9_31.

[CGKO06] 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*. 2006. DOI: 10.1145/1180405.1180417.

[FH06] J. Fischer and V. Heun. "Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE". In: *Combinatorial Pattern Matching*. 2006.

[Pet08] H. Petersen. "Improved Bounds for Range Mode and Range Median Queries". In: *SOFSEM 2008: Theory and Practice of Computer Science*. 2008, pp. 418–423.

[Gen09] C. Gentry. "A Fully Homomorphic Encryption Scheme". PhD thesis. Stanford University, 2009.

[PG09] H. Petersen and S. Grabowski. "Range Mode and Range Median Queries in Constant Time and Sub-Quadratic Space". In: *Inf. Process. Lett.* 109.4 (2009), pp. 225–228. DOI: 10.1016/j.ipl.2008.10.007.

[GJLT10] 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*. 2010, pp. 605–616.

[CK11] M. Chase and S. Kamara. *Structured Encryption and Controlled Disclosure*. Cryptology ePrint Archive, Report 2011/010. https://eprint.iacr.org/2011/010. 2011.

[CML11] 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*. 2011.

[PRZB11] 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*. 2011.

[DRS12] P. Davoodi, R. Raman, and S. R. Satti. "Succinct Representations of Binary Trees for Range Minimum Queries". In: *Computing and Combinatorics*. 2012, pp. 396–407.

[GOT12] M. T. Goodrich, O. Ohrimenko, and R. Tamassia. "Data-Oblivious Graph Drawing Model and Algorithms". In: *CoRR* abs/1209.0756 (2012). URL: `http://arxiv.org/abs/1209.0756`.

[IKK12] M. S. Islam, M. Kuzu, and M. Kantarcioglu. "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation". In: *NDSS*. 2012.

[Ara+13] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. "Orthogonal Security with Cipherbase." In: *CIDR*. 2013.

[BSA13] 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*. 2013.

[Ska13] M. Skala. "Array Range Queries". In: *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. 2013.

[TKMZ13] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. "Processing Analytical Queries over Encrypted Data". In: *Proc. VLDB Endow.* 6.5 (2013), pp. 289–300. DOI: `10.14778/2535573.2488336`.

[Cas+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*. 2014.

[Gro+14] 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". In: *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)* (2014).

[NT14] G. Navarro and S. V. Thankachan. "Encodings for Range Majority Queries". In: *Combinatorial Pattern Matching*. 2014, pp. 262–272.

[Pap+14] 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.

[CGPR15] 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*. 2015, pp. 668–679. DOI: `10.1145/2810103.2813700`.

[FJKNRS15] 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.

[NKW15] 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*. 2015. DOI: 10.1145/2810103.2813651.

[BPP16] T. Boelter, R. Poddar, and R. A. Popa. *A Secure One-Roundtrip Index for Range Queries*. Cryptology ePrint Archive, Report 2016/568. https://eprint.iacr.org/2016/568. 2016.

[DPPDG16] 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*. 2016, pp. 185–198.

[DDC16] 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*. 2016. DOI: 10.1145/2976749.2978379.

[KKNO16] 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. DOI: 10.1145/2976749.2978386.

[LLWB16] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar. "Fast and Scalable Range Query Processing With Strong Privacy Protection for Cloud Computing". In: *IEEE/ACM Transactions on Networking* 24.4 (2016), pp. 2305–2318. DOI: 10.1109/TNET.2015.2457493.

[ZKP16] 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.

[GSBNR17] 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.

[Aut18] P. C. Authority. *pyca/cryptography*. version 3.4.7. 2018. URL: https://cryptography.io/.

[BGCRS18] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. "The Tao of Inference in Privacy-Protected Databases". In: (2018).

[DPPDGP18] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, M. Garofalakis, and C. Papamanthou. "Practical Private Range Search in Depth". In: *ACM Trans. Database Syst.* (2018).

[GLMP18] 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*. 2018.

[KM18] S. Kamara and T. Moataz. "SQL on Structurally-Encrypted Databases". In: *Advances in Cryptology – ASIACRYPT 2018*. 2018.

[LMP18] 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. DOI: 10.1109/SP.2018.00002.

[ZSLSP18] 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. DOI: 10.1007/978-3-319-98989-1_12.

[BKM19] L. Blackstone, S. Kamara, and T. Moataz. *Revisiting Leakage Abuse Attacks*. Cryptology ePrint Archive, Report 2019/1175. https://ia.cr/2019/1175. 2019.

[DS19] S. Durocher and R. Singh. "A simple linear-space data structure for constant-time range minimum query". In: *Theoretical Computer Science* (2019).

[GLMP19] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. "Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019. DOI: 10.1109/SP.2019.00030.

[GJW19] 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*. 2019.

[NLM19] J. Ni, J. Li, and J. McAuley. "Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects". In: *2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*. 2019. DOI: 10.18653/v1/d19-1018.

[WC19] J. Wang and S. S. M. Chow. *Forward and Backward-Secure Range-Searchable Symmetric Encryption*. Cryptology ePrint Archive, Report 2019/497. https://eprint.iacr.org/2019/497. 2019.

[EHMNS19] 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.

[Fal+20] 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*. 2020.

[Gru+20] 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)*. 2020, pp. 2451–2468. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/grubbs.

[KPT20] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. "The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution". In: *2020 IEEE Symposium on Security and Privacy (SP)* (2020).

[MFST20] E. A. Markatou, F. Falzon, W. Schor, and R. Tamassia. *Reconstructing with Less: Leakage Abuse Attacks in Two-Dimensions*. Cryptology ePrint Archive, Report 2020/1531. https://eprint.iacr.org/2020/1531. 2020.

[SKGY20] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom. *SGAxe: How SGX Fails in Practice*. https://sgaxeattack.com/. 2020.

[Foc21] M. Focus. *Vertica*. 2021. URL: https://www.vertica.com.

[Int21] Intel. *11th Generation Intel Core Processor Desktop*. English. Version 003. Intel. 2021. 136 pp.

[KPT21] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. "Response-Hiding Encrypted Ranges: Revisiting Security via Parametrized Leakage-Abuse Attacks". In: *IEEE Symp. on Security and Privacy*. 2021.

[OK21] 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)*. 2021, pp. 127–142. URL: `https://www.usenix.org/conference/usenixsecurity21/presentation/oya`.

[WBNM21] 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*. 2021, pp. 1892–1905. DOI: `10.1145/3448016.3457306`. URL: `https://doi.org/10.1145/3448016.3457306`.

[Esp22] Z. Espiritu. *Arca*. 2022. URL: `https://github.com/cloudsecuritygroup/arca`.

[EMT22] Z. Espiritu, E. A. Markatou, and R. Tamassia. *Time- and Space-Efficient Range Aggregate Queries on Encrypted Databases*. Under review. 2022.

[FMET22] 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. `https://ia.cr/2022/090`. 2022.

A Formalization and Proof of Corollary 4.4

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

Definition A.1 (Equivalence of plaintext aggregate range query schemes)**.** *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* Π_1 *and* Π_2 *are* equivalent *plaintext aggregate range query schemes if, for all* $k \in \mathbb{N}$*, for all* $\mathsf{poly}(k)$*-size tables* 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*

$$\begin{aligned} \mathsf{DS}_i &\leftarrow \mathbb{S}_i(\mathsf{DB}), \\ U_i &\leftarrow \mathbb{Q}_i(m, Q), \\ S_{i:0} &\leftarrow \{\mathsf{DS}_i[u] \mid u \in U_i\}, \\ (\mathrm{st}_{i:0}, R_{i:0}, U_{i:0}) &\leftarrow \mathbb{R}_i(\perp, Q, S_{i:0}), & \text{and} \\ (\mathrm{st}_{i:j}, R_{i:j}, U_{i:j}) &\leftarrow \begin{cases} \mathbb{R}(\mathrm{st}_{i:j-1}, Q, S_{i:j-1}) & \text{if } U_{i:j-1} \neq \perp \\ (\perp, R_{i:j-1}, \perp) & \text{if } U_{i:j-1} = \perp \end{cases} & \text{for all } j > 0, \end{aligned}$$

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

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

Proof. By (C2) of Definition 4.3, we know that $\Pi = (\mathbb{S}, \mathbb{Q}, \mathbb{R})$ always generates the same subqueries for the aggregate index DS for a given query Q regardless of the underlying contents of DB. Then, we create the following plaintext aggregate range query scheme $\Pi' = (\mathbb{S}', \mathbb{Q}', \mathbb{R}')$ that is Π-equivalent. (We emphasize that $(\mathbb{S}', \mathbb{Q}', \mathbb{R}')$ is not necessarily the most *efficient* definition of a Π-equivalent scheme, but our goal with this proof is to simply prove the *existence* of such a scheme.)

- $\underline{\mathsf{DS} \leftarrow \mathbb{S}'(\mathsf{DB})}$:
  1. output $\mathbb{S}(\mathsf{DB})$;
- $\underline{T \leftarrow \mathbb{Q}'(m, Q)}$:
  1. generate random DB' of size m;
  2. compute $\mathsf{DS}' \leftarrow \mathbb{S}(\mathsf{DB}')$;
  3. compute $U \leftarrow \mathbb{Q}(m, Q)$;
  4. initialize sequence T;
  5. initialize state $\mathrm{st} = \perp$;
  6. while $U \neq \perp$,
     (a) append all elements in U to T;
     (b) compute $V \leftarrow \{\mathsf{DS}'[u] \mid u \in U\}$;

(c) compute $(\mathrm{st}, R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, V)$;

7. output T;

- $\underline{(\mathrm{st}, R, U) \leftarrow \mathbb{R}'(\mathrm{st}, m, Q, S)}$:
  1. generate random DB' of size m;
  2. compute $\mathsf{DS}' \leftarrow \mathbb{S}(\mathsf{DB}')$;
  3. compute $U \leftarrow \mathbb{Q}(m, Q)$;
  4. initialize empty sequences T and `query_sizes`;
  5. initialize state $\mathrm{st} = \perp$;
  6. initialize aggregate $R = \perp$;
  7. while $U \neq \perp$,
     (a) append $|U|$ to `query_sizes`;
     (b) append all elements in U to T;
     (c) compute $V \leftarrow \{\mathsf{DS}'[u] \mid u \in U\}$;
     (d) compute $(\mathrm{st}, R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, V)$;
  8. compute $U \leftarrow \mathbb{Q}(m, Q)$;
  9. set $\mathrm{st} = \perp$;
  10. for `size` in `query_sizes`,
      (a) pop first `size` elements of T into sequence S';
      (b) compute $(\mathrm{st}, R, U) \leftarrow \mathbb{R}(\mathrm{st}, m, Q, S')$;
  11. output $(\perp, R, \perp)$; □

B Workarounds for Query Size Limitation in LinearMin

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 encrypted database deployments, the encrypted minimum structure would likely be stored alongside a standard encrypted range structure (e.g., [FJKNRS15; DP-PDGP18; FMET22]). In such settings, when faced with an unanswerable query, the client may instead simply send a standard range query to the encrypted range structure. Then, the client can compute the minimum by decrypting the returned records 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 benefits of the LinearMin scheme on larger queries while incurring minimal performance overhead for the small queries.

C Query Type Transformations

Transformations from range sum query. Solutions to 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. Solutions to 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. [DPPDGP18]. Instead of creating one aggregate structure, we create k aggregate structures, where the ith 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 Acknowledgements

When you stand on the shoulders of giants, you can see really far.

Nothing in this thesis would have happened had it not been for **Roberto Tamassia**, who was not only my thesis advisor but also my professor for cs1660, the first security course I took at Brown. I distinctly remember the magical feeling I had when Roberto introduced me to structured encryption for the first time—in that moment, I discovered a way to unify my interests in algorithms, data structures, and systems with my love for security. Roberto took a chance on me multiple times throughout my Brown career, such as when he agreed to advise my thesis during his sabbatical, or when he supported my last-minute application for the Randy Pausch award, or, and most especially, when he hired me as a HTA for cs1660 in 2019 and sparked a multi-year dive into the security field. I also am grateful to **Vasilis Kemerlis** for serving as the reader to this thesis.

I am also extremely grateful for my CSG and ESL collaborators, all of whom welcomed me into their research meetings with open arms: **Francesca Falzon**, **Marilyn George**, **Evangelia Anna Markatou**, **Lucy Qin**, and **Zheguang Zhao**. Their mentorship, professionalism, and kindness transformed an emotionally turbulent period into some of the most intellectually engaging and inspirational times I've had at Brown. I am thankful for all of their time and energy in a time where energy was often hard to find. I also must thank **Seny Kamara**, who was the first to open the door to applied cryptography research in August 2020. Seny's "Crypto for the People" was one of the main drivers that caused me to pursue research further in 2020, and I aspire for my own work to one day be as "cool" and impactful as Seny's work.

I received several sources of funding that enabled me to devote a significant amount of time to research in 2021: **CrowdStrike Foundation**, the **Center for Cyber Safety and Education**, and Brown CS's **Randy F. Pausch Computer Science Undergraduate Summer Research Award**. These grants allowed me to multiply many of the results that appear in this thesis as well as those that appear in other publications.

On a more personal level, 2020 and 2021 were filled with a number of challenges, some obvious to the world and others much less so. My friendship with my recurring cs1660 and cs2951e family, in particular, **Abigail Siegel** and **William Schor**, helped immensely to get me through the academic year. Their willingness to listen to my research rambles, share my enthusiasm for new insights, and just be friends at the end of the day helped me be happy with and proud of my work. I am also fortunate to have had a fantastic co-MTA and friend, **Julia McClellan**, whose organization skills often surpassed my own and to whom much credit is owed for shepherding 1200+ UTAs through the variety of challenges we encountered. Additionally, **Casey Nelson**'s unconditional friendship helped in many ways to push me to the completion of my Brown degrees. Finally, I am thankful to the many players who joined the "unofficial board game club", all of whom trusted me with hours of time so I could (try to) craft interesting, confusing, and hopefully memorable experiences for them with games, cooking, and storytelling each evening. I hope those sessions were as much of a community-building respite from work for you all as they were for me.

Lastly, as a small number of people at Brown know, I was diagnosed with and battled Stage IV cancer for seven months in June 2019. I was extremely lucky to have had a strong support network of people at Brown CS during this time, including (but not at all limited to) my Brown PLT colleagues and friends, my MTA supervisor (**Thomas Doeppner**), my family, and **Amy Wang**, **Andrew Wagner**, **Karen Tu**, **Lisa Phinisee**, **Nicole Steinberg**, and **Shawna Huang**. In spite of the many challenges that arose in the months following, they all extended exceptional amounts of patience, understanding, and support during my treatment and throughout my recovery, and neither I nor this thesis would have been around today if it wasn't for their life-saving support.