Sketch-based Secure Query Processing for Streaming Data

JIANZHE YU, Hong Kong University of Science and Technology, China
FENG HAN, Alibaba Group, China
QI DONG, Hong Kong University of Science and Technology, China
QIYAO LUO, Hong Kong University of Science and Technology, China
WEIRAN LIU, Alibaba Group, China
LIN QU, Alibaba Group, China
KE YI, Hong Kong University of Science and Technology, China

Sketching is an effective approach to dealing with high-volume streaming plaintext data with bounded memory and computing cost, while providing provable guarantees on the query accuracy. In this paper, we present a sketching framework under the model of outsourced secure multi-party computation (MPC), where data is uploaded to the computing parties in a secret-shared form. We show how our framework supports a variety of sketches, including the Count-Min Sketch, HyperLogLog, SpaceSaving, and the Greenwald-Khanna Sketch, which allow the secure evaluation of group-by aggregation queries, frequency estimation, top-k queries, distinct count, and rank/quantile queries. Our framework can maintain these sketches with $\tilde{O}(1)$ cost per update amortized, while using a bounded amount of memory that can be configured based on the user's budget and query accuracy requirements. Experimental results show that our framework can process each stream update with an amortized cost of less than 1 ms, significantly outperforming prior work.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**; • **Theory of computation** → Streaming, sublinear and near linear time algorithms.

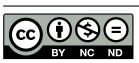
Additional Key Words and Phrases: Secure multi-party computation; Sketching; Approximate query processing

ACM Reference Format:
Jianzhe Yu, Feng Han, Qi Dong, Qiyao Luo, Weiran Liu, Lin Qu, and Ke Yi. 2026. Sketch-based Secure Query Processing for Streaming Data. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 233 (June 2026), 25 pages. https://doi.org/10.1145/3802110

1 Introduction

Secure multi-party computation (MPC) is a powerful technique that allows several parties to jointly compute a given function on their secret inputs. A variant of the model, known as the outsourced MPC model [7, 8, 11, 12, 31, 41, 42, 55, 61] or server-aided secure computation [52, 53], has gained much traction in recent years. In this model, the parties only provide storage and computing powers (hence also called "servers"), while the input is uploaded by one or more data owners in a secret-shared form (see Section 2 for the details on secret sharing). A client (who may be one of the data owners) later queries the uploaded data and receives the query results also in secret-shared form. This variant captures the scenario where the data owners/clients do not have the

Authors' Contact Information: Jianzhe Yu, jyuca@cse.ust.hk, Hong Kong University of Science and Technology, China; Feng Han, fengdi.hf@alibaba-inc.com, Alibaba Group, China; Qi Dong, qdongaf@connect.ust.hk, Hong Kong University of Science and Technology, China; Qiyao Luo, qluoak@cse.ust.hk, Hong Kong University of Science and Technology, China; Weiran Liu, weiran.lwr@alibaba-inc.com, Alibaba Group, China; Lin Qu, xide.ql@taobao.com, Alibaba Group, China; Ke Yi, yike@cse.ust.hk, Hong Kong University of Science and Technology, China.



ACM 2836-6573/2026/6-ART233
https://doi.org/10.1145/3802110

storage/computing power and/or do not trust each other, hence choosing to upload their data to two or more non-colluding cloud service providers. It also minimizes the implementation complexity of the data owners/clients, which used to be a main hurdle for deploying MPC solutions in practice. In particular, it allows the input data to be contributed by any number of stateless data owners.

Most work on the outsourced MPC model studies the static case, where all data is uploaded to the servers in one go. In this paper, we consider the streaming setting, where data is uploaded as a stream, while a client may at any time issue queries to the servers on all the data uploaded so far. A naive solution for the streaming problem is to simply store the uploaded data on the servers (in a secret-shared form) upon each update, and then run a static MPC protocol for each query. This solution requires $O(n)$ space on the servers, where n is the current length of the stream, incurs $O(1)$ cost[1] per update, and answers each query with cost $O(n \log n)$ for most query types, including all the queries considered in this paper. Very recently, [61] has proposed a different solution for this problem (although they did not use the term "streaming" explicitly). Their solution uses $O(n)$ space and incurs $O(n)$ cost per update and per query, namely, they reduce the query cost by an $O(\log n)$ factor but at the expense of increasing the update cost by an $O(n)$ factor. It is not clear if this trade-off is advantageous in typical streaming applications where there are much more updates than queries. Furthermore, it still uses $O(n)$ space, which grows as the length of the stream. This is clearly not scalable for long or even unbounded streams.

1.1 Our Solution

In this paper, we propose S^3, a **S**ketch-based **S**ecure **S**treaming query processing framework, to process a high-speed stream of updates while supporting various types of queries with bounded memory, computing time, and communication cost on the servers. On plaintext data, sketching is a powerful tool for dealing with high-speed streams, trading a small loss of accuracy for much better performance. Under the MPC model, the potential benefits of sketching are even greater, considering the high storage, communication, and computation cost when data must be manipulated in a secret-sharing form. However, existing work under MPC has only used sketches to store a static dataset compactly and securely, but has not considered updates.

There are two standard solutions to support updates for a sketch under MPC: updating all the cells in the sketch per update, incurring $O(s)$ cost where s is the sketch size, or running the sketch using ORAM on top of MPC [27, 56], whose cost is polylogarithmic in theory but very high in practice. To bridge the gap, our S^3 framework uses a batch update idea to reduce the update cost to $O(\log s)$ amortized, while only using $O(s)$ storage on the servers; please see Table 1 for the more precise complexity results. S^3 supports a variety of common sketches so that many queries can be answered with a provable error guarantee proportional to $1/s$ or $1/\sqrt{s}$. Thus, users can control the trade-off between query accuracy and their cloud service cost by setting s accordingly.

Specifically, we make the following contributions in this paper:

(1) We formalize S^3, a sketch-based framework in the streaming-MPC model for processing high-speed streaming data with bounded memory, computational and communication costs. This is the first time sketches can be efficiently (in both theory and practice) updated under MPC.
(2) We design MPC protocols to support four different sketches under our S^3 framework: *Count-Min Sketch*, *SpaceSaving*, *HyperLogLog*, and the *Greenwald-Khanna Sketch*, which support group-by aggregation queries, frequency estimation, top-k queries, distinct count, and rank/quantile queries, respectively, with provable error guarantees. All these sketches

[1]We use the term "cost" to refer to both the running time and communication cost of the servers when they are asymptotically the same.

can be configured appropriately to control the trade-off between the query answer accuracy and the processing cost on the servers.

(3) We have built a system prototype with three semi-honest non-colluding servers. Through an extensive experimental evaluation, we demonstrate that S^3 has an update cost as low as 1 ms and a few KB of communication per update amortized for most sketches, significantly outperforming existing solutions. On long streams of size exceeding 10^5, our framework is the only viable solution capable of processing updates and queries efficiently. Furthermore, we also show that the approximation error is minimal: mostly less than 1% on both synthetic and real datasets while incurring a reasonably low cost.

1.2 Related Work

Many exact MPC query processing engines have been developed that allow servers to collaboratively compute query results without disclosing private data from clients [7, 8, 11, 31, 41, 46, 50]. These works typically design secure SQL operators to answer complex SQL queries through composition for static databases. Recently, MPC query processing protocols for dynamic data have attracted some attention. ShortCut [61] follows the MPC query processing engine approach and designs secure update protocols for different SQL operators using materialized views which avoid query recomputation for each update data. Other works [12, 53, 59] reduce the workload of MPC protocols by allowing differentially private information leakage.

Approximate query processing (AQP) techniques have been very effective in overcoming the high computation cost of MPC protocols. On plaintext, there are mainly two AQP techniques: sketching and sampling. Sketch-based techniques use compact data structures to provide approximate query results [18, 25, 28, 29, 43, 44], while sampling-based techniques take small random subsets of data to approximate query results [19, 36, 48]. Both techniques can achieve $\tilde{O}(1)$ update time and $\tilde{O}(s)$ query time, where s is the sketch size or sample size. Sketch-based solutions usually provide more accurate query answers but each sketch only supports a particular query type, while a sample can be used to answer many types of queries but with lower accuracy. Both techniques have been applied to the MPC setting. SAQE [13] and MASQUE [42] apply sampling techniques on static data, where MASQUE uses batch sampling to achieve an amortized cost of $\tilde{O}(s)$ per sample. Sketch-based MPC protocols have been developed for different query types including heavy hitters [16], quantile [40] and cardinality estimation [34, 54]. However, these sketch-based MPC solutions have only considered how to build the sketch on static data, and do not support streaming updates.

While MPC is a purely software-based approach to dealing with private data, only relying on standard cryptographic assumptions, there are also many works adopting a trusted execution environment, such as Intel SGX [9, 39, 47]. These hardware-based solutions are usually more efficient, but one has to be careful in the implementation (e.g., using oblivious algorithms) in order to avoid side channel attacks, in addition to the trust assumption on the hardware vendor.

2 Preliminaries

2.1 The Streaming-3PC Model

In this section we define our streaming-3PC model, which involves three computing parties, with at most one party being corrupted by a semi-honest adversary, meaning that the adversary can view the data and all the communication transcripts of the corrupted party but cannot alter its behavior. The 3PC variant has been widely adopted in recent years in the MPC literature due to the high efficiency of its primitives [6, 16, 30, 31, 41, 45, 46, 55, 61]. Note that, however, all our algorithmic results apply to other MPC variants (such as 2PC or a malicious adversary), by appropriately changing the MPC primitives.

2.1.1 Secret Sharing. Secret sharing is an essential building block of MPC. In this paper, we adopt *replicated secret sharing* [45] tailored for 3PC: A private ℓ-bit number x is split into three shares x_0, x_1, x_2, while each party is given two of the three shares. There are two ways to perform the sharing: *additive shares* satisfy $x_0+x_1+x_2 \equiv x \pmod{2^\ell}$, which are suitable for arithmetic operations such as addition and multiplication, while *boolean shares* satisfy $x_0 \oplus x_1 \oplus x_2 = x$ ($\oplus$ stands for bitwise-XOR), which are more efficient for bitwise operations. To obtain a sharing of x, the data owner simply picks uniformly random ℓ-bit numbers x_1 and x_2, and then computes x_0 such that the respective equality above holds. It should be clear that the three shares can reconstruct x, while any two of them reveal no information about x. There are efficient protocols to convert between the two kinds of sharing schemes with constant cost [32, 45], and we can perform such conversions as needed. Henceforth we will use $[\![x]\!]$ to denote the secret-sharing form of x without specifying which particular sharing scheme is used.

2.1.2 The Streaming-3PC Model. In the outsourced 3PC model [7, 8, 11, 12, 31, 41, 42, 55, 61] (a.k.a. server-aided MPC model [52, 53]), data owners first upload their data to three computing parties (a.k.a. servers) in a secret-shared form. Afterwards, a client, which can be one of the data owners, may issue queries to the three parties. Then they jointly evaluate the query using a 3PC protocol and return the query result to the client, still in a secret-shared form, which can be reconstructed by the client.

In our streaming-3PC model, the data owners upload their data to the three servers in a streaming fashion. Similarly, clients may also issue queries at any time during the streaming process. More formally, the input is a potentially unbounded stream $x = ((o_1, x_1), (o_2, x_2), \cdots, (o_t, x_t), \cdots)$, where o_t denotes the operation (upload/query) at time t, and x_t is the uploaded data or query parameter. For a query (o_t, x_t), its output is a function $\mathcal{F}(\bigcup_{i<t, o_i \text{ is an upload}} \{x_i\}, x_t)$, i.e., it is evaluated on all the data uploaded so far with a query parameter x_t. The semantics of x_t will depend on the query type. For instance, for a look-up query, x_t is the queried key; for a range counting query, it is the query range; for queries without a parameter (e.g., distinct count), it is set to a dummy value $\perp$.

As with stream processing over plaintext, we would also like the costs of the servers (memory, computing, and communication) to be bounded, or grow mildly with the stream length (e.g., logarithmically), while providing reasonably accurate query results.

2.1.3 Security definition. It should be clear from the definition of the streaming-3PC model that the data owners learn nothing and the client learns only the query results. Thus it suffices to focus the security definition on the adversary, who corrupts one of the 3 computing parties. In our streaming-3PC model, we require that the adversary should learn nothing other than the operator type (i.e., each o_t) of the stream. This can be formalized by extending the standard real-ideal paradigm [24] to the streaming setting. In the real world, the adversary sees the stream received by the corrupted party, as well as all of its communication transcripts during running a 3PC protocol π (with a security parameter κ). On the other hand, the ideal-world view only consists of the stream of operators $(o_1, o_2, \dots)$. We say that π is secure against a semi-honest adversary if there exists a simulator that, for each t, can simulate the adversary's view in the real world up to time t from the ideal-world view up to time t, in the sense that they can be distinguished only with negligible (in κ) probability.

2.1.4 Protection Scope. While the streaming-3PC model protects the computation procedure, there are potential information leakages that fall outside the protection scope of MPC but still require attention. The first arises from query frequencies, as we assume the operation type is public. A straightforward mitigation strategy is a prior agreement on query frequency (i.e., one query per 1000 updates). The second concern is that query results are disclosed to the clients, who may

compute differences between successive queries. This falls under the scope of differential privacy, which we elaborate on in Section 5.

2.1.5 Complexity measures. The complexity of a secure protocol is measured by the computation time, communication cost, and the number of communication rounds. Following most existing work [31, 41, 42, 46, 57], we measure the asymptotic complexity on the word level: any standard arithmetic or logical operation between two ℓ-bit words (in secret shared form) is considered to have $O(1)$ cost in running time, communication cost and rounds. We use the term "cost" to refer to both computation time and communication cost, as they are asymptotically the same in our protocols.

2.2 3PC Primitives

All the 3PC primitives introduced below take an input in secret-shared form and output the result also in secret-shared form but using fresh randomness, so that there is no correlation between the randomness in the input and output shares.

2.2.1 Circuit-based Computation. There are 3PC protocols [45] that take $[\![x]\!]$, $[\![y]\!]$ and output $[\![x \oplus y]\!]$, where $\oplus$ can be any standard arithmetic or logic operation, including addition, multiplication, bitwise-AND, bitwise-XOR, etc. These protocols thus allow any circuit-based computation to be carried out in the 3PC model, with cost proportional to the size of the circuit and the number of rounds proportional to the depth. For example, $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ can be computed by a circuit of $O(n)$ size and $O(\log n)$ depth, thus $[\![x_1 \oplus x_2 \oplus \cdots \oplus x_n]\!]$ can be computed from $[\![x_1]\!], \ldots, [\![x_n]\!]$ with $O(n)$ cost and $O(\log n)$ rounds. Note that, however, when $\oplus$ is $+$, $[\![x_1 + \cdots + x_n]\!]$ can be computed from $[\![x_1]\!], \ldots, [\![x_n]\!]$ completely locally when using additive shares.

Another useful circuit is a multiplexer. On input $[\![x_0]\!]$, $[\![x_1]\!]$, $[\![c]\!] \in \{0, 1\}$, it outputs $[\![(x_1 \text{ if } c = 1 \text{ else } x_0)]\!]$. This circuit has $O(1)$ size and $O(1)$ depth.

2.2.2 Prefix and Suffix. For an associative operator $\oplus$, the prefix-$\oplus$ operation takes a sequence $X = ([\![x_1]\!], \ldots, [\![x_n]\!])$ and outputs the sequence $([\![x_1]\!], [\![x_1 \oplus x_2]\!], \ldots, [\![x_1 \oplus \cdots \oplus x_n]\!])$. In particular, we will make use of the following instantiations ($\perp$ denotes the dummy value):

(1) When $\oplus$ is arithmetic addition $+$ (define $x + \perp = x$), we call the result the prefix-sum of X.
(2) When $\oplus$ is taking the maximum or minimum, we call it the prefix-max/min of X.
(3) When $\oplus$ is bitwise-AND operation of two bits, we refer to the result as the prefix-and of X.
(4) When $\oplus$ is defined as:

$$x \oplus y = \begin{cases} x & \text{if } y = \perp \\ y & \text{otherwise} \end{cases},$$

this operation, denoted by prefix-copy, replaces dummy values in X with the nearest preceding non-dummy value.

A variant is the segmented prefix-$\oplus$ [58], which takes two sequences $X = ([\![x_1]\!], \ldots, [\![x_n]\!])$ and $Y = ([\![y_1]\!], \ldots, [\![y_n]\!])$ as input. In this variant, consecutive equal elements in X define segments, and the operation computes the prefix-$\oplus$ of Y within each segment. For instance, given $X = ([\![1]\!], [\![1]\!], [\![2]\!], [\![3]\!], [\![3]\!])$, the output would be $([\![y_1]\!], [\![y_1 \oplus y_2]\!], [\![y_3]\!], [\![y_4]\!], [\![y_4 \oplus y_5]\!])$.

There are circuits of $O(n)$ size and $O(\log n)$ depth for both prefix-$\oplus$ and segmented prefix-$\oplus$ [38, 58], so they can be computed under 3PC with $O(n)$ cost and $O(\log n)$ rounds. But when $\oplus$ is $+$, prefix-$\oplus$ can be computed locally when using additive shares.

The suffix-$\oplus$ operation outputs $([\![x_1 \oplus \cdots \oplus x_n]\!], \ldots, [\![x_{n-1} \oplus x_n]\!], [\![x_n]\!])$, and it can be computed similarly.

Protocol		Space	Update		Query		Total cost		Result
			Cost	Round	Cost	Round	Dense	Sparse	
SECRECY [41]		$O(n)$	$O(1)$	$O(1)$	$O(n \log n)^*$	$O(\log n)$	$O(n^2 \log n)$	$O((n^2/s) \log n)$	Exact
ShortCut [61]			$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$		
S^3	CM	$O(s)$	$O(\log s)^\dagger$	$O(\log s)^\ddagger$	$O(s)$	$O(1)$	$O(ns)$	$O(n \log s)$	Approx.
	HLL				$O(s \log s)$	$O(\log s)$	$O(ns \log s)$		
	SS				$O(s \log s)$	$O(\log s)$	$O(ns \log s)$		
	GK	$O(s')^\diamond$	$O(\log s')^{\diamond\dagger}$	$O(\log s')^{\diamond\ddagger}$	$O(s')^\diamond$	$O(1)$	$O(ns')^\diamond$	$O(n \log s')^\diamond$	

* SECRECY uses the $O(n \log^2 n)$ bitonic sorting network [10], which can be replaced by the $O(n \log n)$ sorting protocol [5].
$^\diamond$ Running size $s' = 2s \ln(\frac{n}{s} + 2) + 2$ † Amortized cost ‡ Total rounds of a batch updates

Table 1. Comparison of different 3PC stream query processing protocols (n is the stream size, s is the sketch size).

2.2.3 Sorting. Sorting takes an array $X = (\llbracket x_1 \rrbracket, \ldots, \llbracket x_n \rrbracket)$ and sorts it by $X.key$. Dummy tuples are placed after non-dummy tuples. Under 3PC, sorting can be done with $O(n \log n)$ cost and $O(\log n)$ rounds [5].

2.2.4 Compaction. The compaction operation takes an array $X = (\llbracket x_1 \rrbracket, \ldots, \llbracket x_n \rrbracket)$ and a binary array $T = (\llbracket t_1 \rrbracket, \ldots, \llbracket t_n \rrbracket)$ where $t_i \in \{0, 1\}$. An element x_i is said to be marked if $t_i = 1$. It permutes X such that all marked elements in X appear before unmarked elements while preserving their original relative order. We denote this operation as compact X by T. When there is no T, compact simply moves all the non-dummy elements to the front in X. Compaction can be done with $O(n)$ cost and $O(1)$ rounds under 3PC [6].

2.2.5 Pseudorandom Function. Define $[n] := \{1, 2, \ldots, n\}$. An ideal hash function (a.k.a. random oracle) is an $f : [2^\ell] \to [V]$ chosen randomly from all functions mapping $[2^\ell]$ to $[V]$, so that $f(1), \ldots, f(2^\ell)$ are uniformly randomly distributed in $[V]$ and are mutually independent. A pseudorandom function (PRF) is also a function $h : [2^\ell] \to [V]$ but chosen randomly from a family of 2^κ functions, such that $f(\cdot)$ and $h(\cdot)$ are computationally indistinguishable (in κ). LowMC [4] provides a circuit-based protocol that evaluates a PRF at n locations with $O(n)$ cost and $O(1)$ rounds, i.e., on an input array $X = (\llbracket x_1 \rrbracket, \ldots, \llbracket x_n \rrbracket)$, it outputs $(\llbracket h(x_1) \rrbracket, \ldots, \llbracket h(x_n) \rrbracket)$.

3 Secure Linear Sketches

In this section, we introduce the S^3 framework and demonstrate how S^3 implements the *linear sketch* in the streaming-3PC model. Linear sketches apply linear transformations to process data streams. It includes well-known algorithms such as *Count-Min Sketch* [18], *HyperLogLog Sketch* [25], and others. Consider a data stream where the operation key k is from a large universe. This universe is mapped to a compact sketch array $\mathcal{S}$ of size s via a linear transformation, usually achieved by a random hash function h. That is, h maps the key k to a specific cell index in the sketch, resulting in an update or a query at $\mathcal{S}[h(k)]$.

3.1 The S^3 Framework

There are two types of operations in data streams: the update operation is associated with a (k, v) pair, where k is always uploaded by data owners and v is either uploaded or computed from k. The query operation is associated with a key k.

When processing the (k, v) pair in plaintext, the sketch is updated as $\mathcal{S}[h(k)] \leftarrow \mathcal{S}[h(k)] \oplus v$, where $\oplus$ differs by different sketches. Under 3PC, the sketch $\mathcal{S}$ is stored in a secret-shared form, as well as the $(\llbracket k \rrbracket, \llbracket v \rrbracket)$ pair. While we can perform the update $\llbracket \mathcal{S}[h(k)] \rrbracket \leftarrow \llbracket \mathcal{S}[h(k)] \rrbracket \oplus \llbracket v \rrbracket$ from $\llbracket \mathcal{S}[h(k)] \rrbracket$ and $\llbracket v \rrbracket$ using standard 3PC primitives, the difficulty is that the parties can only compute $\llbracket h(k) \rrbracket$, namely, the servers have no plaintext access to the index to the array cell to be updated.

Note that, while revealing the plaintext $h(k)$ is safe for one update as it is uniformly random in $[s]$, it is not over multiple updates. In particular, updates with the same key will always update the same cell in the array.

One naive approach is to perform updates to all the cells of $\mathcal{S}$ with a multiplexer: $[\![\mathcal{S}[j]]\!] \leftarrow [\![\mathcal{S}[j]]\!] \oplus ([\![v]\!]$ if $h(k) = j$ else $[\![0]\!])$ for all $j \in [s]$, but it would incur $O(s)$ cost. S^3 reduces this to $O(\log s)$ amortized using a batch update idea that performs s updates in a batch. The S^3 framework is presented in Algorithm 1.

Algorithm 1: S^3 Framework

```
Initialize buffer [[B]] ← ∅ and sketch [[S]];
foreach operation type T, private data [[x]] do
    if T is Update then
        Append [[x]] to [[B]] ;
        // Update when the buffer is full
        if |[[B]]| = s then
            [[S]] ← Merge ([[B]], [[S]]), [[B]] ← ∅;
    if T is Query then
        Output Query ([[x]], [[B]], [[S]]) ;
```

S^3 keeps the current sketch $\mathcal{S}$ plus a buffer $\mathcal{B}$ of unprocessed updates, both in the secret-shared form. New updates are appended to $\mathcal{B}$ until its size reaches s. When the buffer is full, the buffer is merged into the sketch and then emptied. The merge follows a group-then-process strategy. First, new updates and existing sketch cells are grouped by their (transformed) keys. Then, updates are applied to the corresponding sketch cells, ensuring the sketch remains fresh and valid. For linear sketches, both new updates and existing sketch cells are transformed via $h(\cdot)$. To merge, we sort the hash values of new updates and sketch cell indices, then update each cell by aggregating values in its segment using the prefix primitive. This approach resembles the join-then-aggregate idea in prior works [14].

When answering a query, we consider both the current sketch and the unprocessed updates in the buffer to ensure the result includes all data prior to the query. However, the procedures differ across sketches and query types. For union-preserving queries (e.g., frequency estimation), it suffices to evaluate the query on $\mathcal{S}$ and $\mathcal{B}$ independently then aggregate the results. For the non-union-preserving queries, the buffer must be merged into the sketch before querying to guarantee correctness, since updates in the buffer may alter the query result (e.g., distinct count and top-k).

In the following sections, we will design the Merge protocol for each sketch with $\tilde{O}(s)$ cost, which amortizes to $\tilde{O}(1)$ per update. We will also design the query procedure for each sketch so that a query can be answered with $\tilde{O}(s)$ cost. The bounds for various sketches are detailed in Table 1. We also compare with prior works that support updates and queries under the 3PC model without using sketching techniques. SECRECY [41] is a static MPC protocol. The bounds given here correspond to its naive adaptation to handling updates, which simply evaluates every query from scratch on all updates received so far. ShortCut [61] is the state-of-the-art 3PC protocol for supporting updates. Compared with ShortCut [61], we achieve a significant reduction in the update cost from $O(n)$ to $O(\log s)$, meaning that our approach is particularly suitable for scenarios where there are more updates than queries. In particular, we say that queries are sparse if there are no more than n/s queries in the stream, otherwise dense. When queries are sparse, the total cost of

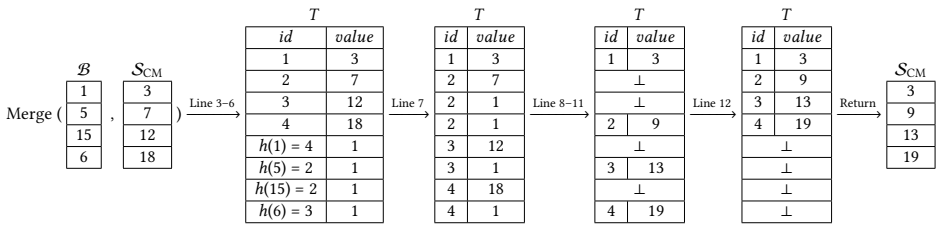


Fig. 1. An example of Count-Min sketch Merge protocol with $s = 4$

S^3 for processing a stream of n operations is $O(n \log s)$. On the other hand, ShortCut [61] incurs $O(n^2)$ total cost regardless of the sparsity of the queries. The only downside of our sketching-based approach is that the query results are approximate. However, we will see that if s is sufficiently large, we often see little to no errors. In the extreme case $s = n$, our approach also provides exact query results, while still being much more efficient than prior approaches. It is worth noting that although a larger sketch size s improves accuracy, it also increases query latency (i.e., response time of a query). Nevertheless, our framework achieves better latency than the other works due to the sketch-based design.

3.2 The Count-Min Sketch under S^3

Suppose each upload operation in the stream uploads a key-value pair (k, v), where the value can be either positive or negative. The Count-Min Sketch stores all the uploaded data compactly so as to support look-up queries that return the approximate total value associated with any given key. An important special case is when the value in each update is 1. In this case, the query returns the approximate count of the queried key, also known as the *frequency estimation* problem.

The Count-Min Sketch is a $\log(1/\delta) \times s$ array $\mathcal{S}_{\text{CM}}$, initialized to all zero. Each row employs a hash function h_i that maps the keys to $[s]$. When processing a key-value pair (k, v), the sketch is updated as $\mathcal{S}_{\text{CM}}[i][h_i(k)] \leftarrow \mathcal{S}_{\text{CM}}[i][h_i(k)] + v$, for $1 \le i \le \log(1/\delta)$. For a query on key k, the sketch returns $\min_{i=1}^{\log(1/\delta)} \{\mathcal{S}_{\text{CM}}[i][h_i(k)]\}$. It is known [18] that when each h_i is pairwise independent, this estimate incurs an error $\le V/s$ with probability at least $1 - \delta$, where V is the total value of all keys. Standard arguments show that if each h_i is a PRF, then the same error guarantees hold with a negligible change in the probability. We will only describe how to run one row of the Count-Min Sketch in the streaming-3PC model in this section; extending it to $\log(1/\delta)$ rows is straightforward. As mentioned above, it suffices to design the merge and query protocols.

The Merge protocol for the Count-Min Sketch is shown in Algorithm 2, with an example given in Figure 1. To avoid cluttering of notation, we drop the $[\![\cdot]\!]$ around all private data, which are always in the secret-sharing form. We only give the version for the frequency estimation problem, i.e., the value in each update is 1; extending it to arbitrary values is straightforward.

Let m be the buffer size. For the Count-Min sketch, we always have $m = s$ during a merge, but m may be less than s for the other sketches. The Merge protocol first creates a table T of two columns $(id, value)$ and $s + m$ rows. The first s rows contain the current sketch, and the next m rows contain the pairs $(h(k), 1)$ for each key k in the buffer. Then we sort T by $T.id$, so that rows with identical indices form contiguous segments. Next, using a segmented prefix-sum, we aggregate the values within each segment, where only the aggregated result for each index segment is maintained and other rows are marked as dummy. Finally, the servers compact the table by moving all dummy entries at the end, yielding the updated Count-Min Sketch by taking the $T.value$ of the first s tuples.

Algorithm 2: Count-Min Sketch Protocols

Protocol: `Merge`
Input: Buffer $\mathcal{B}$, CM sketch $\mathcal{S}_{\text{CM}}$
Output: Updated CM sketch $\mathcal{S}_{\text{CM}}$

$m \leftarrow |\mathcal{B}|, s \leftarrow |\mathcal{S}_{\text{CM}}|$;
Initialize $T(id, value)$ of size $s + m$;
for $i \leftarrow 1$ **to** s **do in parallel**
  $T[i] \leftarrow (i, \mathcal{S}_{\text{CM}}[i])$;
for $i \leftarrow 1$ **to** m **do in parallel**
  $T[s+i] \leftarrow (h(\mathcal{B}[i]), 1)$;
sort T by $T.id$;
$T.value \leftarrow$ prefix-sum of $T.value$ segmented by $T.id$;
for $i \leftarrow 1$ **to** $s + m - 1$ **do in parallel**
  if $T[i].id = T[i+1].id$ **then**
    $T[i] \leftarrow \perp$;
$T \leftarrow$ compact T;
return $\{T[i].value\}_{i=1}^{s}$

Protocol: `Query`
Input: Query key q, buffer $\mathcal{B}$, CM sketch $\mathcal{S}_{\text{CM}}$
Output: Query result r

$m \leftarrow |\mathcal{B}|, s \leftarrow |\mathcal{S}_{\text{CM}}|$;
$q' \leftarrow h(q)$;
$r \leftarrow \sum_{i=1}^{s} (\mathcal{S}_{\text{CM}}[i]$ if $i = q'$ else $0)$;
$r \leftarrow r + \sum_{i=1}^{m} (1$ if $\mathcal{B}[i] = q$ else $0)$;
return r

By plugging in the 3PC primitives of evaluating pseudorandom functions, sorting, prefix-sum, and compaction, we see that the merge protocol has $O(s \log s)$ cost and $O(\log s)$ rounds. This means that the amortized cost is $O(\log s)$ per update, as claimed in Table 1.

The query algorithm is given in Algorithm 2. For a query key q, the servers first compute $h(q)$ using PRF evaluation. Then we use multiplexers and summation to retrieve $r = \mathcal{S}_{\text{CM}}[h(q)]$. Finally, we scan the buffer and count how many times q appears there and add it to r, again using multiplexers and summation. This has $O(s)$ cost and $O(1)$ rounds, as claimed in Table 1.

The security of Count-Min Sketch under S^3 in the streaming-3PC model is directly implied via the *composition theorem* [17], as our protocol is a sequential composition of existing cryptographic primitives (e.g., circuit computation, sorting, compaction, etc.). The security of these primitives has been formally established in their respective works. We also provide formal proofs for all sketches we discussed using the real-ideal paradigm, which are included in our full version[2].

3.3 The HyperLogLog Sketch under S^3

Suppose each upload operation in the stream uploads a key k. The HyperLogLog Sketch stores all the uploaded keys compactly so that the number of distinct keys can be retrieved.

[2]https://github.com/hkustDB/StreamingMPC

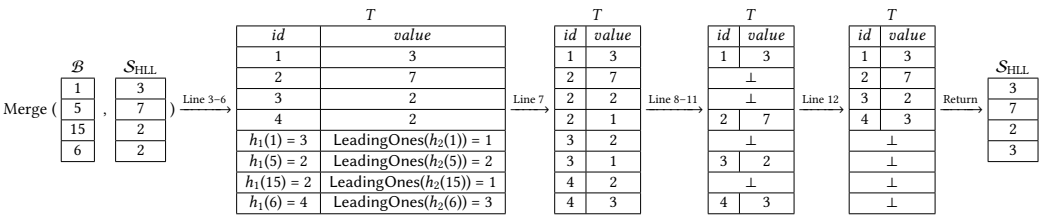


Fig. 2. An example of HyperLogLog sketch Merge protocol with $s = 4$

Algorithm 3: HyperLogLog Sketch Protocols

Protocol: `Merge`
Input: Buffer $\mathcal{B}$, HLL sketch $\mathcal{S}_{\text{HLL}}$
Output: Updated HLL sketch $\mathcal{S}_{\text{HLL}}$

$m \leftarrow |\mathcal{B}|, s \leftarrow |\mathcal{S}_{\text{HLL}}|$;
Initialize $T(id, value)$ of size $s + m$;
for $i \leftarrow 1$ **to** s **do in parallel**
  $T[i] \leftarrow (i, \mathcal{S}_{\text{HLL}}[i])$;
for $i \leftarrow 1$ **to** m **do in parallel**
  $T[s + i] \leftarrow (h_1(\mathcal{B}[i]), \text{LeadingOnes}(h_2(\mathcal{B}[i])))$
sort T by $T.id$;
$T.value \leftarrow$ prefix-max of $T.value$ segmented by $T.id$;
for $i \leftarrow 1$ **to** $s + m - 1$ **do in parallel**
  if $T[i].id = T[i + 1].id$ **then**
    $T[i] \leftarrow \perp$;
$T \leftarrow$ compact T;
return $\{T[i].value\}_{i=1}^{s}$

Protocol: LeadingOnes
Input: x
Output: Number of consecutive leading ones in x

$x_b \in \{0, 1\}^l \leftarrow$ binary representation of x in l bits;
$x' \leftarrow \text{prefix-and}(x_b)$;
$y \leftarrow \text{sum}(x')$;
return y

Protocol: `Query`
Input: Buffer $\mathcal{B}$, HLL sketch $\mathcal{S}_{\text{HLL}}$
Output: Query result r

$\mathcal{S}_{\text{HLL}} \leftarrow$ `Merge` $(\mathcal{B}, \mathcal{S}_{\text{HLL}})$;
$r \leftarrow \text{sum}(\mathcal{S}_{\text{HLL}})$
return r

The HyperLogLog Sketch is an array $\mathcal{S}_{\text{HLL}}$ containing s counters, initialized to all zero. The sketch employs two hash functions: h_1 maps the keys to $[s]$, while h_2 maps the keys to $[2^\ell]$, both of which

can be a PRF. When processing key k, the sketch is updated as $\mathcal{S}_{\text{HLL}}[h_1(k)] \leftarrow \max(\mathcal{S}_{\text{HLL}}[h_1(k)], \text{LeadingOnes}(h_2(k)))$, where LeadingOnes returns the number of consecutive leading ones in the binary representation. The sketch approximates the number of distinct keys by computing a normalized geometric or arithmetic[3] mean of all counter values (i.e., $r = \alpha_s s^2 (\sum_{i=1}^{s} 2^{-\mathcal{S}_{\text{HLL}}[i]})^{-1}$ where α_s is a constant). It is known [21, 25] that HyperLogLog achieves a relative error of $O(1/\sqrt{s})$ for approximating the distinct count, provided that the distinct count is less than 2^ℓ.

To run the HyperLogLog Sketch under S^3, we need to design the merge and query protocols when both the sketch and the buffer are stored in secret-shared form. The Merge protocol is shown in Algorithm 3, with an example given in Figure 2. The Merge protocol creates a table T of two columns *id*, *value* and $s+m$ rows, where the last m rows contain the pairs $(h_1(k), \text{LeadingOnes}(h_2(k))$ for each key k in the buffer. For the LeadingOnes protocol, it first performs a prefix-and circuit on the bits in the input x, resulting in consecutive ones at the front and zeroing all bits after the first zero. Subsequently, a sum circuit obtains the count of these ones which can be done locally. For example, consider a 4-bit number $x = 13 = (1101)_b$. The result of prefix-and is $[1, 1, 0, 0]$ and the sum is 2. Thus $\text{LeadingOnes}(13) = 2$. After getting the hash-based indexing and the number of consecutive leading ones, the following operation is similar to the CM sketch merge protocol, where values are grouped by their indices and aggregated by the maximum value accordingly. We conclude that the merge protocol for HyperLogLog Sketch has $O(s \log s)$ cost and $O(\log s)$ rounds by plugging in the complexity of 3PC primitives used in our protocols, as claimed in Table 1.

The query algorithm is given in Algorithm 3. Before calculating the estimated cardinality, the servers need to merge the elements in the buffer into the sketch because we cannot determine whether the keys in the buffer have appeared in the previous stream data. The estimator (query result) for HyperLogLog sketch is defined as follows:

$$\tilde{r} = \alpha_m s^2 (\textstyle\sum_{i=1}^{s} 2^{-\mathcal{S}_{\text{HLL}}[i]})^{-1}$$

The exponential computation in this estimator leads to high computation overhead in MPC setting. Instead, we use a LogLog estimator as an alternative solution, which is more MPC-friendly with a slightly worse accuracy [21]. Specifically, the protocol performs a sum circuit of all values in the sketch and returns it. The result r will then be post-processed in plaintext by the receiver to obtain the cardinality estimation as follows: $\tilde{r} = \alpha'_m s 2^{r/s}$. Since α'_m is a public correction constant and s is the sketch size, this formula is deterministic and invertible, making revealing r and $\tilde{r}$ equivalent from privacy perspectives. It is clear that query protocol is dominated by the merge operation, and also has $O(s \log s)$ cost and $O(\log s)$ rounds.

4 Secure Count-Based Sketches

In this section, we show how S^3 supports count-based sketches in the streaming-3PC model. A count-based sketch stores only a small set of keys (typically logarithmic in size) from a large universe. Each sketch cell acts as a counter associated with a selected key, and the cells are organized in a specific order.

Although there is no index-access problem as in linear sketches, maintaining the specific order introduces a new challenge. The naive approach of scanning the entire sketch is costly as well, so we still adopt the S^3 framework for batch update. We carefully design the Merge protocols to ensure the sketches remain valid. We present protocols for two representative count-based sketches: *SpaceSaving Sketch* [43] and *Greenwald-Khanna (GK) Sketch* [28, 29].

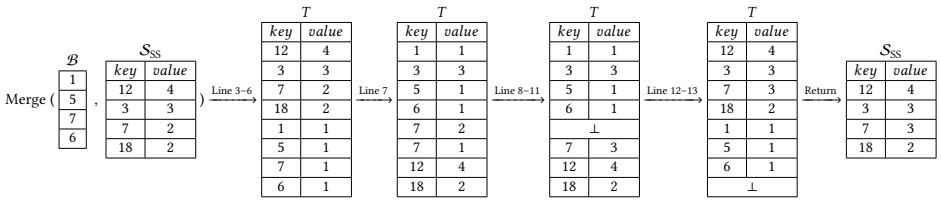


Fig. 3. An example of SpaceSaving sketch Merge protocol with $s = 4$

Algorithm 4: SpaceSaving Sketch Protocols

Protocol: `Merge`
Input: Buffer $\mathcal{B}$, SpaceSaving sketch $\mathcal{S}_{SS}$
Output: Updated SpaceSaving sketch $\mathcal{S}_{SS}$

$m \leftarrow |\mathcal{B}|, s \leftarrow |\mathcal{S}_{SS}|$;
Initialize $T(key, value)$ of size $s + m$;
for $i \leftarrow 1$ **to** s **do in parallel**
  $T[i] \leftarrow \mathcal{S}_{SS}[i]$;
for $i \leftarrow 1$ **to** m **do in parallel**
  $T[s + i] \leftarrow (\mathcal{B}[i], 1)$;
sort T by $T.key$;
$T.value \leftarrow$ prefix-sum of $T.value$ segmented by $T.key$;
for $i \leftarrow 1$ **to** $s + m - 1$ **do in parallel**
  if $T[i].key = T[i + 1].key$ **then**
    $T[i] \leftarrow \perp$;
$T \leftarrow$ compact T;
sort T on $T.value$;
return $\{T[i]\}_{i=1}^{s}$

Protocol: `Query`
Input: Query size c, Buffer $\mathcal{B}$, SpaceSaving sketch $\mathcal{S}_{SS}$
Output: Result $\mathcal{R}$

$\mathcal{S}_{SS} \leftarrow$ `Merge` $(\mathcal{B}, \mathcal{S}_{SS})$;
return $\{\mathcal{S}_{SS}[i]\}_{i=1}^{c}$

4.1 The Secure SpaceSaving Sketch

The SpaceSaving Sketch [43] maintains a collection $\mathcal{S}_{SS}$ of at most s key-value pairs $(k_i, v_i), i = 1, \ldots, s$. When processing key k, if there is an $k_i = k$, then it increments the associated value v_i by 1. Otherwise, it adds $(k, 1)$ to the collection if there is still room; if the sketch is full, it replaces the key with the minimum value, and increases its counter by 1. Like the Count-Min sketch, SpaceSaving also supports look-up queries that return the estimated frequency of any given key k: If there is a $k_i = k$, it returns v_i as an estimate of the frequency of k; if k is not maintained by the sketch, the estimate is just 0. It is known [2, 43] that the error of the estimate is at most n/s in the worst case, where n is the total number of keys of stream. Such a deterministic guarantee makes it the preferred

[3]This is sometimes called LogLog Sketch [21]

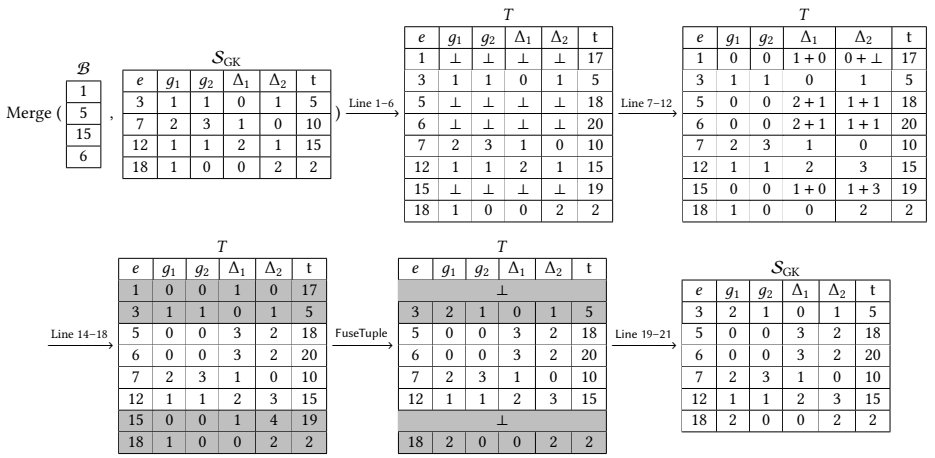


Fig. 4. An example of GK sketch Merge protocol starting from $n = 16$ with $s' = 4$ and $(n + m)/s = 2$. (Note that the size used in this example is for illustration purpose only and does not follow lemma 4.1.)

choice over Count-Min in many applications. In addition, SpaceSaving can also be used to answer top-c queries approximately by simply returning the c keys with the largest estimated frequencies. The SpaceSaving Sketch can also be extended to support a stream of key-value pairs (i.e., each key k has an associated value v), but the value must be positive (while Count-Min supports both positive and negative values).

By carefully designing merge and query protocol, we show how to run a SpaceSaving Sketch under S^3 for a stream of keys, where both the sketch and the buffer are stored in secret-shared form. During initialization, we put s dummy key-value pairs in $\mathcal{S}_{\text{SS}}$.

The Merge protocol for the SpaceSaving Sketch is shown in Algorithm 4, with an example given in Figure 3. The intuition of the merge protocol is that we can treat the elements in the buffer as another sketch, and merge the buffer with the sketch using the MergeableMinError algorithm in [2]. The merge operation in SpaceSaving Sketch is almost identical with the Count-Min sketch's merge protocol until Line 12, where the values for each key in T are aggregated using sorting and segmented prefix-sum primitives. After this aggregation, the servers sort T by the values and the first s tuples of T will be returned as the updated sketch. The correctness and error guarantees of updated sketch follows from [2]. It is easy to see that the merge has $O(s \log s)$ cost and $O(\log s)$ rounds.

To answer a query using the secure SpaceSaving sketch, the servers first perform the merge operation to flush the buffer. Then a look-up query can be answered in a way similar to the Count-Min sketch with cost $O(s)$ and $O(1)$ rounds. For a top-c query, we sort the sketch by value, and then return the c key-value pairs with the largest values, which has $O(s \log s)$ cost and $O(\log s)$ rounds.

4.2 The Secure Greenwald-Khanna Sketch

The Greenwald-Khanna (GK) Sketch [28, 29] is a compact data structure storing a set of keys in the data stream, where the keys are drawn from a totally ordered domain. It supports approximate rank queries, i.e., for any given key k, returns the number of keys in the stream that are smaller than k, as well as quantile queries which return a key whose rank is approximately equal to a given rank. The rank error for both types of queries is guaranteed to be at most n/s, for a parameter s set

Algorithm 5: GK Sketch Protocols

Public Parameters: Total number of data n, initial size s
Protocol: `Merge`
Input: Buffer $\mathcal{B}$, GK sketch $\mathcal{S}_{\text{GK}}$
Output: Updated GK sketch $\mathcal{S}_{\text{GK}}$

$m \leftarrow |\mathcal{B}|, s' \leftarrow |\mathcal{S}_{\text{GK}}|$;
Initialize $T(e, g_1, g_2, \Delta_1, \Delta_2, t)$ of size $s' + m$;
for $i \leftarrow 1$ **to** s' **do in parallel**
  $T[i] \leftarrow \mathcal{S}_{\text{GK}}[i]$;
for $j \leftarrow 1$ **to** m **do in parallel**
  $T[s+i] \leftarrow (\mathcal{B}[j], \perp, \perp, \perp, \perp, n+j)$;
sort T on $T.e$;
$(T.g_1, T.\Delta_1) \leftarrow \text{suffix-copy}(T.g_1, T.\Delta_1)$;
$(T.g_2, T.\Delta_2) \leftarrow \text{prefix-copy}(T.g_2, T.\Delta_2)$;
for $k \leftarrow 1$ **to** $s' + m$ **do in parallel**
  if $T[k].t > n$ **then**
    $T[k] \leftarrow (T[k].e, 0, 0, T[k].g_1 + T[k].\Delta_1 + 0, T[k].g_2 + T[k].\Delta_2 + 0, T[k].t)$
$n \leftarrow n + m$;
for 1 **to** $\lceil \log s' \rceil$ **do**
  for $i \leftarrow 1$ **to** $s' + m - 1$ **by** 2 **do in parallel**
    $T[i], T[i+1] \leftarrow \text{FuseTuple}(T[i], T[i+1])$;
  for $i \leftarrow 2$ **to** $s' + m - 1$ **by** 2 **do in parallel**
    $T[i], T[i+1] \leftarrow \text{FuseTuple}(T[i], T[i+1])$;
  $T \leftarrow$ compact T;
$s' \leftarrow \lceil 2s \ln(\frac{n}{s} + 2) + 2 \rceil$;
return $\{T[i]\}_{i=1}^{s'}$

Protocol: `Query`
Input: Query item q, buffer $\mathcal{B}$, GK sketch $\mathcal{S}_{\text{GK}}$
Output: Query result r

$m \leftarrow |\mathcal{B}|, s' \leftarrow |\mathcal{S}_{\text{GK}}|$;
$r_{min} \leftarrow 0, r_{max} \leftarrow 0, r' \leftarrow 0$;
$T \leftarrow$ prefix-sum of $\{\mathcal{S}_{\text{GK}}[i].g_1 + \mathcal{S}_{\text{GK}}[i].g_2 + 1\}_{i=1}^{s'}$;
for $i \leftarrow 1$ **to** $s' - 1$ **do in parallel**
  if $\mathcal{S}_{GK}[i].e \le q$ **and** $\mathcal{S}_{GK}[i+1].e > q$ **then**
    $r_{min} \leftarrow T[i] - \mathcal{S}_{\text{GK}}[i].\Delta_2 - \mathcal{S}_{\text{GK}}[i].g_2$;
    $r_{max} \leftarrow T[i] + \mathcal{S}_{\text{GK}}[i+1].\Delta_1 + \mathcal{S}_{\text{GK}}[i+1].g_1$
$r' \leftarrow \sum_{i=1}^{m} (1 \text{ if } \mathcal{B}[i] \le q \text{ else } 0)$;
return $(r_{min} + r_{max})/2 + r'$
return R_1, R_2

Algorithm 6: GK Sketch Protocols Cont.

Protocol: FuseTuple
Input: Two tuples R_1, R_2
Output: Two updated tuples R_1, R_2

```
if R_1 ≠ ⊥ and R_2 ≠ ⊥ then
    if R_1.t > R_2.t then
        if R_1.g_1 + R_1.g_2 + R_2.g_1 + R_2.Δ_1 + 1 ≤ n/s then
            R_1 ← ⊥, R_2.g_1 ← R_1.g_1 + R_1.g_2 + R_2.g_1 + 1;
    else
        if R_2.g_1 + R_2.g_2 + R_1.g_2 + R_1.Δ_2 + 1 ≤ n/s then
            R_1.g_2 ← R_2.g_1 + R_2.g_2 + R_1.g_2 + 1, R_2 ← ⊥ ;
```

at the beginning of the stream. The Greenwald-Khanna Sketch can also be extended to a weighted version that supports a stream of key-value pairs (the values must be positive). In this case, the rank of a key k is defined to be the total value of keys less than k. By asking two rank queries, the sketch can also answer range-count or range-sum (sum of all values) queries over a range of keys. We focus on the unweighted version in this paper; extending to the weighted version requires some additional efforts but the basic idea is the same.

The sketch maintains a sorted list of tuples $\mathcal{S}_{\text{GK}}$, representing disjoint ranges of the key domain. A tuple stores a representative key e from this range, along with some auxiliary information. When a new key arrives, the sketch first inserts it as a new tuple in the sorted list, then fuses adjacent tuples when certain conditions are met to maintain space efficiency. In the original version [28], each tuple maintains two attributes in addition to the key: g_1, representing the number of keys in the range, and Δ_1, capturing the uncertainty in the rank of the representative key. The original Greenwald-Khanna paper [28] has described two fusion algorithms: a complicated one guaranteeing a sketch size upper bound of $O(s \log(n/s))$ with a large hidden constant, and a practical one but with no sketch size bound, neither of which is appropriate under MPC.

Recently, a new version of the sketch [29] has been proposed, which is both practical and has a provable sketch upper bound. Each tuple now has four attributes, $g_1, g_2, \Delta_1, \Delta_2$: g_1 is the number of keys in the range that are smaller than e, g_2 is the number of keys greater than e, Δ_1 is the total number of possible keys greater than e in the preceding tuples, and Δ_2 is the total number of possible items smaller than e in the subsequent tuples. Additionally, it records the timestamp t when the representative key e was inserted into the sketch. The size of the new version of the GK sketch has shown to be bounded by $O(s \log(n/s))$ [29]. In the MPC setting, we will need an exact size upper bound, as we will need to pad dummy tuples so that its size only depends on s and n but not the actual stream content. We show the exact size bound in the following lemma.

Lemma 4.1. *The size of the GK sketch when running with parameter s is at most $2s \ln(\frac{n}{s} + 2) + 2$ when the stream length is n.*

Proof. Following the potential function defined in [29] , for any i-th update data x_i inserted before n, let

$$P_{x_i}(n) = \frac{1}{1+\frac{n-i}{s}}$$

The total number of non-fusible tuples (i.e. the size) in GK sketch is bounded as follow,

$$2\sum_{i=1}^{n} P_{x_i}(n) = \sum_{i=1}^{n} \frac{2}{1+\frac{i}{s}} \leq 2s \sum_{l=1}^{1+\lceil \frac{n}{s} \rceil} (\frac{1}{l}) \leq 2s \ln(\frac{n}{s} + 2) + 2$$

□

Below we show how to run GK sketch under S^3 by describing the merge and query protocols. For the GK sketch, we replace the sketch size parameter s with $s' = \lceil 2s \ln(\frac{n}{s} + 2) + 2 \rceil$. Note that, unlike the other secure sketches which have a fixed size s, the size of the secure GK sketch grows mildly with n/s.

4.2.1 Merge Protocol. The Merge protocol is shown in Algorithm 5, with an example given in Figure 4. We first create a new tuple for each key in the buffer. Its g_1 and g_2 will be set to 0. For Δ_1, we need to calculate the number of keys in the sketch which are possibly greater than the new key. This equals to $g_1 + \Delta_1$ of the following tuple in the current GK sketch. Similarly, we set Δ_2 as $g_2 + \Delta_2$ of the preceding tuple. Note that all new tuples between two existing tuples have the same Δ value, since there is no uncertainty introduced by these new tuples. In our protocol, we can set Δ_1 for the new tuples by using a suffix-copy circuit and Δ_2 by using a prefix-copy circuit. While all these values are secret-shared, the timestamp is public information, which can be stored in plaintext in the tuple.

After setting the attributes for new tuples, we fuse tuples in the GK sketch using the FuseTuple protocol, which realizes the same fusion criteria in the plaintext version [29] but using a circuit that performs $\lceil \log s' \rceil$ rounds of compression. Each round checks all adjacent tuples in the sketch, and fuses all eligible pairs (i.e., Line 15-19 in Algorithm 5). The following lemma shows that $\lceil \log s' \rceil$ rounds of compression is sufficient to complete all the fuse operations, so that the resulting sketch is guaranteed to be no larger than s'.

Lemma 4.2. *All eligible pairs can be fused within $\lceil \log s' \rceil$ rounds of compression.*

Proof. After a merge operation, there are s' new tuples in the sketch. In the worst case, these newly inserted tuples are consecutive where all $s' - 1$ pairs satisfy the fusion condition. However, due to the constraint that no two overlapping pairs can be fused simultaneously (since each tuple can participate in at most one fusion per round), we can fuse at most $\lfloor s'/2 \rfloor$ pairs in the first round. Furthermore, these $s' - 1$ pairs can be fused into one single tuple when the fusing threshold n/s is greater than s'. After the first round we have at most $\lceil s'/2 \rceil$ remaining tuples that could be fused, and similarly $\lceil s'/4 \rceil$ after the second round, and so on. Therefore, at most $\lceil \log s' \rceil$ rounds are required to complete all eligible fusions in the worst case. □

Finally, we recalculate s' and retain the first s' tuples in T as the updated GK sketch. For the correctness and error guarantees of updated GK sketch, we compare our batch insertion algorithm with sequential insertion in plaintext [29] which fuses at most one tuple pair after a single update. First, we fuse at least as much tuples compared with plaintext version by deferring tuple fusions after a batch update, and thus the size s' satisfies lemma 4.1. Second, our fusing conditions are identical with [29], and thus the approximation error for both insertion algorithms will be bounded by the same fusing threshold.

Merge operation has $\lceil \log s' \rceil$ rounds of compression where each round involves a compaction primitive and a series of multiplexers. By the complexity of the 3PC primitives used including sorting, prefix-sum, we conclude that Merge protocol has $O(s' \log s')$ cost and $O(\log s')$ rounds.

4.2.2 Query Protocol. For rank query estimation, GK scans the list of tuples and finds where the query element would fall in the sketch, then processes the tuples sequentially using stored information to provide approximate answers. In the Query protocol, partial rank results from the sketch and buffer can be calculated independently and then securely combined. First, for the sketch

results, we find the entry $\mathcal{S}_{\text{GK}}[i]$ whose representative element e_i is the largest element less than the query item q and the entry $\mathcal{S}_{\text{GK}}[i+1]$ whose representative element e_{i+1} is the smallest element greater than the query item q. Then we can calculate minimum possible rank

$$r_{min} = i + \sum_{j=1}^{i} (\mathcal{S}_{\text{GK}}[j].g_1) + \sum_{j=1}^{i-1} (\mathcal{S}_{\text{GK}}[j].g_2) - \mathcal{S}_{\text{GK}}[i].\Delta_2,$$

and maximum possible rank

$$r_{max} = i + \sum_{j=1}^{i+1} (\mathcal{S}_{\text{GK}}[j].g_1) + \sum_{j=1}^{i} (\mathcal{S}_{\text{GK}}[j].g_2) + \mathcal{S}_{\text{GK}}[i+1].\Delta_1.$$

For the buffer result r', we securely count the number of elements in the buffer less than the query item q by using multiplexers. The final result will be $r = \frac{r_{min}+r_{max}}{2} + r'$. By the complexity of prefix-sum circuits and a series of multiplexers, Query protocol has $O(s')$ cost and $O(1)$ rounds. It is clear that the cost for updates and queries are bounded by the last running size s'. Therefore, GK sketch has $O(ns')$ total cost for dense query pattern, and $O(n \log s')$ total cost for sparse query pattern as claimed in Table 1.

5 Extensions

5.1 General Streaming Setting

While this paper focuses on insertion operations, our approach can be extended to handle deletions in streaming setting. A straightforward solution is maintaining two identical sketches to handle insertions and deletions separately. Alternatively, certain sketch algorithms naturally support deletions within a single data structure. For example, the Count-Min sketch can handle deletions (i.e., negative values) by applying the same Update protocol.

For simplicity, we assume a single message per timestamp, though our approach can be generalized to multiple messages per timestamp. Previous works [52, 53] have identified that update patterns (i.e., the number of messages at each timestamp) may cause information leakage. They have also proposed differentially private mechanisms (i.e., adding a sufficient number of dummy messages that satisfy DP) to protect this information. When such protection is needed, and the solutions in these works are compatible with our framework.

5.2 Private Query Release

Differential privacy (DP) [22] has become the standard notion for private data release, due to its strong protection of individual information. When data owners want to release query results in public without revealing additional sensitive information, they can apply DP mechanisms by adding calibrated noise to the outputs. Extensive research has explored DP mechanisms for various sketch data structures, including Count-Min sketches [35, 60], HyperLogLog sketches [33, 34], and GK sketches [3]. These approaches typically add noise proportional to the sensitivity of the sketch to ensure privacy and our framework can easily support them by securely sampling differentially private noise [26] under 3PC. Additionally, continual observation [20, 23] is particularly relevant to our streaming setting, as query results are released continuously as the data stream evolves.

5.3 More Applications

S^3 provides a general framework that can incorporate various sketch data structures besides the four sketches we implement, such as Bloom filters [15] for membership queries and Q-digest [51] for rank/quantile queries.

Algorithm 7: Optimized Segmented Prefix-Sum then Compact Protocol

Public Parameters: Group size s
Input: Two arrays X, Y of size n
Output: Aggregated result of size s

```
1  Initialize T of size n of 0's;
2  for i ← 1 to n do in parallel
3      if y[i] ≠ y[i + 1] then
4          T[i] ← 1;
5  Convert X to additive shares;
6  X' ← prefix-sum of X;
7  X' ← compact X' by T;
8  for i ← 2 to s do in parallel
9      X'[i] ← X'[i] − X'[i − 1];
10 Covert X' to boolean shares;
11 return {X'[i]}_{i=1}^{s}
```

Sampling, another AQP technique, can also be applied within our framework. For instance, reservoir sampling [37] can be implemented where servers maintain a sample of fixed size s and update this sample as stream data arrives.

6 Experiments

6.1 Implementation and Optimization

We have implemented our protocols in Java[4] with the following practical optimizations.

6.1.1 Segmented Prefix-sum then Compaction. In the CM sketch Merge protocol, there is a "segmented prefix-sum then compaction" operation (i.e., Line 8-12 in Algorithm 2) to obtain the total sum for each group. Since additive shares can perform addition (and subtraction) locally without any communication, we convert the values to additive shares and compute the prefix-sum of the whole array X[5]. After compaction, we use another local subtraction to compute the exact sum for each segment defined by Y. This optimization does not change the asymptotic cost, but reduces the round complexity from $O(\log n)$ to $O(1)$.

See an example of $X = (4, 1, 1, 3, 2, 6)$ and $Y = (a, a, a, b, b, c)$. The tag sequence T is $(0, 0, 1, 0, 1, 1)$ and the prefix-sum of X computes $X' = (4, 5, 6, 9, 11, 17)$ locally. The compaction results $X' = (6, 11, 17)$, which is exactly the prefix-sum of aggregated result in each segment. And the local subtraction reveals the sum in each segment, which is $(6, 5, 6)$.

6.1.2 Sorting. [30] provides a 3PC sorting protocol based on radix sort, which can be implemented with prefix-sum and compaction primitives. The costs of [5] and [30] are asymptotically the same on the bit level $O(ln \log n)$, while [5] has $O(\log n)$ rounds and [30] has $O(l)$ rounds. In practice, [30] has a better performance and we adopt it in our implementation.

6.1.3 Top-k Selection. The second sort protocol in the SpaceSaving Merge protocol (Line 13 in Algorithm 4) retrieves the largest s count values in the sketch, which can be optimized by a secure Top-k operator [55] in $O(n)$ cost and $O(\log n)$ rounds, which is based on quickselect. However,

[4]Codes are available at https://github.com/alibaba-edu/mpc4j.
[5]See the conversion details and experimental results in our full version.

since the sketch is not ordered by the count value in the merge operation, the Query protocol needs an extra secure Top-k operator to obtain the first c elements in the sketch as the query result.

6.2 Experiment Setup

We conduct experiments on three servers, each equipped with 32GB of memory and an Intel(R) Xeon(R) Platinum 8272CL CPU. The experiments are conducted in LAN setting by default, where the network delay is around 2ms and the bandwidth is around 2Gb/s. We also simulate WAN setting by setting network delay as 20ms and bandwidth as 200Mb/s using Linux tc command. We set computational security parameter $\kappa = 128$ and the statistical security parameter $\sigma = 40$. We use boolean shares for stream data and sketches by default and the bit length of the keys is 32.

6.2.1 Baselines. We choose ShortCut [61], the only existing MPC protocol that supports both updates and queries. We compare our secure CM sketch, HLL sketch, and SS sketch protocols with ShortCut's update protocol for group-by materialized tables, which handle queries for aggregated statistics; and our secure GK sketch protocol with ShortCut's update protocol for order-by materialized tables, which handle queries for ordered statistics.

We also include exact query evaluation from raw data as a baseline for the mixed updates and queries scenarios. We focus on two query types: *linear scan* and *group by*. Specifically, we will compare Count-Min Sketch against a linear scan baseline [31] for lookup queries, where the protocol securely scans the full data stream and computes frequencies via equality tests. Group by aggregation queries will be compared with HyperLogLog and SpaceSaving, and Greenwald-Khanna sketches. For these comparisons, the protocol securely sorts the data by keys then follows a prefix circuit to compute aggregated values which aligns with the group by protocols in several prior works [7, 8, 14, 31, 41].

6.2.2 Metrics. We evaluate (1) the communication cost among all three servers in kilobytes (KB); (2) the running time including both server computation time and network data transmission time in milliseconds (ms); and (3) the absolute error of all queries, and the recall rate in Table 6 to show the quality of top-k queries in SpaceSaving Sketch. For all metrics, We repeat the experiments 10 times and report the average after removing the largest and smallest values. n denotes the total stream size and s denotes the sketch size which are set to a power of 2. We report amortized cost and time, calculated as the total cost and time divided by the number of messages.

Remark 1. We omit the time intervals between data arrivals in our experiments; however, these intervals also affect real-world performance. The amortized running time reflects the minimum processing time the system requires per data item. If the average inter-arrival time is greater than this amortized time, the system can process update and query operations in real time.

Remark 2. We implement the secure Count-Min sketch using 7 rows in our experiments, which guarantees $O(n/s)$ absolute error with high probability ($> 99\%$).

6.2.3 Dataset. We use both synthetic and real datasets and all keys are drawn from a domain of $[2^{32}]$. The synthetic data is sampled from the following distributions:

- Gaussian distribution (Gaus.): $\mu = 2^{31}, \sigma = 2^{13}$;
- Uniform distribution (Unif.).

The two real datasets, using one attribute of each:

- AOL-user-ct-collection (AOL) [49]: a collection of real-world website accesses with userID and clicking time. We hash the website of each record into 32-bit as input data.
- Netflix-Prize (Netf.) [1]: contains a set of rating scores with rating time given by users to movies. We use the user ID number as input data.

	$n = 2^{14}$			$n = 2^{17}$			$n = 2^{20}$			$n = 2^{23}$		
	Time		Comm.	Time		Comm.	Time		Comm.	Time		Comm.
	LAN	WAN		LAN	WAN		LAN	WAN		LAN	WAN	
ShortCut	27.04	880.63	254.73	52.82	1365.00	2369.26	570.00	2118.00	38146.30	3119.00	8686.00	305139.90
S^3-CM	0.42	2.74	35.03	0.43	2.73	35.03	0.42	2.72	35.03	0.42	2.72	35.03
S^3-HLL	0.10	0.75	5.30	0.10	0.74	5.30	0.10	0.73	5.30	0.10	0.73	5.30
S^3-SS	0.16	1.46	10.30	0.15	1.45	10.30	0.16	1.46	10.30	0.16	1.45	10.30

Table 2. Amortized time and communication costs per update of different sketches for aggregated statistics ($s = 2^{14}$)

	$n = 2^{10}$			$n = 2^{14}$			$n = 2^{17}$			$n = 2^{20}$		
	Time		Comm.	Time		Comm.	Time		Comm.	Time		Comm.
	LAN	WAN		LAN	WAN		LAN	WAN		LAN	WAN	
ShortCut	42.44	1419.73	43.49	49.84	1583.00	522.41	154.00	1891.00	9640.75	943.00	3522.00	77077.31
S^3-GK	1.92	27.26	48.74	1.92	27.14	48.74	1.63	22.08	51.74	1.46	20.02	52.89

Table 3. Amortized time and communication costs per update of different sketches for ordered statistics ($s = 2^{10}$)

6.3 Experimental Results

6.3.1 Update Only. In this part, we conduct experiments for the update-only scenario. The experimental results for aggregated statistics are shown in Table 2, and for ordered statistics are shown in Table 3. The total running time for ShortCut exceeds 1 day for long stream experiments (i.e., $n \geq 2^{20}$ in Table 2 and $n \geq 2^{17}$ in Table 3), thus we only report the cost of the last update in the stream.

The amortized running time and communication cost for sketches in Table 2 remain almost the same throughout the entire stream, which is $O(\log s)$ and only depends on the sketch size. Specifically, our protocol has an amortized time of approximately 1ms and communication cost of less than 40KB per update in LAN setting. However, ShortCut has a linear increase as stream data grows, with time and cost increasing to around 3s and 300MB when $n = 2^{23}$. At this size, the performance gap between our approach and ShortCut is on the order of thousands. In the WAN setting, S^3 has a similar advantage over baseline, although all methods incur longer running times compared to the LAN setting because of the worse network latency and bandwidth.

The size of the GK sketch is slightly increasing as data arrives (i.e., $s' = \lceil 2s \ln(\frac{n}{s} + 2) + 2 \rceil$), which explains the small increase in amortized communication cost in Table 3. Interestingly, the amortized time decreases as data size increases due to fewer communication rounds. As the sketch size grows, the number of Merge protocols decreases, reducing network transmission overhead and improving overall efficiency. The experiment results in Table 3 show the efficiency of our sketch protocols for order statistics with amortized 2ms time and 50KB communication cost in LAN setting. This result of S^3 significantly outperforms ShortCut up to thousand times in both LAN and WAN setting.

6.3.2 Mixed Updates and Queries. Table 4 shows the results in LAN setting for different query patterns and query types. There are $2^{20} \approx 10^6$ operations in the stream, where a query is performed for every $10^2, 10^3, 10^4, 10^5$ and 10^6 updates, respectively, i.e., the number of queries is $v = 10^4, 10^3, 10^2$,10 and 1. Since ShortCut cannot handle more than 2^{20} updates, their results are not shown in Table 4.

For Count-Min sketch, we estimate the frequency of the given key by look-up queries. The query cost is linear to the number of queries (i.e., $v \cdot O(s)$) as shown in the results. We also observe that the cost increases slightly as the sketch size increases, which corresponds to the $O(s)$ query cost

	$v = 10^4$		$v = 10^3$		$v = 10^2$		$v = 10^1$		$v = 1$	
	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.	Time	Comm.
Linear Scan	8.76	502.93	0.88	49.82	0.09	4.68	0.01	0.45	0.01	0.04
S^3-CM ($s = 2^{13}$)	1.26	54.94	0.52	35.23	0.44	33.26	0.42	33.42	0.42	33.04
S^3-CM ($s = 2^{14}$)	1.52	79.87	0.54	40.28	0.44	36.32	0.43	35.56	0.42	35.03
S^3-CM ($s = 2^{15}$)	1.97	127.35	0.60	47.65	0.45	39.66	0.45	38.42	0.44	37.83
Group By	463.46	25566.62	46.38	2558.96	4.68	258.20	0.51	28.12	0.09	5.12
S^3-HLL ($s = 2^{14}$)	8.06	390.46	0.88	41.79	0.14	7.04	0.12	5.39	0.10	5.30
S^3-SS ($s = 2^{14}$)	18.56	871.81	1.93	92.04	0.29	13.83	0.19	10.90	0.16	10.30
S^3-GK ($s = 2^{10}$)	6.29	340.77	2.19	80.62	1.68	54.62	1.64	52.02	1.63	51.74

Table 4. Amortized time and communication costs per element for different query number ($n = 2^{20}$)

	$s = 2^{14}$			$s = 2^{15}$			$s = 2^{16}$			$s = 2^{17}$		
	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error
S^3-CM	0.42	35.03	50.95	0.44	37.83	23.25	0.47	38.55	10.43	0.57	41.39	4.27
S^3-HLL	0.10	5.30	7945.62	0.10	5.71	8596.78	0.11	5.82	5327.69	0.12	6.22	5767.19
S^3-SS	0.16	10.30	0.98	0.16	10.32	0.98	0.16	10.36	0.94	0.17	10.37	0.94
	$s = 2^8$			$s = 2^9$			$s = 2^{10}$			$s = 2^{11}$		
	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error
S^3-GK	1.78	45.01	3045.00	1.77	47.21	1548.00	1.47	51.94	312.00	1.34	53.15	164.00

Table 5. Trade-off between the approximation error and running time/communication cost ($n = 2^{20}$)

	Gaus.			Unif.			AOL.			Netf.		
	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error	Time	Comm.	Error
S^3-CM ($s = 2^{14}$)	0.42	35.03	9.18	0.42	35.03	50.69	0.41	35.03	20.32	0.42	35.03	36.76
S^3-HLL ($s = 2^{14}$)	0.10	5.30	1336.37	0.11	5.30	9796.70	0.10	5.30	1878.36	0.10	5.30	2891.78
S^3-SS ($s = 2^{14}$)	0.16	10.34	8.11 | 87%	0.16	10.34	1.00 | 6%	0.16	10.34	1.83 | 99%	0.17	10.34	4.79 | 82%
S^3-GK ($s = 2^{10}$)	1.47	51.94	299.00	1.45	51.94	282.00	1.47	51.94	677.00	1.46	51.94	311.00

Table 6. Cost and approximation error for different sketches on different datasets ($n = 2^{20}$)

and $O(\log s)$ amortized update cost. Compared with the update-only ShortCut protocol which has amortized 500ms time and 2MB communication cost, we only have amortized time and cost less than 2ms and 150KB for all query patterns.

We also estimate distinct counts using HyperLogLog sketch, top-1000 elements using SpaceSaving sketch, and ranks of the query elements using GK sketch. We can see that the increase in SpaceSaving sketch and HyperLogLog sketch costs is greater than the increase in Count-Min sketch and GK sketch, since the query cost is $v \cdot O(s \log s)$. Nevertheless, the total cost using sketches still outperform ShortCut even in the update-only scenario.

Our sketch-based solution also demonstrates scalability when processing large volumes of query messages compared to exact query evaluation baselines. The computational costs for the linear scan and group by scale linearly with the number of query messages v by complexities of $v \cdot O(n)$ and $v \cdot O(n \log n)$ respectively. Thus, these baselines become computationally expensive as the query load increases. As a result, S^3 stands out as the only framework that can efficiently process both updates and queries in data streams.

6.3.3 Approximation error. We show that the approximation error is minimal compared with the significant efficiency gains by using sketches under S^3. In Table 5, we perform the sketch update

protocol on streaming data which follow a uniform distribution with different sketch sizes. The amortized cost for each update is $O(\log s)$, and we see a logarithmic increase in both running time and communication cost as the sketch size increases. Meanwhile, all of sketches have provable error bounds proportional to $O(1/s)$ or $O(1/\sqrt{s})$. Therefore, there is a decrease in the approximation error as sketch size increases for CM sketch, HLL sketch and GK sketch. SpaceSaving sketch has a poor performance on uniform dataset and approximation error barely change in Table 5, while it is more powerful for skewed dataset including Gaussian dataset and real datasets as shown in Table 6. We see that it has a high recall rate for the top-1000 query and small absolute error for their frequency estimations.

Since all MPC protocols are data-independent, their running time and communication costs depend solely on the sketch size and the actual data values and their distribution do not affect protocol performance. Table 6 shows that all sketches have the same running time and communication cost across all datasets.

7 Conclusions

This paper introduces a framework to support a variety of sketches under the streaming-MPC model. The secure sketches can all be updated in $O(\log s)$ time amortized, and can be used to answer many different types of queries with error guarantees. They also work very well in practice, as demonstrated by our experimental study using three non-colluding servers, providing a viable approach to query answering on outsourced streaming data. One interesting future direction is how to support joins, which would significantly expand the class of queries that can be supported.

Acknowledgments

This work is supported by HKRGC under grants 16205422, 16204223, and 16203924; and by the HKUST-Alibaba Joint Laboratory on Big Data and Artificial Intelligence. We also thank the anonymous reviewers for valuable suggestions on improving the paper.

References

[1] [n. d.]. Netflix Prize dataset. https://archive.org/download/nf_prize_dataset.tar. Accessed: 2016-02-04.

[2] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *ACM Trans. Database Syst.* 38, 4, Article 26 (Dec. 2013), 28 pages. https://doi.org/10.1145/2500128

[3] Daniel Alabi, Omri Ben-Eliezer, and Anamay Chaturvedi. 2022. Bounded Space Differentially Private Quantiles. arXiv:2201.03380 https://arxiv.org/abs/2201.03380

[4] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 430–454.

[5] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 610–629. https://doi.org/10.1145/3460120.3484560

[6] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. 2022. Efficient Secure Three-Party Sorting with Applications to Data Analysis and Heavy Hitters. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 125–138. https://doi.org/10.1145/3548606.3560691

[7] Gilad Asharov, Koki Hamada, Ryo Kikuchi, Ariel Nof, Benny Pinkas, and Junichi Tomida. 2023. Secure Statistical Analysis on Multiple Datasets: Join and Group-By. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 3298–3312. https://doi.org/10.1145/3576915.3623119

[8] Saikrishna Badrinarayanan, Sourav Das, Gayathri Garimella, Srinivasan Raghuraman, and Peter Rindal. 2022. Secret-Shared Joins with Multiplicity from Aggregation Trees. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 209–222. https://doi.org/10.1145/3548606.3560670

[9] Kassem Bagher and Shangqi Lai. 2023. SGX-Stream: A Secure Stream Analytics Framework In SGX-enabled Edge Cloud. *Journal of Information Security and Applications* 72 (2023), 103403. https://doi.org/10.1016/j.jisa.2022.103403

[10] K. E. Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (Atlantic City, New Jersey) *(AFIPS '68 (Spring))*. Association for Computing Machinery, New York, NY, USA, 307–314. https://doi.org/10.1145/1468075.1468121

[11] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2017. SMCQL: secure querying for federated databases. *Proc. VLDB Endow.* 10, 6 (Feb. 2017), 673–684. https://doi.org/10.14778/3055330.3055334

[12] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient SQL query processing in differentially private data federations. *Proc. VLDB Endow.* 12, 3 (Nov. 2018), 307–320. https://doi.org/10.14778/3291264.3291274

[13] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. SAQE: practical privacy-preserving approximate query processing for data federations. *Proc. VLDB Endow.* 13, 12 (July 2020), 2691–2705. https://doi.org/10.14778/3407790.3407854

[14] Eli Baum, Sam Buxbaum, Nitin Mathai, Muhammad Faisal, Vasiliki Kalavri, Mayank Varia, and John Liagouris. 2025. ORQ: Complex Analytics on Private Data with Strong Security Guarantees. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles* (Lotte Hotel World, Seoul, Republic of Korea) *(SOSP '25)*. Association for Computing Machinery, New York, NY, USA, 802–833. https://doi.org/10.1145/3731569.3764833

[15] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (July 1970), 422–426. https://doi.org/10.1145/362686.362692

[16] Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2361–2377. https://doi.org/10.1145/3460120.3484557

[17] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptol.* 13, 1 (Jan. 2000), 143–202. https://doi.org/10.1007/s001459910006

[18] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[19] Binyang Dai, Xiao Hu, and Ke Yi. 2024. Reservoir Sampling over Joins. *Proc. ACM Manag. Data* 2, 3, Article 118 (May 2024), 26 pages. https://doi.org/10.1145/3654921

[20] Wei Dong, Qiyao Luo, and Ke Yi. 2023. Continual Observation under User-level Differential Privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 2190–2207. https://doi.org/10.1109/SP46215.2023.10179466

[21] Marianne Durand and Philippe Flajolet. 2003. Loglog Counting of Large Cardinalities. In *Algorithms - ESA 2003*, Giuseppe Di Battista and Uri Zwick (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 605–617.

[22] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[23] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing* (Cambridge, Massachusetts, USA) *(STOC '10)*. ACM, New York, NY, USA, 715–724. https://doi.org/10.1145/1806689.1806787

[24] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. A Pragmatic Introduction to Secure Multi-Party Computation. *Found. Trends Priv. Secur.* 2, 2–3 (Dec. 2018), 70–246. https://doi.org/10.1561/3300000019

[25] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *DMTCS Proceedings (DMTCS Proceedings, Vol. DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07))*, Philippe Jacquet (Ed.). Discrete Mathematics and Theoretical Computer Science, Juan les Pins, France, 137–156. https://doi.org/10.46298/dmtcs.3545

[26] Yucheng Fu and Tianhao Wang. 2024. Benchmarking Secure Sampling Protocols for Differential Privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 318–332. https://doi.org/10.1145/3658644.3690257

[27] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. 2012. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 513–524. https://doi.org/10.1145/2382196.2382251

[28] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (Santa Barbara, California, USA) *(SIGMOD '01)*. Association for Computing Machinery, New York, NY, USA, 58–66. https://doi.org/10.1145/375663.375670

[29] Elena Gribelyuk, Pachara Sawettamalya, Hongxun Wu, and Huacheng Yu. 2024. Simple & Optimal Quantile Sketch: Combining Greenwald-Khanna with Khanna-Greenwald. *Proc. ACM Manag. Data* 2, 2, Article 109 (May 2024), 25 pages. https://doi.org/10.1145/3651610

[30] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. 2013. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Information Security and Cryptology – ICISC 2012*, Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 202–216.

[31] Feng Han, Lan Zhang, Hanwen Feng, Weiran Liu, and Xiangyang Li. 2022. Scape: Scalable Collaborative Analytics System on Private Database with Malicious Security. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1740–1753. https://doi.org/10.1109/ICDE53745.2022.00176

[32] D. Harris. 2003. A taxonomy of parallel prefix networks. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. 2213–2217 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292373

[33] Jonathan Hehir, Daniel Ting, and Graham Cormode. 2023. Sketch-flip-merge: mergeable sketches for private distinct counting. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 522, 20 pages.

[34] Changhui Hu, Jin Li, Zheli Liu, Xiaojie Guo, Yu Wei, Xuan Guang, Grigorios Loukides, and Changyu Dong. 2021. How to Make Private Distributed Cardinality Estimation Practical, and Get Differential Privacy for Free. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 965–982. https://www.usenix.org/conference/usenixsecurity21/presentation/hu-changhui

[35] Ziyue Huang, Yuan Qiu, Ke Yi, and Graham Cormode. 2022. Frequency estimation under multiparty differential privacy: one-shot and streaming. *Proc. VLDB Endow.* 15, 10 (June 2022), 2058–2070. https://doi.org/10.14778/3547305.3547312

[36] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. 2016. Quickr: Lazily Approximating Complex Ad-Hoc Queries in Big Data Clusters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2016)* (proceedings of the acm sigmod international conference on management of data (sigmod 2016) ed.). ACM - Association for Computing Machinery. https://www.microsoft.com/en-us/research/publication/quickr-lazily-approximating-complex-ad-hoc-queries-in-big-data-clusters/

[37] Donald E. Knuth. 1997. *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., USA.

[38] Richard E. Ladner and Michael J. Fischer. 1980. Parallel Prefix Computation. *J. ACM* 27, 4 (Oct. 1980), 831–838. https://doi.org/10.1145/322217.322232

[39] Shangqi Lai, Xingliang Yuan, Joseph K. Liu, Xun Yi, Qi Li, Dongxi Liu, and Surya Nepal. 2021. OblivSketch: Oblivious Network Measurement as a Cloud Service. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society. https://doi.org/10.14722/ndss.2021.24330

[40] Xiao Lan, Hongjian Jin, Hui Guo, and Xiao Wang. 2023. Efficient and Secure Quantile Aggregation of Private Data Streams. *IEEE Transactions on Information Forensics and Security* 18 (2023), 3058–3073. https://doi.org/10.1109/TIFS.2023.3272775

[41] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. 2023. SECRECY: Secure collaborative analytics in untrusted clouds. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1031–1056. https://www.usenix.org/conference/nsdi23/presentation/liagouris

[42] Qiyao Luo, Yilei Wang, Ke Yi, Sheng Wang, and Feifei Li. 2023. Secure Sampling for Approximate Multi-party Query Processing. *Proc. ACM Manag. Data* 1, 3, Article 219 (Nov. 2023), 27 pages. https://doi.org/10.1145/3617339

[43] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2006. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.* 31, 3 (Sept. 2006), 1095–1133. https://doi.org/10.1145/1166074.1166084

[44] J. Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming* 2, 2 (1982), 143–152. https://doi.org/10.1016/0167-6423(82)90012-0

[45] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 35–52. https://doi.org/10.1145/3243734.3243760

[46] Payman Mohassel, Peter Rindal, and Mike Rosulek. 2020. Fast Database Joins and PSI for Secret Shared Data. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1271–1287. https://doi.org/10.1145/3372297.3423358

[47] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. 2019. Streambox-TZ: secure stream analytics at the edge with trustzone. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 537–554.

[48] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA)

(SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 1461–1476. https://doi.org/10.1145/3183713.3196905

[49] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*.

[50] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M. Hellerstein. 2021. Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2129–2146. https://www.usenix.org/conference/usenixsecurity21/presentation/poddar

[51] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems* (Baltimore, MD, USA) *(SenSys '04)*. Association for Computing Machinery, New York, NY, USA, 239–249. https://doi.org/10.1145/1031495.1031524

[52] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2021. DP-Sync: Hiding Update Patterns in Secure Outsourced Databases with Differential Privacy. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1892–1905. https://doi.org/10.1145/3448016.3457306

[53] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2022. IncShrink: Architecting Efficient Outsourced Databases using Incremental MPC and Differential Privacy. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 818–832. https://doi.org/10.1145/3514221.3526151

[54] Pinghui Wang, Chengjin Yang, Dongdong Xie, Junzhou Zhao, Hui Li, Jing Tao, and Xiaohong Guan. 2023. An Effective and Differentially Private Protocol for Secure Distributed Cardinality Estimation. *Proc. ACM Manag. Data* 1, 1, Article 60 (May 2023), 24 pages. https://doi.org/10.1145/3588914

[55] Qichen Wang, Qiyao Luo, and Yilei Wang. 2024. Relational Algorithms for Top-k Query Evaluation. *Proc. ACM Manag. Data* 2, 3, Article 168 (May 2024), 27 pages. https://doi.org/10.1145/3654971

[56] Xiao Wang, Hubert Chan, and Elaine Shi. 2015. Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 850–861. https://doi.org/10.1145/2810103.2813634

[57] Yilei Wang and Ke Yi. 2021. Secure Yannakakis: Join-Aggregate Queries over Private Data. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1969–1981. https://doi.org/10.1145/3448016.3452808

[58] Yilei Wang and Ke Yi. 2022. Query Evaluation by Circuits. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Philadelphia, PA, USA) *(PODS '22)*. Association for Computing Machinery, New York, NY, USA, 67–78. https://doi.org/10.1145/3517804.3524142

[59] Yanping Zhang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Longshot: Indexing Growing Databases Using MPC and Differential Privacy. *Proc. VLDB Endow.* 16, 8 (April 2023), 2005–2018. https://doi.org/10.14778/3594512.3594529

[60] Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. 2022. Differentially private linear sketches: efficient implementations and applications. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 922, 14 pages.

[61] Peizhao Zhou, Xiaojie Guo, Pinzhi Chen, Tong Li, Siyi Lv, and Zheli Liu. 2024. Shortcut: Making MPC-based Collaborative Analytics Efficient on Dynamic Databases. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 854–868. https://doi.org/10.1145/3658644.3690314

Received October 2025; revised January 2026; accepted February 2026