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

Zachary Espiritu  
Brown University  
zachary_espiritu@brown.edu

---

**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. In practice, 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 traditional range-reporting schemes, the encrypted form of the records means that the client must compute the desired aggregate function themselves, incurring bandwidth and client-side computation proportional to the number of records in the queried range. This makes searchable encryption schemes problematic to use in settings that necessitate high aggregate computation performance.

In this work, we tackle this challenge by developing cryptographic schemes for handling range aggregate queries such as sum, minimum, median, and mode queries. For each of these query types, we propose new schemes that answer range aggregate queries and offer different time, space, and bandwidth tradeoffs for each of these queries. More generally, our work aims to demonstrate how one may unify techniques from the traditional data management literature with STE techniques to produce efficient STE schemes with small amounts of leakage. We are the first to present encrypted range query schemes for median and mode queries; our schemes specifically focus on the approximate variation of the problem in order to achieve practical, storage-efficient constructions. We also present structures for sum and minimum queries that are more space-efficient in practice over the previous structures presented by Demertzis et al. (ACM Trans. Database Syst. '18) at the cost of small leakage.

---

**I. INTRODUCTION**

Searchable symmetric encryption (SSE) [5] and its more generalized variant, structured encryption (STE) [3], are privacy-preserving techniques that allow clients to outsource their data to a semi-honest server in an encrypted format and perform various kinds of queries on this data. Much attention has been drawn to improving the practicality for such structures by specializing them for more expressive types of queries, such as range queries. Encrypted range structures are SSE structures specifically optimized for range queries. The BlindSeer construction from [13], the garbled-circuit-based construction from [1], and the range tree constructions from [6–8], [15], [16] are all examples of such range structures. In these works, range queries are effectively presented as a simple filter operation, where a predicate is applied to a single attribute in a database and the set of records matching the desired predicate are 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 (such as count, min, sum, average, etc.) 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;
```

This query applies a filter over the age attribute and an aggregation function (MEDIAN) over the salary attribute. 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 responding server only needs to output a single numerical value (the median salary). For applications where the amount of bandwidth between the server and client is a performance concern, integrating aggregation functions directly into queries can substantially reduce bandwidth as the server only needs to send a single, constant-size value to the client.

Range aggregates over encrypted databases. For practical applications, using standard SSE to answer aggregate queries incurs a significant overhead over traditional “plaintext” data management approaches. In many SSE structures, the server cannot compute the aggregate server-side and must return the entire set of records satisfying the filter. This incurs linear-size bandwidth and requires the client to spend linear resources to decrypt each of the records before computing the aggregate. Conversely, one may precompute 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 bandwidth.

Many approaches have been proposed to bridge this performance gap for encrypted databases. Fully homomorphic encryption (FHE) [9], 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) [12] 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 (such as min and max) and still requires the server to spend computation time to add AHE ciphertexts.

Alternatively, some of the aforementioned encrypted range structures (e.g., [6]–[8]) 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. This makes these schemes somewhat more practical for aggregate queries; however, such an approach still incurs a high storage overhead for the overall range structure, especially when the underlying dataset is sparse.

**Our contributions.** In this work, we take strides in filling in the performance gap of encrypted aggregate range queries by presenting several STE schemes that efficiently answer different kinds of common aggregate queries. We take an interdisciplinary approach by demonstrating how simple combinations of data structures previously developed in the plaintext data management community with STE primitives can yield efficient (and surprisingly secure) constructions in the encrypted setting.

Our goal is to discover schemes that enjoy nearly constant query time and bandwidth with low storage overhead. A comparison of our schemes with prior work is listed in Table I. Our schemes can be divided into two halves. In the first half, we propose a series of domain reduction schemes which are designed to allow one to optimize encrypted range aggregate structures for sparse databases. We then explain how to use our reduction techniques to easily and significantly improve the performance of the previous range sum and range minimum constructions by Demertzis et al. [7] on sparse databases, with small performance and leakage tradeoffs.

In the second half, we propose entirely new schemes for the encrypted variant of the range minimum, approximate range mode, and approximate range median problems. Our range minimum scheme improves the previously-best-known \( O(m + n \log n) \) storage overhead of Demertzis et al.'s 2-round protocol [7] to a \( O(m) \) 1-round protocol at the cost of prohibiting some small queries. To our knowledge, our approximate range mode and range median schemes are the first schemes considering the range mode and median problems in the STE literature and they allow for constant-time and constant-size queries. Furthermore, we formally define the leakage of each of our schemes, which may include endpoint query patterns (EQP), or whether or not two range queries to the same structure have the same endpoints, and subquery patterns (SQP), or whether or not two range queries rely on the same precomputed subaggregates in the query structure. We additionally identify a new security definition that we refer to as data-obliviousness that we use to signpost our leakage goals in the aggregate setting. Finally, we perform an empirical evaluation of all of our proposed schemes in terms of storage, query time, and bandwidth, and conclude with remarks on future potential directions for this work.

**II. Evaluation**

In this section, we describe our evaluation of our schemes.

**Implementation.** We implemented all of our schemes in Python. For cryptographic primitives, we used version 3.4.7 of the Python cryptography library [14]. We implemented \( \Sigma_{MM}, \Sigma_{DX}, \) and \( \Sigma_A \) via our own implementation of the \( \Pi_{bas} \) scheme from Cash et al. [2]. We also used the Python numpy package [10] for optimized operations over large arrays. We evaluated all schemes with \( \alpha = 0.5 \).

**Datasets.** We use two real-world datasets in our evaluation. Gowalla [4] 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 check-in date and times. The date and time of the check-ins were converted to Unix time integers, then shifted so that the domain of the times was \( \{0, \ldots, 54083068\} \). 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. \(^1\) We use this dataset (and this partitioning scheme) to replicate Demertzis et al.'s [7] evaluation of their schemes. \(^2\)

Amazon [11] contains 51,311,621 Amazon item ratings for all 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 1 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 DATABASE bucket scheme results in significantly lower storage and construction overhead than DATABASE—since \( m \) and \( \alpha \) are public parameters, since 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.

---

1. We were unable to run the DPPDGP-MIN1 scheme on any partition set of Gowalla within 8 hours due to the number of encryptions needed to encrypt the ST. Thus, we instead computed the number of bytes needed to store the encrypted form of each entry of ST (162 B) and the average time for computing a hash of each entry’s label and encrypting its value (15356 ns). Then, we computed the \# of cells in the ST and extrapolated the index size and runtime of DPPDGP-MIN1 using the previous two metrics. Note that our encryption time estimate is conservative, as it does not take into account the time it takes to access elements from the large (plaintext) ST array.

2. We did not use the “USPS” dataset mentioned in Demertzis et al. [7] because the reference for the dataset linked in that paper was a dead link.
TABLE I

Our contributions compared to previous work of Demertzis et al. (DPPDGP) [7]. Asymptotics are in terms of big-O, where \( m \) denotes the size of the domain and \( n \) is the number of records, and, when mentioned, \( 0 < \alpha < 1 \) and \( d \geq 2 \) (where \( d \in \mathbb{Z} \)) are tunable constants. All schemes have constant-size queries. “Query” refers to query handling runtime at the server and the client independently. For security comparison, we denote whether the schemes are data-oblivious (DO), leak endpoint query patterns (EQP), or leak subquery patterns (SQP).

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Storage Complexity</th>
<th>Communication Complexity</th>
<th>Client Complexity</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Server</td>
<td>Query</td>
<td>Bandwidth</td>
<td>Rounds</td>
</tr>
<tr>
<td>Sum</td>
<td>( m^\alpha + n )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sum+DOMAINBUCKET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum+DATABUCKET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum+DIMENSIONLIFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>( m \log m )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Minimum+DOMAINBUCKET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum+DATABUCKET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum+DIMENSIONLIFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>( 1/2)-APPROXMODE</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Median</td>
<td>( \alpha)-APPROXMEDIAN</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 1. Scheme costs on the Gowalla dataset with DPPDGP-SUM [7], DOMAINBUCKET [-], and DATABUCKET [-].

For the minimum schemes, Figure 2 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. We believe that further applications of our domain reductions schemes to LINEARMIN can push the storage overhead down even further; we leave such evaluations to future work.

Table II provides baseline performance benchmarks for the \( 1/2\)-APPROXMODE and \( \alpha\)-APPROXMEDIAN schemes for \( \alpha = 0.5 \). We can see that the storage and build time overhead of both schemes is comparable to that of DPPDGP-MIN1.

III. Conclusion

We have submitted a long form of this work for publication (currently under double-blind review) and expect to have an preprint available for public dissemination within the next two weeks. We are currently investigating further extensions of this work, such as developing structures for higher-dimensional aggregate queries, as well as integration with more complex SQL-like encrypted databases.

TABLE II

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Index size (MB)</th>
<th>Build time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1/2)-APPROXMODE</td>
<td>219.025.72</td>
<td>20.761.48</td>
</tr>
<tr>
<td>( \alpha)-APPROXMEDIAN</td>
<td>130.459.63</td>
<td>13.603.23</td>
</tr>
</tbody>
</table>
REFERENCES


