KVSAgg: Secure Aggregation of Distributed Key-Value Sets

Yuhan Wu*, Siyuan Dong*, Yi Zhou*, Yikai Zhao*, Fangcheng Fu*,
Tong Yang*†, Chaoyue Niu‡, Fan Wu‡, Bin Cui*
**School of Computer Science, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing, China* †*Peng Cheng Laboratory, Shenzhen, China*
‡*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*

***Abstract*—In global data analysis, the central server needs the global statistic of the user data stored in local clients. In such cases, an Honest-but-Curious central server might put user privacy at risk in trying to collect individual statistics of each user. In response, the secure aggregation provides a solution for calculating global statistics without revealing users' privacy data. However, existing secure aggregation protocols only focus on the data in the form of vectors or common sets, which limits their application scope. We formalize a general problem—key-value set secure aggregation—that not only includes secure vector aggregation and private set union but also supports more applications. To address the proposed problem, we devise our solution (called the KVSAgg framework) that promises satisfactory performance in security, efficiency, and accuracy. Our key technique is a homomorphic transform algorithm (called HyperIBLT) that is not only capable of bidirectionally transforming data between key-value sets and vectors, but also able to transform sum operation of sets to addition of vectors. We implement KVSAgg on both CPU and GPU platforms and perform the evaluation on three use cases including federated learning, distributed data counting, and finding global hot items. Compared with our baselines, KVSAgg simultaneously achieves the best security, efficiency higher by orders of magnitude, and zero-error in nearly all cases. All codes are open-source anonymously.**

***Index Terms*—Secure Aggregation, Key-Value Sets, Federated Learning, Distributed Machine Learning, Multi-Party Computation, Semi-Honest, Honest-but-Curious, Sketches**

I. Introduction

In wide-area data management where private user data are generated and stored in local clients like mobile phones, the data curator (*i.e.*, a central server) calculates a global summary (*e.g.*, the sum vector, global counting, global hot items) of user data to do analysis and provide services. However, when the server is not fully trusted, the local private user data may be leaked to the server during the calculation of global summary. We focus on protecting user data under a common assumption that the server is Honest-but-Curious (HBC), *i.e.*, the server honestly follows the protocol but tries to figure out additional private data from each user. This problem of computing a multiparty global summary on an HBC server is called *secure aggregation*, which has attracted wide attention and has a great many of solutions [1]–[10].

Co-primary authors: Yuhan Wu, Siyuan Dong, and Yi Zhou. Corresponding author: Tong Yang (yangtongemail@gmail.com).

However, the application of existing secure aggregation solutions are limited by the form of data representation, according to which we classify them into two types. The first type is Secure Vector Aggregation (SVA) [1]–[6], where users' data are vectors of uniform dimension/length and the global summary is the sum of all vectors. The second type is Private Set Union (PSU) [7]–[10], where each user has a set and the global summary is the union of all sets. Supporting only vectors and ordinary sets are far from enough. For example, both types of solutions cannot efficiently summing up highly sparse vectors. Given two local sparse vectors $A = [5, 2, 0, 0, 0]$ and $B = [0, 4, 0, 0, 0]$, compared with uploading the whole vector using SVA, an ideal efficient way is to upload non-zero elements through Key-Value (KV) sets, *i.e.*, $\{\langle 0, 5\rangle, \langle 1, 2\rangle\}$ for A and $\{\langle 1, 4\rangle\}$ for B. The required sum set is $\{\langle 0, 5\rangle, \langle 1, 6\rangle\}$, which can then be converted to a sum vector $A + B = [5, 6, 0, 0, 0]$. Here PSU does not work because it cannot merge sets with values. Consequently, we aim at solving a more general secure aggregation problem that supports data in the form of KV sets.

We formalize the **Key-Value Set Secure Aggregation problem** (P_{Agg}^{KVS}): In a distributed system including one central server and N local clients, each local client u holds a set S_u with n_u KV pairs. The central server aims to acquire the aggregated sum set $S = \bigoplus_{\forall u} S_u$, where the sum set includes all local keys and $\bigoplus$ is the sum operation of KV sets that will sum up all values with the same key. Our security model assumes that the central server is HBC, which means it will execute the protocols correctly but also try to acquire as much information as possible. All local clients are honest and should be protected against the HBC adversary. Our P_{Agg}^{KVS} problem is a general one that includes existing SVA and PSU as special cases (§ II-A). Besides, we give three typical use cases of P_{Agg}^{KVS}, that cannot be effectively solved by SVA or PSU:

Use case 1: Federated learning. The Federated Learning (FL) [11]–[16] is a rising machine learning framework that trains models at local clients without exchanging local private data, and aggregates/sums up local gradient vectors in a central Parameter Server. However, secure aggregation is still required in FL because gradient vectors can leak user privacy and incur attacks [17]–[24]. Instead of uploading all gradients in one client, recent studies including MinMax [25] and GSpar

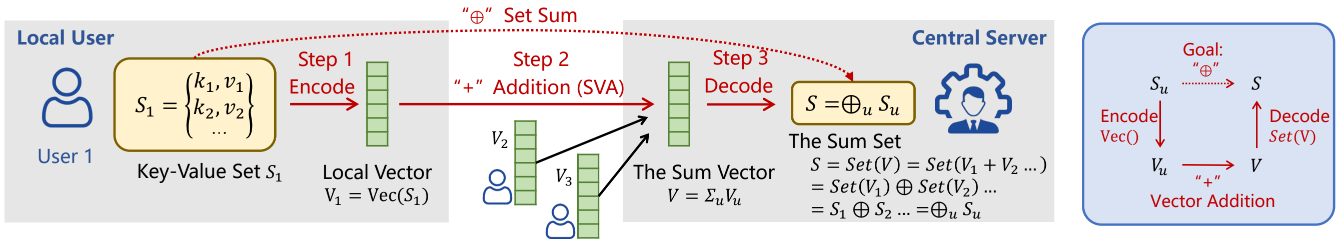

"⊕" Set Sum
Local User
User 1
$S_1 = \{k_1, v_1; k_2, v_2; \dots\}$
Key-Value Set S_1
Step 1 Encode
Local Vector $V_1 = Vec(S_1)$
Step 2 "+" Addition (SVA)
V_2
V_3
The Sum Vector $V = \Sigma_u V_u$
Step 3 Decode
Central Server
$S = \oplus_u S_u$
The Sum Set
$S = Set(V) = Set(V_1 + V_2 \dots) = Set(V_1) \oplus Set(V_2) \dots = S_1 \oplus S_2 \dots = \oplus_u S_u$
Goal: "⊕"
S_u
S
Encode Vec()
Decode Set(V)
V_u
V
"+" Vector Addition

Fig. 1. The workflow of the KVSAgg framework.

[26] shows that only uploading a small portion (*e.g.*, 1%) of the gradient vector is sufficient to achieve satisfactory performance[1]. In this case, it is more efficient to aggregate the data in the form of KV sets instead of highly sparse vectors. Another example with much more sparse vectors is the FL recommendation models [7], [27]. The full model with billions of goods has no less than billions of parameters, while each local user only accesses a certain number of goods that update a few parameters [7]. It is also a better choice to aggregate in the form of KV sets.

Use case 2: Distributed data counting. In distributed scenarios including global geo-distributed databases [28]–[30] and Content Delivery Networks (CDN) [31]–[33], the central server aggregates local states (*e.g.*, the frequency distribution of system events, and historical request frequency of a resource) to show a global view. The local states can be many KV pairs of the event/resource identifier and the frequency (or any additive value like the web-page visiting time) [34], [35]. Since the identifier may have an enormous range, the aggregation of vectors is not applicable. For example, when counting global access number of each 128-bit IPv6 address using vectors, the vector will be with a size of $2^{128} \approx 3.4 \times 10^{38}$ that cannot be stored and transmitted.

Use case 3: Finding global hot items. In edge computing [36]–[40] and CDN, the central operator finds hot resources, and caches/pushes them to the edge server for lower access latency. The access logs in clients are KV sets and private data. Compared with use case 2 that emphasizes values, here we focus on hot keys. Similarly, P_{Agg}^{KVS} is applicable.

From the above three use cases, we list three key requirements for the P_{Agg}^{KVS} solution as follows. **R1: Security.** The primary concern is security. The private data in local clients must be secure against the HBC central server. **R2: Efficiency.** The solution should be efficient in communication and local computation. The local clients can be mobile phone or Internet of Things sensors, where network traffic and compute power are limited. In contrast, the central server has cheaper electricity expenses and specialized hardware, *e.g.*, GPUs. **R3: Accuracy.** The solution should be as accurate as possible under tolerable overhead. In the ideal case that the sum set is error-free, the user can trust the answer without considering the impact of errors on subsequent applications, *e.g.*, data analysis and FL.

[1]MinMax and GSpar selects a few elements in the gradient vector, transmits the indexes of these elements with renewed values, and holds the others.

In this paper, we present the **K**ey-**V**alue Set **S**ecure **Agg**regation (KVSAgg) framework (including a key transform algorithm HyperIBLT) that can address the P_{Agg}^{KVS} problem and meet the above three requirements. *Security:* To the best of our knowledge, KVSAgg is the first secure solution for KV set data against the HBC adversary under the centralized aggregation model. *Efficiency:* KVSAgg uses small amount of bandwidth usage and has a low computation overhead that only three hash calculations are required to encode every KV pair. *Accuracy:* KVSAgg has a sharp threshold that the error of the aggregation sharply drops to zero when the size of HyperIBLT exceeds 1.23 times of the data.

The methodology of our framework KVSAgg is problem reduction. To be specific, we reduce the more general problem of KV sets (P_{Agg}^{KVS}) to more well-studied SVA. As shown in Fig. I, KVSAgg works in three steps: (Step1 Encode) At local, each user u transforms its KV set S_u to a vector V_u using encoding function $Vec(\cdot)$; (Step2 Aggregate) Aggregate local vectors by vector addition through SVA and the server gets the sum vector $V = \sum_u V_u$; (Step3 Decode) The server transforms the sum vector V back to the desired sum set S using the decoding function $Set(\cdot)$. Generally, KVSAgg is secure in every step because Step1 and Step3 are performed in individual devices without data exchange, and the security of Step2 is provided by SVA. Here, the key technique of the framework is the *transform algorithm*, which includes encoding and decoding functions.

It is critical for the transform algorithm to be homomorphic: it must transform the sum operation of KV sets "$\oplus$" to the addition of vectors "+". For example, for any two user sets S_1 and S_2, the algorithm must satisfy $Vec(S_1) + Vec(S_2) = Vec(S_1 \oplus S_2)$, which means that, in Step 2, the addition of two vectors is equivalent to merging two sets first and then converting the merged set into a vector. Only when the algorithm is homomorphic can KVSAgg get the correct results: $S = Set(V) = Set(Vec(S_1) + Vec(S_2)\dots) = Set(Vec(S_1 \oplus S_2 \dots)) = S_1 \oplus S_2 \dots = \bigoplus_{\forall u} S_u$. Besides being homomorphic, the transform algorithm should also be compact (*i.e.*, the vector size is small) and accurate (*i.e.*, the decoded KV set is accurate).

It is a challenge to find the desired transform algorithm that is homomorphic, compact and accurate. No existing algorithms we know satisfies the requirement, including Invertible Bloom Lookup Tables (IBLT) [41], sketch-based encode [42]–[44], and one-hot encode (§ III-A). Among them, IBLT has the best potential to be improved into the desired algorithm. IBLT is a hash table that can transform data bidirectionally with compact vector sizes while being error-free in the vast majority of cases (arbitrarily close to 100%), which is compact and accurate enough. However, IBLT is not homomorphic, which is the primarily required feature. IBLT focuses on the operations on a single device without considering the merge/addition of multiple IBLT vectors. IBLTs with different hash functions and hash seeds cannot be merged by vector addition, because the merged IBLT cannot be decoded. However, even if we set the hash functions and seeds to be the same, when one key appears in different local sets with different values, the decoding operation will fail and be considered as a failure.

We design HyperIBLT (inspired by IBLT) as the transform algorithm for the KVSAgg framework. To make it homomorphic, we propose the merge operation of multiple HyperIBLTs corresponding to the sum operation on KV sets. To ensure that the merged HyperIBLT can be converted back, we propose a new decoding operation that supports adding up different values with the same key. In decoding, a same key with multiple values is no longer a fault, and we try to calculate the key and the sum of values. In the decoding operation, we propose a technique (called Rehash) to verify whether one bucket contains the same key, regardless of the repetition of keys and the corresponding different values. After the verification, we can acquire the key and the sum value. We introduce detailed HyperIBLT in § III-B. For efficiency and accuracy, HyperIBLT benefits from IBLT, and, based on IBLT, HyperIBLT further reduces 30% vector-size/communication/storage plus half local hashing computation, with negligible negative impact on accuracy.

Key Contributions:

- We define the problem of P_{Agg}^{KVS} under the HBC model.
- We propose the framework KVSAgg and the transform algorithm HyperIBLT to address P_{Agg}^{KVS} with satisfactory security, efficiency and accuracy.
- We implement KVSAgg on both CPU and GPU platforms and evaluate it in three use cases. Compared with our baselines, KVSAgg simultaneously achieves the best security, orders-of-magnitude higher efficiency, and zero-error in nearly all cases. All codes are available at Github [45].

II. Preliminary

A. Problem Definition

We define the problem of Key-Value set Secure Aggregation (P_{Agg}^{KVS}) as follows. N local clients are given, and each client u owns a set of n_u key-value pairs with no duplicate key[2]:

$$S_u = \{\langle k_{u,j}, v_{u,j}\rangle | j = 1, 2, \ldots, n_u\}, u = 1, 2, \ldots, N.$$

Let $K(S_u)$ denote the key set, *i.e.*, $K(S_u) = \{k_{u,1}, k_{u,2}, \cdots, k_{u,n_u}\}$, and $v_u(key)$ denote the value corresponding to key in set S_u (zero when key does not exist). A central server aims at the sum set S, which is the sum of all local sets:

$$S = \bigoplus S_u = \left\{ \langle k, v\rangle \;\middle|\; k \in \bigcup_{u=1}^{N} K(S_u), v = \sum_{u\in[1,N]\wedge k\in K(S_u)} v_u(k) \right\},$$

where we sum up the value corresponding to the same key. Let $|S|$ denote the size of S, *i.e.*, the number of distinct keys. For simplicity, we assume that all parties (clients and server) know the upper bound of $|S|$, denoted as M ($\geqslant |S|$). We discuss how to acquire M in § III-B.

Threat model. We assume that the central server is HBC (semi-honest): due to regulatory or reputational pressure, the server will obey the protocol but attempt to collect private client data for profit. Clients want to disclose as little private data as possible (preferably 0) in cooperating with the server. We define the ideal functionality $\mathcal{F}_{AGG}$: Each client inputs its own set S_u with M and the server inputs M. The server outputs the sum set S whilst the clients output nothing. Following the ideal-real paradigm in multi-party computation (MPC), KVSAgg is required to be semantically secure:

Definition 1. (Security. Following Definition 4.1 of [46].) The protocol π securely realizes $\mathcal{F}_{AGG}$ if for every Probabilistic Polynomial-Time (PPT) adversary attacking the real interaction, there exists a PPT simulator $\mathcal{S}_Q$ attacking the ideal interaction, such that for any input $\mathcal{X}$ and any security parameter $n \in \mathbb{N}$, the ideal interaction ($\mathcal{S}_Q(1^n, M, S)$) and the real interaction ($view_Q^\pi(\mathcal{X}, n)$) are computationally indistinguishable as follows:

$$\{\mathcal{S}_Q(1^n, M, S)\}_{\mathcal{X},n} \overset{c}{\equiv} \{view_Q^\pi(\mathcal{X}, n)\}_{\mathcal{X},n},$$

where two probability ensembles[3] F_1 and F_2 are computationally indistinguishable (denoted by $F_1 \overset{c}{\equiv} F_2$) if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every $\mathcal{X}$ and every $n \in \mathbb{N}$,

$$\left|\Pr[D(F_1(\mathcal{X}, n)) = 1] - \Pr[D(F_2(\mathcal{X}, n)) = 1]\right| \leqslant \mu(n).$$

Besides, we assume that the user is honest and will not attack the server (*e.g.*, poisoning [47]) or be corroded by the server.

Our P_{Agg}^{KVS} is a general problem that includes both SVA and PSU as special cases (explained in the following section).

B. Related Work on Secure Aggregation

We introduce existing secure aggregation solutions that include SVA and PSU. Then we briefly introduce other works on privacy protection under different models/assumptions, and discuss why P_{Agg}^{KVS} is a general problem.

Secure Vector Aggregation (SVA) [1]–[6], [48] can securely compute the sum of local data vectors at a central server in the HBC setting. Specifically, each client u owns a local data vector V_u and the aggregation computes the sum vector $V = \sum_{u=1}^{n} V_u$. The sum vector and all local vectors are of the same size. One milestone in secure aggregation is the Secure Aggregation Protocol (SAP) [1] proposed by Bonawitz *et al.* SAP leverages the secure multiparty computation to compute

[2]The absence of duplicate key is only for the convenience of description. If duplicate keys do exist, HyperIBLT can still be applied directly.

[3]A probability ensemble $F_1 = \{F_1(\mathcal{X}, n)\}_{\mathcal{X};n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $\mathcal{X}$ and $n \in \mathbb{N}$.

the sum vector in a lossless manner. We outline the key steps of SAP: (1) broadcast private keys using t-out-of-n secret shares to prevent user dropouts. (2) generate, from each pair of clients, a private random vector and masks it on the local vector, and (3) sum up all masked local vectors and remove the effect of masks. In the end, the server can acquire the sum vector but cannot figure out each local vector.

Private Set Union (PSU) can securely compute the union of local ordinary sets without values at a central server in the HBC setting. Existing solutions can be divided into two types: polynomial based [8], [49], [50] and Bloom filter based [7], [51]–[53]. They cannot be directly extended to solve P_{Agg}^{KVS}.

Other Models. In the decentralized model without the central server, existing work [54] studies the problem of secure KV set aggregation, but their solution does not meet the design requirements for our centralized model. Another related topic is the Differential Privacy (DP) [55]–[66]. DP can aggregate KV sets securely by adding noise to the data at the cost of accuracy [67]–[71]. See Sec V-C for experiment comparisons. P_{Agg}^{KVS} can potentially be composed with DP well. When aggregating KV sets with local DP, the uploaded data is in the form of the KV set whose values are with the noise [72]. P_{Agg}^{KVS} can upload those KV sets efficiently.

Discussion—Why P_{Agg}^{KVS} is a general problem. Our P_{Agg}^{KVS} is general because both SVA and PSU are its special cases solvable by its general solution. Suppose we already have a solution for P_{Agg}^{KVS}. To solve SVA, for each vector, we can set each key to the index of the element in the vector, and the value is the key-th element in the vector. Thus the sum of key-value set can figure out the sum of vectors. To solve PSU, we set the key in the KV set to be the same as the ordinary set, and assign the value to zero (*i.e.*, no value). Thus the sum of KV sets can figure out the union of ordinary sets.

C. Transform Algorithms between KV sets and Vectors

The sketch and the Bloom filter. Sketches [43], [44], [73]–[84] are probabilistic algorithms capable of encoding the KV set into a mergeable vector and approximately decoding the value from the vector when the keys are given. The Bloom filter [85]–[88] is a special case of sketch. It can only approximate whether each key exists, not the corresponding value. Among all sketch variants, the Count Sketch (CSketch) [42] is well-known for its unbiased estimation and has been applied to sparse vector compression [89]. A CSketch supports three operations: encoding a KV pair, estimating the total encoded value of a given key, and merging two CSketches. A CSketch $\mathcal{C}$ consists of d sub-tables, $C_1, C_2, \dots, C_d$, and each sub-table has w counters. By concatenating d sub-tables, CSketch can be regarded as a vector with the size of $w \times d$.

`Encode(`$\langle key, value\rangle$`)`. To encode $\langle key, value\rangle$, CSketch picks one *selected counter* from each sub-table by independent hashing and updates all these selected counters. Specifically, the i-th selected counter is $\mathcal{C}_i[h_i(key)]$ and the counter will be added by $value \cdot H_i(key)$, where $h_i(\cdot)$ is a hash function mapping a key to one counter in $\mathcal{C}_i$ equally, and $H_i(\cdot)$ is a hash function mapping a key to $\{+1, -1\}$ equally. In other words, CSketch alters the d selected counters by $value$ according to the hash. Users can encode multiple key-value pairs with the same key and different values.

`Estimate(`key`)`. Given a key, CSketch can estimate the sum of all encoded values corresponding to the key, denoted as V_{sum}. In each sub-table $\mathcal{C}_i$, CSketch gets one estimated value by multiplying the selected counter by $H_i(key)$, *i.e.*, $v_i = \mathcal{C}_i[h_i(key)] \cdot H_i(key)$. Among the d estimated values from each sub-table, CSketch selects the median value as the final estimation, *i.e.*, $\widehat{V}_{sum} = Median\{v_1, v_2, \dots, v_d\}$.

`Merge(`$\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$`)`. If two CSketches share the same parameters and hash functions, we can merge them by adding up each corresponding counter and get a merged sketch with the same parameters and hash functions. $\mathcal{C} = \mathcal{C}^{(1)} + \mathcal{C}^{(2)}$.

Accuracy. Although each estimated value $\widehat{V}_{sum}$ is usually inexact, the error is controllable with high confidence. Specifically, CSketch guarantees each estimation has only a small probability δ that the absolute error is greater than $\epsilon\|V\|_2$, where ϵ and δ are two small constants: $\Pr\left[\left|\widehat{V}_{sum} - V_{sum}\right| > \epsilon\|V\|_2\right] \leqslant \delta$, when $w = 4\epsilon^{-2}$, $d = 8\ln(\delta^{-1})$ and $\|V\|_2 = \sqrt{\sum_{\forall key}\left(V_{sum}(key)\right)^2}$ [90]–[92].

Invertible Bloom Lookup Tables (IBLT) [41] is a compact data structure that can encode the KV set into a vector and decode all KV pairs exactly in nearly all cases. Although its name includes Bloom, it is quite different from the Bloom filter and the sketch in terms of function and accuracy. IBLT has d sub-tables, $\mathcal{B}_1, \dots, \mathcal{B}_d$, with w buckets per sub-table.

`Encode(`$\langle key, value\rangle$`)`. To insert a KV pair $\langle key, value\rangle$, we first select one bucket from each sub-table by hashing. The i-th selected bucket is $\mathcal{B}_i[h_i(key)]$, where $h_i(\cdot)$ is a hash function mapping a key to any bucket in the i-th sub-table with equal probability. Then we update all selected buckets. In each bucket, there are three fields: (1) $Freq$, how many pairs have been inserted into this bucket; (2) $KeySum$, the sum of the keys in this bucket; (3) $ValSum$, the sum of the values in this bucket. For every selected bucket, we add $Freq$ by 1, add $KeySum$ by key, and add $ValSum$ by $value$.

`Decode(`$\mathcal{B}$`)`. IBLT $\mathcal{B}$ can decode all inserted KV pairs exactly in nearly all cases. The decoding procedure is:

1) Find out one bucket (called *pure* bucket) that contains only one KV pair, *i.e.*, $\mathcal{B}_i[j].Freq = 1$, by scanning all buckets. From the pure bucket, we can decode one KV pair, *i.e.*, $key = \mathcal{B}_i[j].KeySum$ and $value = \mathcal{B}_i[j].ValSum$. Then we add $\langle key, value\rangle$ to the answer set and delete it from the IBLT $\mathcal{B}$ as follows.
2) To delete $\langle key, value\rangle$, we access every sub-table i, locate the bucket $\mathcal{B}_i[h_i(key)]$, and subtract its $\{Freq, KeySum, ValSum\}$ by $\{1, key, value\}$, respectively.
3) Repeat the above two steps until all buckets are zero.

If each key is inserted only once, IBLT works well. However, different users may have the same key. When every user converts the local set into an IBLT and the server merges all the IBLTs securely, one key in the merged IBLT may correspond to different values from multiple users. Although the optimization of IBLT has considered that a single key may be inserted multiple times[4], its decoding procedure fails when one key corresponds to multiple values in different key-value pairs, which is inevitable in calculating the sum set on the central server. The hash tables [93]–[95] other than IBLT cannot merge the entire data structure by vector addition because a key may be stored in different locations, so they do not meet our desire of being homomorphic.

TABLE I
NOTATIONS.

Symbol	Meaning
N	the number of local clients
S	the aggregated sum set
M	the upper bound of $\|S\|$
d	the number of sub-tables
w	the width of each sub-table
R	the ratio of total buckets to inserted keys, $R = \frac{w \times d}{M}$

TABLE II
THE COMPARISON OF ALGORITHMS.

Algorithm	Security	Space Efficiency	Accuracy
Baseline 1 (One-hot)	Yes	Low	High
Baseline 2 (CS)	Yes	High	Low
Baseline 3 (CSK)	No	High	Median
HyperIBLT	**Yes**	**High**	**High**

III. KEY-VALUE SET SECURE AGGREGATION

In this section, we propose our framework KVSAgg for P_{Agg}^{KVS} and introduce the baseline transform algorithms using One-hot vector and CSketch. These algorithms are either non-secure, space-expensive, or inaccurate. Then we introduce our transform algorithm HyperIBLT and discuss its performance in security, efficiency, and accuracy in detail.

The KVSAgg Framework. Our KVSAgg framework is to use a bidirectional homomorphic data form transform algorithm between KV sets and vectors, and combine it with existing SVA solutions. The workflow is as follows: **(Step 1 Encode)** locally convert set data to vectors, **(Step 2 Aggregate)** aggregate the vectors by SVA, and **(Step 3 Decode)** convert the aggregated vector back to the set. The framework is secure in every step: In Step 1, all clients convert data locally; in Step 2, existing solutions can aggregate vectors securely; in Step 3, data from different clients are no longer distinguishable after the aggregation. Our baseline 3 is not secure because it transmits additional information outside the framework. The efficiency and accuracy of the framework depend on the quality of the transform algorithm.

A. The Baseline Algorithms

Baseline 1—One-hot vector. The simplest baseline solution of P_{Agg}^{KVS} is to combine One-hot vector with SVA. In step 1, for each local client u and its KV set S_u with x-bit keys and y-bit values, we construct a local One-hot sub-table of 2^x y-bit elements, where the key-th element records $value$. In step 2, the One-hot vector is securely summed up by SVA. In step 3, we scan the sum vector, find all non-zero elements, and convert each element back to the KV pair. It shall be noted that the One-hot vector cannot efficiently handle long keys due to excessive vector size.

[4]The IBLT [41] adds an additional field called hashKeySum to each bucket.

Algorithm 1: Baseline Algorithm 2— CSketch

```
1  for Each local client $u = 1, 2, \ldots, N$ do
2      Insert all KV pairs in local set $S_u$ to $\mathcal{C}^{(u)}$.
3      Initialize $Encrypt()$ function of SAP.
4      $E^{(u)} \leftarrow Encrypt(\mathcal{C}^{(u)})$
5  end
6  $\mathcal{C} \leftarrow Decrypt(E^{(1)}, E^{(2)}, \ldots, E^{(N)})$
7  for Each key in the universal set $\mathcal{U}$ do
8      $\widehat{V}_{sum} \leftarrow Median_i\{v_i\}, v_i = \mathcal{C}_i[h_i(key)] \cdot H_i(key)$
9      Append $\langle key, \widehat{V}_{sum} \rangle$ to the sum set $S$.
10 end
```

Baseline 2—CSketch (CS) (Pseudocode in Algorithm 1). In step 1, we construct a local CSketch $C^{(u)}$ with size d and w (according to § II-C), and insert all KV pairs in S_u into the sketch. In step 2, after concatenating the sub-tables of CSketch, the CSketch can be regarded as a vector V_u with the size of $w \times d$, and then we use SVA to securely merge/sum up all local sketches. In step 3, the central server restores the sum set, which includes all local keys and sums up the values corresponding to the same key. However, CSketch cannot restore keys because keys are not recorded in the vector. Here we assume that the server knows a universal set $\mathcal{U}$ that all keys must belong to. The server can enumerate all possible keys, estimate the corresponding values by CSketch, and construct the sum set. The problem is that this method will generate many outlier keys (*i.e.*, keys not in the sum set) and the corresponding values, especially when the universal set $\mathcal{U}$ is much larger than the sum set. Therefore, the second baseline solution CS suffers from low accuracy and high computation overhead on the server.

Baseline 3—CSketch with Keys (CSK). We make some modifications to step 3 of Baseline 2. Let each client transmit the keys contained in the local set to the central server, so that the server does not need to enumerate the universal set. Thereafter, the server can estimate the values using given keys, and construct the sum set. Owing to this modification, baseline 3 eliminates outliers at the cost of exposing the key in a non-secure way and consuming more traffic for key transmission.

Summary. As shown in TABLE II, One-hot vector requires an excessive vector size (low space/communication efficiency). CS can hardly convert a vector back to a set in Step 3 because it does not record/insert keys (low accuracy). CSK is not secure because it transmits additional information outside the framework without protection. Our HyperIBLT aims at addressing all these defects and we introduce it as follows.

B. The HyperIBLT Algorithm

We propose HyperIBLT that overcomes the defects of the baseline algorithms and achieves high security, efficiency and accuracy. HyperIBLT follows our three-step framework (Pseudocode in Algorithm 3 and 2).

Step 1 Encode: At each local client, given a set, we insert each KV pair in the set into a local HyperIBLT structure one by one. The data structure and the encoding operation of HyperIBLT are identical to those of IBLT introduced in § II-C.

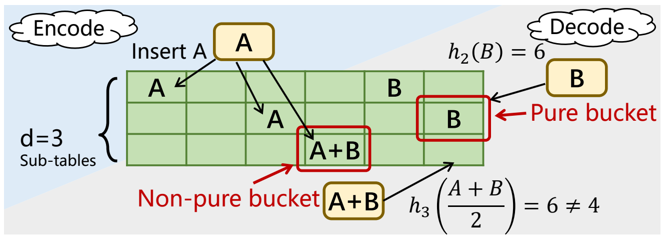


Fig. 2. Illustration of HyperIBLT. This is a HyperIBLT with $d=3$ sub-tables and each sub-table has $w=6$ buckets. When key A is inserted, it is hashed to the 1-st, 3-rd and 4-th buckets in different sub-tables. To verify whether the 6-th bucket in the second sub-table is pure, we hash the key B in the bucket again to check whether $h_2(B)$ equals to 6. To verify whether the 4-th bucket in the third sub-table is pure, we hash the key $(A{+}B)/2$ in the bucket again to check whether $h_2((A{+}B)/2)$ equals 4.

Algorithm 2: The KVSAgg Framework/Protocol.

```
1  The server sends the parameters $\{w, d, seed\}$ to all users.
2  **Parallel-for** *Each local client* $u = 1, 2, \ldots, N$ **do**
3                                                    ▷ Step 1
4      Encode the local KV set:
       $\mathcal{B}^{(u)} \leftarrow$Encode$(S_u, \{w, d, seed\})$
5                                                    ▷ Step 2
6      Initialize $Encrypt()$ function of SAP.
7      $E^{(u)} \leftarrow Encrypt(\mathcal{B}^{(u)})$
8  **end**
9  $\mathcal{B} \leftarrow Decrypt(E^{(1)}, E^{(2)}, \ldots, E^{(N)})$
10                                                   ▷ Step 3
11 Decode the sum HyperIBLT:
12 $S \leftarrow$Decode$(\mathcal{B})$
```

Algorithm 3: HyperIBLT: Encode and Decode.

```
1  **Function** Encode$(S_u, \{w, d, seed\})$:
2      Initialize HyperIBLT $\mathcal{B}^{(u)}$ with parameters $\{w, d, seed\}$.
3      **for** *Each* $\langle key, value\rangle$ *in* $S_u$ **do**
4          **for** *Layer* $i = 1, 2, \ldots, d$ **do**
5              $j \leftarrow h_i(key)$
6              $\mathcal{B}_i^{(u)}[j].Freq \leftarrow \mathcal{B}_i^{(u)}[j].Freq + 1$
7              $\mathcal{B}_i^{(u)}[j].KeySum \leftarrow \mathcal{B}_i^{(u)}[j].KeySum + key$
8              $\mathcal{B}_i^{(u)}[j].ValSum \leftarrow \mathcal{B}_i^{(u)}[j].ValSum + value$
9          **end**
10     **end**
11     **return** $\mathcal{B}^{(u)}$
12 **Function** Decode$(\mathcal{B})$:
13     Initialize the sum set $S \leftarrow \{\}$.
14     **for** *Each* $\mathcal{B}_i[j]$ *satisfies* $h_i\left(\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}\right) = j$ **do**
15         $\langle key, value\rangle \leftarrow \left\langle \frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}, \mathcal{B}_i[j].ValSum \right\rangle$
16         $t \leftarrow \mathcal{B}_i[j].Freq$
17         $S \leftarrow S + \{\langle key, value\rangle\}$
18         **for** *Layer* $i' = 1, 2, \ldots, d$ **do**
19             $j' \leftarrow h_{i'}(key)$
20             $\mathcal{B}_{i'}[j'].Freq \leftarrow \mathcal{B}_{i'}[j'].Freq - t$
21             $\mathcal{B}_{i'}[j'].KeySum \leftarrow \mathcal{B}_{i'}[j'].KeySum - key \times t$
22             $\mathcal{B}_{i'}[j'].ValSum \leftarrow \mathcal{B}_{i'}[j'].ValSum - value$
23         **end**
24     **end**
25     **return** $S$
```

Step 2 Aggregate: We arrange local HyperIBLTs into vectors and sum them up at the central server using SVA protocol (*e.g.*, SAP [1]). Specifically, we get the vector by putting the k-th field of the j-th bucket in the i-th sub-table in the position $p = (i\text{-}1)w{+}(j\text{-}1){+}(k\text{-}1)wd \quad (i{\in}[1,d], j{\in}[1,w], k{\in}[1,3])$. The vector size is $3wd$. Then, we sum up local vectors using SVA, which can be easily applied to different field sizes. The `merging` operation of two HyperIBLTs adds the counters at the same position when all hash functions and seeds are the same. The procedure of securely summing up all local vectors naturally accomplishes the merging of multiple HyperIBLTs.

Step 3 Decode: We obtain the sum of all local vectors at the central server using SVA and convert the vector back to a HyperIBLT by the reverse arrangement procedure of Step 2. Then we perform the decoding operation to convert the HyperIBLT back to the sum set. The following part is the key difference between our HyperIBLT and IBLT. The decoding procedure of HyperIBLT should consider the case of multiple inserts of a key with different values, because those KV pairs are from different users. At a high level, the decoding procedure of HyperIBLT is to find out one *pure* bucket that all the KV pairs it contains have the same key, delete the key from each sub-table, and repeat until all buckets are cleared. There are two key operations: (1) *pure bucket verification* that verifies whether a bucket only records a single kind of keys using *Rehash*; (2) *key-value pair extraction* that extracts the KV pair from a pure bucket and deletes it in all sub-tables.

Pure Bucket Verification—Rehash: Suppose a bucket $\mathcal{B}_i[j]$ is a pure bucket that only records t KV pairs with the same key. We know that $\mathcal{B}_i[j].Freq = t, \mathcal{B}_i[j].KeySum = t \times key$, and $\mathcal{B}_i[j].ValSum$ records the total value. If the bucket is pure, we can get the key by $key = \frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}$. The hash index of the key must point to the current bucket, *i.e.*, $j = h_i(key) = h_i\left(\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}\right)$ because the key has been inserted here before. If the bucket is not pure, on the other hand, $\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}$ is either indivisible or near-random. In the near-random case, it is highly unlikely that hash index $h_i\left(\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}\right)$ equals to j. Therefore, we report PURE if $\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}$ is divisible and $h_i\left(\frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}\right) = j$. Otherwise, we report NOT-PURE. We give an example in Fig. 2. With a small probability (*i.e.*, $\frac{1}{w}$), we report PURE for a non-pure bucket. We show that such false rates have little impact on decoding (§ III-D).

Key-Value Pair Extraction: For every bucket $B_i[j]$ verified as pure, we calculate the pair $\langle key, value\rangle$ by $key = \frac{\mathcal{B}_i[j].KeySum}{\mathcal{B}_i[j].Freq}$ and $value = \mathcal{B}_i[j].ValSum$. Then we delete it as the reverse of the insert operation: for every sub-table i', locate the bucket $\mathcal{B}_{i'}[h_{i'}(key)]$, and subtract its $\{Freq, KeySum, ValSum\}$ by $\{t, key \times t, value\}$ respectively, where $t = \mathcal{B}_i[j].Freq$.

To search for pure buckets efficiently, we use a queue to store the pure bucket candidates. To begin with, we scan all buckets and insert all the pure buckets among them to the queue. Then we perform KV Pair Extraction for each bucket in the queue until the queue is empty. In each extraction, we modify/subtract d buckets. For each modified bucket, we check whether it is pure, and add it to the queue if it is. In this way, we search for each pure bucket without exhaustive scanning, and a bucket will not be verified repeatedly without being modified.

The decoding operation succeeds when all `Freq` fields are zero, *i.e.*, all buckets are empty, and fails when there is no pure buckets yet some buckets are not empty. In case of failure, some KV pairs (called failed keys) cannot be decoded from HyperIBLT. Fortunately, HyperIBLT with recommended parameters fails at a very low rate, and the number of failed keys is a small constant (§ III-D) in rare cases of failure. When decoding fails, we can still estimate the value of failed keys: for a failed key, find its d selected buckets and report the minimal ValSum field as the estimated value. In case of failure, we can also rerun HyperIBLT with a new random hash seed until it succeeds.

Parameter Configuration. With an upper bound of the set size M, by default, we set $d = 3$ and $w = \frac{R\times M}{d}$, where $R = 1.25$ is the ratio of total buckets to M. We will explain it in § III-D. In the cases with M unknown, a simple way is to use $\sum_u n_u$ as M, which can be summed up by SVA of 1-element vector. When the local sets are similar to each other and therefore $|S|$ is much smaller than $\sum_u n_u$, for efficiency, we need a better upper bound of $|S|$. We can leverage the recent efforts of private set union cardinality problem [96] to calculate an upper bound of $|S|$ with high confidence, *e.g.*, a PDCE protocol [97].

Novelty. Our HyperIBLT is inspired by IBLT but with the following two key contributions. **First,** We devise the merging operation of multiple HyperIBLTs, and devise the decoding algorithm that can decode the sum of multiple different values corresponding to the same key. IBLT focuses on a single data structure and regards the case that one key has different values as a fault. IBLT uses an optional additional field called hashvalueSum to confirm that all values are the same, but HyperIBLT does not. **Second,** HyperIBLT is more efficient in transmission and local computation using rehash. In every bucket, IBLT needs another extra field called hashkeySum (the sum of the hash values of keys, not the previous hashvalueSum). Compared with HyperIBLT, the field can take 30% more traffic and storage space in a typical setting[5]. The field mandates an additional hash function in every bucket, doubling the local hash computation in encoding and increasing the cost of decoding. Our Rehash avoids the extra cost of computation, storage and communication. Rehash has few shortcomings except for a negligible increase in the probability of decoding failure (§ III-D). In summary, HyperIBLT differs from IBLT in (1) merging operation, (2) summation of values, and (3) rehash verification. Similarly, HyperIBLT and IBLT share the same decoding idea of finding pure buckets and deleting them, which guarantees desirable accuracy.

[5]By 1-Byte Freq, 4-Byte KeySum, 4-Byte hashkeySum, and 8-Byte ValSum, the addition cost is $\frac{hashkeySum}{Freq+KeySum+ValSum} = \frac{4}{1+4+8} \approx 30.8\%$.

Next we introduce how we achieve the three key requirements of security, accuracy and efficiency (proposed in § I).

C. Security

Our KVSAgg is secure in all three steps against the HBC adversary. In the first step of locally converting set data to vectors, all data is processed locally without exchanging. In the second step of aggregating vectors, existing work (*e.g.*, SAP) ensures secure aggregation. To the server, the client to whom we add a KV pair is indistinguishable.

More strictly, we follow the standard simulation proof framework [46] to prove that our Algorithm 2 securely computes the P_{Agg}^{KVS} problem. Lemma 1 is introduced by existing work [1], and Theorem 1 proves that our algorithm is semantically secure.

Lemma 1. *Let τ be the secure aggregation algorithm (Encrypt, Decrypt) with security parameter n used in Algorithm 2. The inputs of N clients $C_1, \cdots, C_N$ are $u_1, \cdots, u_N$, and the outputs are all empty strings λ; The input of server Q is an empty string λ, and the output is $\sum_{i=1}^{N} u_i$. Let $view_Q^\tau(u_1, \cdots, u_N, n)$ be the view of the server Q during an execution of τ. Then there exists an simulator $\mathcal{S}_Q^\tau$ such that for any input $\mathcal{X} = \{u_i\}$, there is*

$$\left\{\mathcal{S}_Q^\tau\left(1^n, \sum_{i=1}^{N} u_i\right)\right\}_{\mathcal{X},n} \overset{c}{\equiv} \left\{view_Q^\tau(\mathcal{X}, n)\right\}_{\mathcal{X},n}.$$

Theorem 1. ***(Security)*** *Let π be the Algorithm 2 with security parameter n. The inputs of client C_u are S_u and M, and their outputs are all empty string λ; The input of server Q is M, and the output is $S = \bigoplus_{i=1}^{N} S_i$. Let $view_Q^\pi(S_1, \cdots, S_N, M, n)$ be the view of the server Q during an execution of τ. Then there exists an simulator $\mathcal{S}_Q$ such that for any input $\mathcal{X} = \{S_1, \cdots, S_N, M\}$, there is*

$$\left\{\mathcal{S}_Q\left(1^n, M, S\right)\right\}_{\mathcal{X},n} \overset{c}{\equiv} \left\{view_Q^\pi(\mathcal{X}, n)\right\}_{\mathcal{X},n}.$$

Proof. For the server Q, the real view can be written as $view_Q^\pi(\mathcal{X}, n) = \left(w, d, r; E^{(1)}, \cdots, E^{(N)}\right)$.

We construct simulator $\mathcal{S}_Q$ as follows:

1) $\mathcal{S}_Q$ generates parameters w and d according to the algorithm π: $d = 3, w = \frac{R\cdot M}{d}$, where $R = 1.25$.
2) $\mathcal{S}_Q$ chooses an uniform distributed random tape r', and uses this tape r' to compute a $seed'$.
3) $\mathcal{S}_Q$ encodes set $\bigoplus_{i=1}^{N} S_i$ as $\mathcal{B} = \sum_{i=1}^{N} B^{(i)}$ using Algorithm 3 with $\{w, d, seed'\}$.
4) According to Lemma 1, by setting $u_i = B^{(i)}$, $\mathcal{S}_Q$ can simulate the view of the secure aggregation algorithm τ on the server Q through simulator $\mathcal{S}_Q^\tau$, *i.e.*,

$$\left\{\mathcal{S}_Q^\tau\left(1^n, \sum_{i=1}^{N} B^{(i)}\right)\right\}_{\mathcal{X},n} \overset{c}{\equiv} \left\{\left(E^{(1)}, \cdots, E^{(N)}\right)\right\}_{\mathcal{X},n}$$

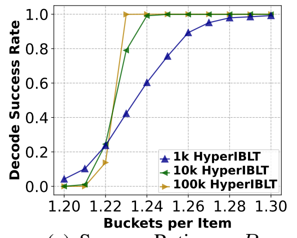
The output of the simulator $\mathcal{S}_Q$ can be written as

$$\mathcal{S}_Q\left(1^n, M, S\right) = \left(w, d, r'; \mathcal{S}_Q^\tau\left(1^n, \mathcal{B}\right)\right).$$

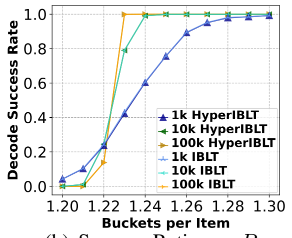
Further, we have

$$\left\{\left(w, d, r'; \mathcal{S}_Q^\tau\left(1^n, \mathcal{B}\right)\right)\right\} \overset{c}{\equiv} \left\{\left(w, d, r; E^{(1)}, \cdots, E^{(N)}\right)\right\}_{\mathcal{X},n}.$$

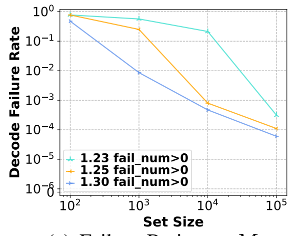
□



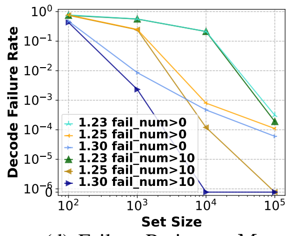
(a) Success Ratio *v.s.* R



(b) Success Ratio *v.s.* R



(c) Failure Ratio *v.s.* M



(d) Failure Ratio *v.s.* M

Fig. 3. Decode Success Ratio of HyperIBLT when $d = 3$.

D. Accuracy and Reliability.

HyperIBLT is error-free with a probability greater than 99.99%, and even in rare cases when failed keys cannot be decoded, the number of failed keys is unlikely to exceed 10.

Accuracy Experiment. We can test the failure probability by replacing the hash seed and reconducting the experiment. This probability does not depend on the specific data set, but only on the parameters M, R, and d. As shown in Fig. 3(a), the success rate of fully decoding sharply approaches 100% as we set R (*i.e.*, the ratio of buckets and keys) to be greater than 1.23. We set the number of keys $|S| = M$ to $10^3, 10^4$, and 10^5. In Fig. 3(b), compared with IBLT, HyperIBLT has nearly no accuracy loss, while saving both 30% size and half the hash computation. In Fig. 3(c), we repeat the decoding 10^5 times with different hash seeds to calculate the portion of the rare failures. Fig. 3(d) shows that the number of failed keys will hardly exceed 10 even when existing KV pairs cannot be decoded.

Accuracy Proofs. We prove that HyperIBLT is error-free with nearly 100% probability/confidence when the size of HyperIBLT is not too small. The decode fails by definition if any KV pair cannot be decoded. Otherwise, HyperIBLT is exact. We have the following theorem:

Theorem 2. ***(Accuracy)*** *When $w \times d > R_d \times M + \epsilon$ and $M \geqslant \Omega(d^{4d} log^d(M))$, the decoding of HyperIBLT fails with probability $O(\frac{1}{M^{d-2}})$, where ϵ and R_d are small constants.*

$$R_d = \left(sup \left\{ \alpha \middle| \alpha \in (0,1), \forall x \in (0,1), 1 - e^{-d\alpha x^{d-1}} \right\} \right)^{-1}$$

For example, $R_3 = 1.222, R_4 = 1.295, R_5 = 1.425, R_6 = 1.570.$

The detailed proofs can be found in our appendix [45].

Parameter Configuration Discussion. Based on the above experiments and theorems, we know that a high success rate can be obtained by setting w to a value slightly greater than $\frac{R_d \times M}{d}$. When $d = 3$, the failure probability is $O(\frac{1}{M})$, which is often negligible. If we need an extremely small failure probability, we can set d to 4, 5, or 6. The smaller the d is, the smaller the R_d will be, so the overall cost (*i.e.*, computation, storage, communication) will be smaller (§ III-E). Therefore, by default, we set $d = 3$, $R = 1.25$, and $w = \frac{R \times M}{d}$.

E. Efficiency.

In this part, we analyze the complexity. Since HyperIBLT can be combined with various SVA schemes, the overall complexity of KVSAgg may vary. We first show the complexity of the HyperIBLT part with no SVA protocol, and then show the overall complexity of KVSAgg using SAP [1] as the SVA protocol. We also show the complexity of the ideal lower bound of the P_{Agg}^{KVS} problem.

Complexity of HyperIBLT. As the client may be mobile devices with limited resources, HyperIBLT should be efficient in transmission, local storage and local computation. Since $w \times d = O(M)$, the complexity of storing and transmitting one HyperIBLT are both $O(M)$. The computation of encoding is $O(dM) = O(M)$, where d is a small constant. The computation of decoding is $O(d^2M) = O(M)$, because we extract at most M KV pairs. Although we describe our hash functions as fully random, many fast and simple hash functions (*e.g.*, MurmurHash3 [98] and BobHash [99]) also perform very well.

Complexity of KVSAgg and Others. The complexity of KVSAgg consists of two parts: the transform algorithm (HyperIBLT) and the SVA (*e.g.*, SAP [1]). In TABLE III, we compare the complexities of (1) the lower bound of P_{Agg}^{KVS} that any solution requires, (2) the individual overhead of HyperIBLT, (3) the KVSAgg including both HyperIBLT and SAP, (4) One-hot with SAP, (5) baseline CS, and (6) baseline CSK. Apart from showing the negligible failure probability, HyperIBLT (as a transform algorithm) achieves the lower bound/optimal complexity. To overcome the overall complexity bottleneck caused by SVA, we can conveniently replace the SAP in KVSAgg with new SVA solutions.

F. Discussions.

Client collusion and dropouts. At present, this work does not study the collusion and dropouts of some clients, which has been well studied by existing SVA schemes. We leave them to the future work.

IV. Applications

We introduce how KVSAgg works in three use cases:

Federated Learning. There are two scenarios of FL that need to aggregate the key-value sets. The first scenario is gradient compression using sampling (including MinMax [25], GSpar [26] and FetchSGD [89]) that can be applied to the upload of gradients in general FL models (*e.g.*, FedAvg [12]). The sampled data is in the form of many local key-value sets that need to be aggregated on the central server. The second scenario is the FL models with an extremely sparse data matrix, *e.g.*, the FL Submodel [7] for big recommendation systems with billions of commodities and model parameters. Only a few commodities and model parameters are related to local data. In our experiment, we only apply KVSAgg to the

TABLE III
COMPLEXITY ANALYSIS. $U \gg M$, WHERE U DENOTES THE SIZE OF THE UNIVERSAL SET OF KEYS. $C = O(\epsilon^{-2}\ln(\delta^{-1}))$ IS THE SIZE OF CSKETCH.

Complexity	Single Client			Server			Failure Prob.
	Communication	Storage	Computation	Communication	Storage	Computation	
Ideal Lower Bound	$\Omega(M)$	$\Omega(M)$	$\Omega(M)$	$\Omega(NM)$	$\Omega(M)$	$\Omega(NM)$	0
HyperIBLT	$O(M)$	$O(M)$	$O(M)$	$O(NM)$	$O(M)$	$O(NM)$	$O(\frac{1}{M^{d-2}})$
HyperIBLT+SAP	$O(M+N)$	$O(M+N)$	$O(NM+N^2)$	$O(NM+N^2)$	$O(M+N^2)$	$O(N^2M)$	$O(\frac{1}{M^{d-2}})$
Baseline 1 (One-hot)	$O(U+N)$	$O(U+N)$	$O(NU+N^2)$	$O(NU+N^2)$	$O(U+N^2)$	$O(N^2U)$	0
Baseline 2 (CS)	$O(C)$	$O(C)$	$O(M)$	$O(NC)$	$O(C)$	$O(U+NC)$	With errors
Baseline 3 (CSK) (Not Secure)	$O(C+M)$	$O(C)$	$O(M)$	$O(NC+NM)$	$O(C)$	$O(NC+NM)$	With errors

first scenario, leaving the second one to future work. Since the secure aggregation becomes harder and inefficient in practice as N increases in size, we divide N clients into groups where each group has at least G clients, and aggregate each group using the intermediate sum proposed by Bonawitz *et al.* [2]. The server has one central aggregation node and multiple sub-aggregation nodes. For each group, all clients in the group use KVSAgg to calculate the intermediate sum set of these clients on one sub-aggregation node. Then all sub-aggregation nodes upload intermediate sum sets to the central aggregation node without secure aggregation.

Distributed Data Counting. Given a large number of local key-value sets, distributed data counting aims at estimating the total value of a given key. For example, the mobile app developer may track the visiting time of each interface or the user's click pattern to guide app development [72]. When applying our KVSAgg, the key of the local set can be URL, IP address, product name, or other identifiers, and the value can be the frequency of the key, the access time, or other attributes. The central server can apply KVSAgg to aggregating full local sets and getting an exact sum set securely. Also, when communication is limited and the exact sum is not required, KVSAgg can be combined with many sampling methods. These methods allow uploading part of the local set for an approximate final result [31], [100].

Finding Global Hot Items. The central server searches for hot/popular keys (*e.g.*, IP addresses, file names and other resource identifiers) with large total values. Following the conversion of local data to the form of key-value sets, KVSAgg can aggregate them securely, and then the server knows which keys are hot after sorting them by value. Similar to the distributed data counting, KVSAgg can be combined with many sampling methods to save cost.

V. EXPERIMENTS

We implement KVSAgg, three baseline solutions (One-hot/CS/CSK), and privKV on both CPU and GPU platforms, and evaluate them in the three use cases. The source codes are available at Github [45].

Following key results are observed: (1) In the FL case, KVSAgg securely aggregates the data with no error in practice, while the baselines require $10\times$ vector size/communication/storage to achieve similar accuracy. (2) In distributed data counting and finding global hot items, even if the baselines use orders-of-magnitude larger vector sizes than HyperIBLT, they cannot achieve zero error aggregation.

A. Experimental Preliminaries

Metrics:

- *Mean Absolute Error (MAE):* For two given key-value sets S_1, S_2, we define MAE as $\sum_{key}(|v_1(key) - v_2(key)|)/|S_1 \cup S_2|$. Specially we set $v_i(key) = 0$ if $key \notin Key(S_i)$.
- *Root Mean Squared Error (RMSE):* $\sqrt{\frac{\sum_{key}(v_1(key)-v_2(key))^2}{|S_1 \cup S_2|}}$.
- *Precision:* The number of true positive results divided by the number of all positive results (including those not reported.)
- *Test Accuracy (Acc.):* The number of correct classification in the test set divided by the size of the test set.

Datasets:

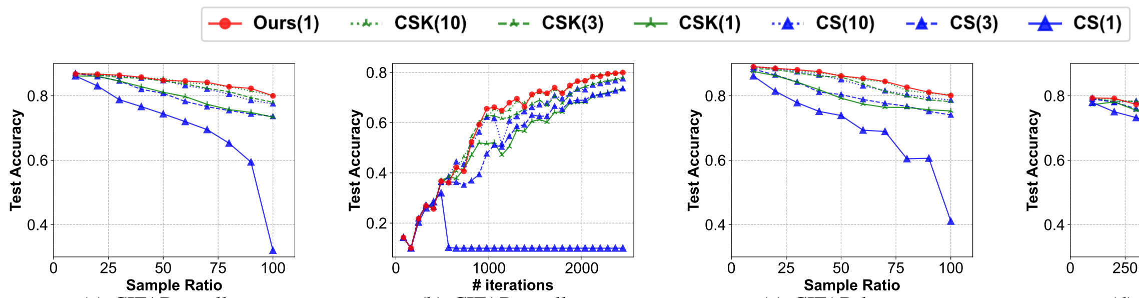
- FEMNIST: Federated EMNIST [101] is a native federated image dataset built by partitioning Extended MNIST based on the writer of images. It has 80,5263 28×28 samples in 62 different classes (10 digits, 52 lower and upper case English letters) and 3550 clients. We use Resnet-101 with 40M parameters and layer normalization for FEMNIST.
- CIFAR-10: CIFAR-10 is an image dataset consisting 60000 32×32 RGB images divided into a 50,000 training sets and 10,000 test sets. Each image belongs to one of the 10 classes.
- CAIDA: We use an anonymized network trace dataset collected by CAIDA [102] in 2018. Each entry in the dataset is a 5-tuple (source IP address, source port, destination IP address, destination port, protocol). A 5-tuple uniquely identifies a UDP/TCP session. The slice used contains network traffic in 1 hour, which includes 1.47G packets and 58.4M distinct entries.

B. Experiments on Federated Learning

We compare KVSAgg with three baseline solutions in FL.

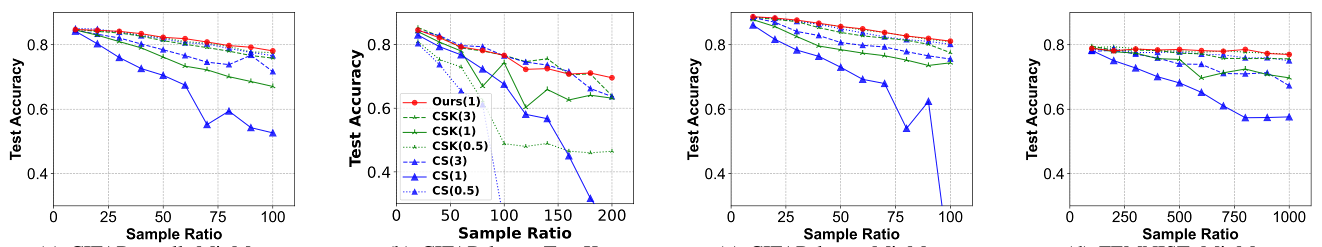
Data Allocation: To conduct federated learning experiments on CIFAR-10, we partition the training set into clients in the following way to make it *non-iid*: Choose a main label in 10 classes for each client, then make sure that in the partitioned CIFAR-10, every client consist of 90% images with the main label. We use Resnet-9 with 6.5M parameters for CIFAR-10. Since the number of clients in CIFAR-10 can be customized, we conduct experiments using FEMNIST [101] dataset and two different federated CIFAR-10 partitions.

CIFAR-small. 200 clients. In each iteration, we randomly choose 12 clients and divide them into groups of 4, then randomly select a total batch of size 600 from the participating clients to participate. The model is trained with 24,000 iterations.



(a) CIFAR-small. (b) CIFAR-small. (c) CIFAR-large. (d) FEMNIST.

Fig. 4. Federated Learning experiments on HyperIBLT+GSpar.



(a) CIFAR-small, MinMax. (b) CIFAR-large, Top-K. (c) CIFAR-large, MinMax. (d) FEMNIST, MinMax.

Fig. 5. Federated Learning experiments on HyperIBLT+{MinMax, Top-K}.

CIFAR-large. 2000 clients, 20 clients participating in each iteration, total batch size 500.

FEMNIST. 3550 clients, 20 clients each iteration, and batch size 600.

Implementation. We adopt the design of FL framework from FetchSGD [89] using PyTorch, and implemented MinMax, GSpar, and Top-K with HyperIBLT on its open source code. We use CS(k) and CSK(k) to represent CS and CSK using the parameter that results in $k\times$ vector size, where we define the data structure sizes of HyperIBLT, CS, and CSK as the vector size. Let K be the number of none zero values in one sampled set, $d = 3$ be the number of hash functions used in all three algorithms. For HyperIBLT, we set the length of HyperIBLT to be $W_0 = \frac{3.27K\times 1.25}{d}$, which enables HyperIBLT to decode successfully in all tests[6]. If HyperIBLT fails to decode, we double its size, change the hash functions and insert the gradients again, guaranteeing the success of decoding. For HyperIBLT/CS/CSK, we use $G = 4$ as introduced in § IV. For CS(1), we set $W_1 = \frac{12.5}{8d}W_0$ to control[7] the same data traffic compared with HyperIBLT. For CSK(1), we have $W_2 = \frac{12.5W_0 d - 4K}{8d}$, considering the extra $4K$ bytes used to store sampled keys.

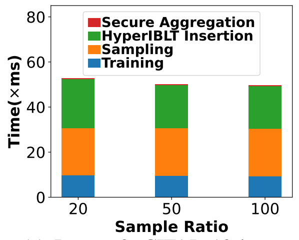
Test Accuracy (Fig. 4 and 5). The results show that HyperIBLT can achieve the best accuracy with $10\times$ size saving (network traffic and storage) compared with CS and CSK. We alter the Sample Ratio (R) to adjust the portion (*i.e.*, $\frac{1}{R}$) of the gradient vector to be uploaded. Under the same size, the CS(1) and CSK(1) averages 17.79% and 5.11% lower accuracy on CIFAR-small, 18.93% and 6.21% lower on CIFAR-large. CS(1) shows unsatisfying performance in Fig. 4(b), because CS(1) is too inaccurate to make the FL model converge. With $3\times$ size than HyperIBLT, CS(3) and CSK(3) are still 7.95% and 2.48% slightly lower that HyperIBLT under 100 sample ratio on CIFAR-small. With $10\times$ size, CSK(10) has nearly the same accuracy as HyperIBLT, but it is not secure for keys. Similar to the results of HyperIBLT+GSpar, when using MinMax, HyperIBLT shows a same degree of advantage. Due to the normal jitter of the accuracy curve, some data points in Fig. 5(b) show that HyperIBLT is slightly less accurate than CSK(3) and CS(3), which use $3\times$ larger size. For Top-K, we saved $3\times$ size instead of $10\times$ using MinMax and GSpar. Since MinMax and GSpar are more advanced and accurate, we believe their results (saving $10\times$ size) are more persuasive.

During the aggregation, both HyperIBLT and the One-hot (*i.e.*, applying SVA directly) are error-free (during the upload gradient process) under any Sample Ratio, because the size of HyperIBLT is set to be large enough to provide successful decoding. The comparison of the sizes will be shown later. As the sample ratio increases, the test accuracy decreases gradually. The decline is not caused by our aggregation but by sampling. In practice, the data curator can make a trade-off between communication overhead and accuracy to select a suitable sampling ratio (detailed discussion in MinMax [25]). The curator can also reduce the decline of accuracy by increasing the number of training rounds/epochs. We believe that with the development of FL, there will be more scenarios where sparse vectors need to be aggregated.

Computation Efficiency (Fig. 6). The computation time of HyperIBLT is in the same order-of-magnitude as other parts. One round of computation on local devices consists of four parts: local training, sampling, HyperIBLT insertion and SAP of HyperIBLT. We choose CIFAR-10-large and FEMNIST using HyperIBLT+MinMax Sampling under different sampling ratios as an example. Compared with One-hot, we spend extra

[6]Statistics reveal that the number of non-zero values in a union set of a group of 4 sampled gradients is always smaller than $3.27K$ in the whole federated learning process.

[7]Here, $\frac{12.5}{8} = \frac{0.5Byte\ Freq + 8Byte\ ValSum + 4Byte\ KeySum}{8Byte\ Counter\ in\ CS}$.


(a) Resnet-9, CIFAR-10-large.

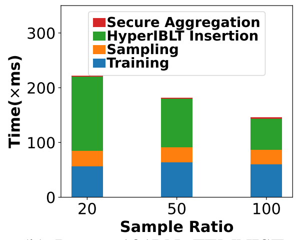
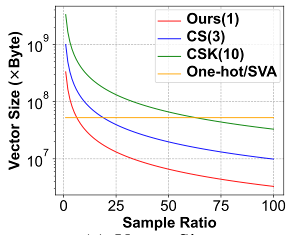
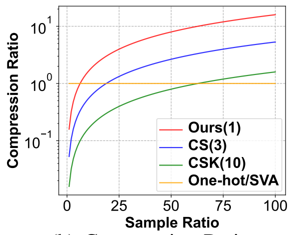
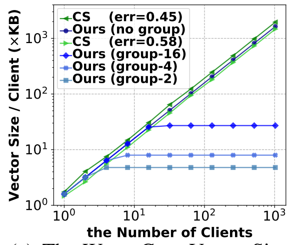

(b) Resnet-101LN, FEMNIST.

Fig. 6. Experiments on computational efficiency in FL.


(a) Vector Size.


(b) Compression Ratio.


(c) The Worst Case Vector Size.

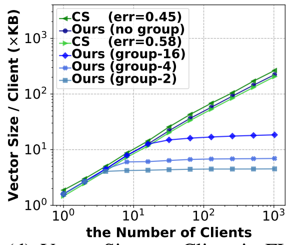

(d) Vector Size per Client in FL.

Fig. 7. The vector size that each client uploads.

computing overhead on the insertion of HyperIBLT, which will reduce the overall computation efficiency.

Communication Efficiency (Fig. 7). In Fig. 7(a), we illustrate the size that one local client uploads under different sample ratios. In Fig. 7(b), we show the communication compression ratio compared with FL without sampling. Our HyperIBLT will be more communication efficient than One-hot when the sample ratio is greater than 6.28. In other words, HyperIBLT can save the communication only when the vector is sparse enough. In Fig. 7(c), the largest vector size is observed when each client holds totally different keys (1.5925KB set for each). Since the size of the sum set increases linearly with the number of clients, the vector sizes of HyperIBLT and CS grows to keep the same accuracy. By dividing clients into constant size groups § III-B, the vector size stops growing. Fig. 7(d) shows a similar case when uploading the data from FL+MinMax with a sample ratio of 10000.

C. Experiments on global estimation tasks

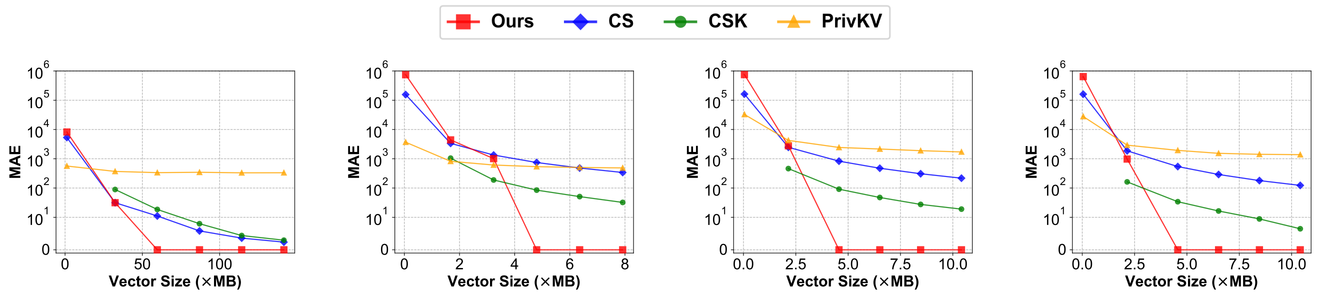
We conduct experiments on two distributed tasks: distributed data counting and finding global hot items, and compare the performance of HyperIBLT, CS, CSK, and PrivKV (a well-known DP solution). Compared with HyperIBLT, One-hot is nearly impractical. When working on 32-bit IPv4 addresses, One-hot vector requires a 2^{32} vector taking 16 GByte while HyperIBLT only takes less than 100MB ($<$ 1% 16GByte). When comparing HyperIBLT with PrivKV, we additionally tell all its clients the key set (*i.e.*, the set of all occurred keys in the sum set) as the key domain of PrivKV. The reason is as follows. PrivKV and other DP solutions for KV sets (*e.g.*, PCKV [71], PrivKVM [67], and Selective MPC [69]) assume that they are aware of the domain of keys and that the domain cannot be too large (*e.g.*, usually less than 10^4). In contrast, the size of our key domain is nearly unlimited (*e.g.*, 2^{32}, or 2^{64}) and is all we required. PrivKV performs poorly if it only knows the key as a 32-bit integer, therefore we additionally provide it with the key set, which gives PrivKV a greater advantage. The privacy budget of PrivKV is 1. Aside from the comparison on original datasets, we also evaluate their combinations with several sampling algorithms, including MinMax, Iceberg, and local Top-k.

Data Allocation: We simulate a 500-device-distributed setting, and the CAIDA dataset is partitioned in two ways:

1) *iid*: For each entry, we uniformly assign it to one of the devices. Each device contains 181K source IPs on average.
2) *non-iid*: For each entry, we assign it by hashing its 5-tuple. Each local device contains 50K source IPs on average.

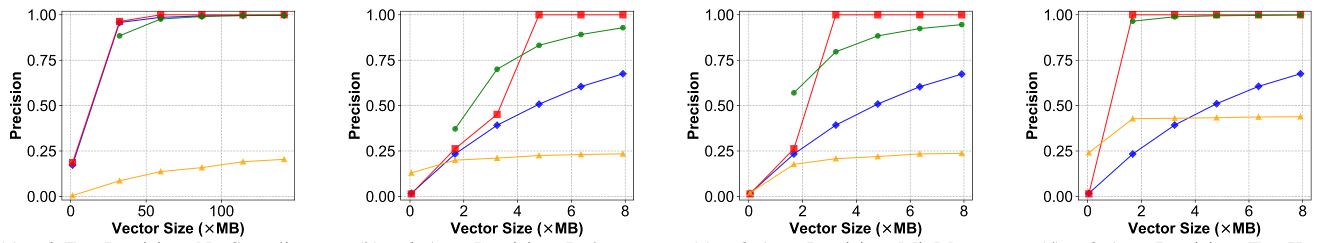
Task Implementation—Distributed data counting. In this task, we count the total number of times/entries that each source IP occurs in local devices. The **workflow** includes: (1) Select D entries on each local device (in chronological order) to participate in counting. Let S_u be the local set where the key is source IP and the value is the number of key's occurrences/entries. (2) If any sampling solution (*e.g.*, Iceberg) is combined, sample K source key-value pairs out in each local device, and ignore the others in the following procedure. Let $\tilde{S_u}$ be the set after sampling. When sampling is not combined, $\tilde{S_u} = S_u$. (3) Aggregate $\tilde{S_u}$ using HyperIBLT/CS/CSK/PrivKV. (4) Evaluate the result $\widehat{S}$ in the central server. Let S be the global sum set without sampling , $S = \sum_u S_u$. Let $\tilde{S}$ be the global sum set with sampling, $\tilde{S} = \sum_u \tilde{S_u}$. We calculate the **metrics** as follows. In the evaluation without sampling, we use the MAE and RMSE to quantify the error by the difference of S and $\widehat{S}$. In the evaluation with sampling, we have two approaches of calculating MAE and RMSE. To show pure aggregation error without contribution of sampling, we compare $\tilde{S}$ with $\widehat{S}$ to calculate the MAE & RMSE, called Aggregation (Agg.) MAE/RMSE. To show the actual error including sampling errors, we compare S with $\widehat{S}$, called Estimation (Est.) MAE/RMSE.

Accuracy *v.s.* Vector Size (Fig. 8). When HyperIBLT eliminates the errors, CS/CSK/PrivKV still lead to considerable errors under the same vector size. When the size is adequate, HyperIBLT has an obvious advantage, *i.e.*, practically zero-error. When the size is very compact, CS/CSK/PrivKV has small advantages against HyperIBLT. As shown in Fig. 8, on both the *iid* and *non-iid* datasets, with enough sizes enabling HyperIBLT to decode, it achieves zero error. While the MAE of CS/CSK/PrivKV approaches but cannot reach 0 when they are twice in size compared to HyperIBLT with 0 MAE. When the size is too small for HyperIBLT to decode, we just estimate the value (§ III-B), where HyperIBLT can perform nearly as well as CS.



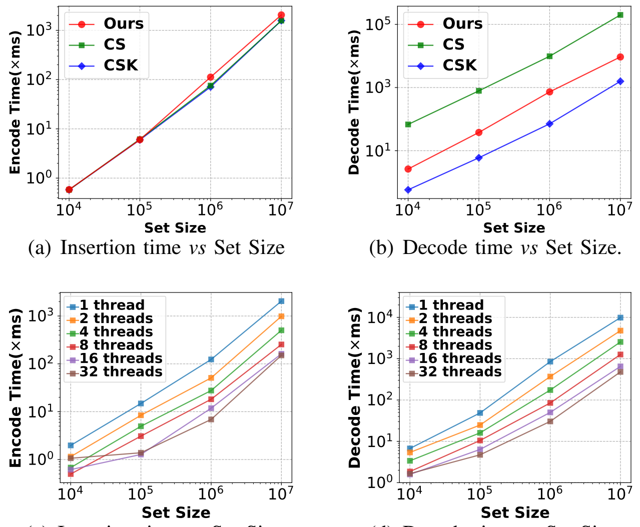
(a) *iid*, Est. MAE, No sampling. (b) *iid*, Agg. MAE, Iceberg. (c) *non-iid*, Agg. MAE, MinMax. (d) *non-iid*, Agg. MAE, Top-K.

Fig. 8. Distributed data counting on different datasets using HyperIBLT w/o or w/ different sampling algorithms.



(a) *iid*, Est. Precision, No Sampling. (b) *iid*, Agg. Precision, Iceberg. (c) *iid*, Agg. Precision, MinMax. (d) *iid*, Agg. Precision, Top-K.

Fig. 9. Estimating global hot items on *iid* dataset using HyperIBLT w/o or w/ different sampling algorithms.



(a) Insertion time *vs* Set Size (b) Decode time *vs* Set Size.

(c) Insertion time *vs* Set Size. (d) Decode time *vs* Set Size.

Fig. 10. Experiments on single-core and multi-core compute time

Task Implementation—Finding global hot items. We find the top-K most frequent source IPs in this task. Similarly, we use Aggregation (Agg.) Precision and Estimation (Est.) Precision to show the precision of HyperIBLT. The MAE in this section only calculates estimated values of real hot items.

Accuracy *v.s.* Vector Size (Fig. 9). Experiments show that HyperIBLT outperforms CS/CSK/PrivKV when the size is large enough. When the vector size goes larger than 50MB, the HyperIBLT becomes error-free with high probability, while the other solutions cannot eliminate errors.

Computational Efficiency (Fig. 10). We use an 18-core Intel(R) Core i9-10980XE with 4.6GHz boost frequency as our computing platform and compare the single thread compute time of HyperIBLT, CS and CSK. For any given sample size, we control the data traffic of the three algorithms to be the same. We set $R = 1.25$ for HyperIBLT. In Fig. 10(a), we show that compared to CS and CSK, HyperIBLT only has negligible tardiness when encoding. In Fig. 10(b), Sample Ratio is set to 100, it is found that HyperIBLT is slightly slower than CSK and way faster than CS in encoding. Experiments are also conducted to measure the multi-thread performance of HyperIBLT. When using p threads, we use p HyperIBLTs instead of one. For every entry, we insert it to one of the p HyperIBLTs by hashing. The encoding/decoding process of the p HyperIBLTs is independent, so the encoding/decoding time decreases linearly with the increase in thread number.

VI. Conclusion and Future Work

In this paper, we formalize the new problem of P_{Agg}^{KVS} and propose the KVSAgg framework to address it. The key innovation is a homomorphic transform algorithm HyperIBLT that brings security, efficiency and accuracy. We conduct rigorous mathematical analysis and perform the evaluation on three use cases including federated learning, distributed data counting, and finding global hot items. The results show that, compared with our baselines, KVSAgg using HyperIBLT simultaneously achieves the best security, orders-of-magnitude higher efficiency, and zero-error in nearly all cases. All codes are open-source anonymously. In the future, we plan to conduct robustness studies that allow a bounded portion of clients to be corroded by the server. Also, we will consider the client dropouts.

Acknowledgment

We thank all anonymous reviewers for their help in improving this paper. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, and National Natural Science Foundation of China (NSFC) (No. U20A20179, 61832001, and 62202296).

References

[1] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[2] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[3] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame *et al.*, "Safelearn: secure aggregation for private federated learning," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 56–62.

[4] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1253–1269.

[5] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *arXiv preprint arXiv:2009.11248*, 2020.

[6] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 479–489, 2021.

[7] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, "Billion-scale federated learning on mobile clients: A submodel design with tunable privacy," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[8] V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang, "Scalable private set union from symmetric-key techniques," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 636–666.

[9] J. Sun, X. Yang, Y. Yao, A. Zhang, W. Gao, J. Xie, and C. Wang, "Vertical federated learning without revealing intersection membership," *arXiv preprint arXiv:2106.05508*, 2021.

[10] C. Zhang, Y. Chen, W. Liu, M. Zhang, and D. Lin, "Optimal private set union from multi-query reverse private membership test," *Cryptology ePrint Archive*, 2022.

[11] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[13] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[14] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[15] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[16] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[17] F. Fu, H. Xue, Y. Cheng, Y. Tao, and B. Cui, "Blindfl: Vertical federated machine learning without peeking into your data," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 1316–1330.

[18] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 691–706.

[19] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 267–284.

[20] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[21] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[22] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1322–1333.

[23] L. Wang, Q. Pang, S. Wang, and D. Song, "Fed-χ^2: Privacy preserving federated correlation test," *arXiv preprint arXiv:2105.14618*, 2021.

[24] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2168–2181, 2020.

[25] Y. Zhao, Y. Zhang, Y. Li, Y. Zhou, C. Chen, T. Yang, and B. Cui, "Minmax sampling: A near-optimal global summary for aggregation in the wide area," in *Proceedings of the 2022 International Conference on Management of Data*, 2022.

[26] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *NeurIPS 2018*, 2018, pp. 1306–1316.

[27] Y. Ding, C. Niu, F. Wu, S. Tang, C. Lv, Y. Feng, and G. Chen, "Federated submodel averaging," *arXiv preprint arXiv:2109.07704*, 2021.

[28] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *NSDI 2014*. USENIX Association, 2014, pp. 275–288.

[29] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. Dhoot, A. Kumar, A. Agiwal *et al.*, "Mesa: Geo-replicated, near real-time, scalable data warehousing," 2014.

[30] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 323–336.

[31] Q. Zhao, M. Ogihara, H. Wang, and J. J. Xu, "Finding global icebergs over distributed data sets," in *PODS 2006*. ACM, 2006, pp. 298–307.

[32] M. A. Salahuddin, J. Sahoo, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on content placement algorithms for cloud-based content delivery networks," *IEEE Access*, vol. 6, pp. 91–114, 2017.

[33] B. Zolfaghari, G. Srivastava, S. Roy, H. R. Nemati, F. Afghah, T. Koshiba, A. Razi, K. Bibak, P. Mitra, and B. K. Rai, "Content delivery networks: state of the art, trends, and future roadmap," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–34, 2020.

[34] R. Vaarandi, "A breadth-first algorithm for mining frequent patterns from event logs," in *International Conference on Intelligence in Communication Systems*. Springer, 2004, pp. 293–308.

[35] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[36] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[37] S. K. Pasupuleti, S. Ramalingam, and R. Buyya, "An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing," *Journal of Network and Computer Applications*, vol. 64, pp. 12–22, 2016.

[38] Q. Wang, D. Chen, N. Zhang, Z. Ding, and Z. Qin, "Pcp: A privacy-preserving content-based publish–subscribe scheme with differential privacy in fog computing," *IEEE Access*, vol. 5, pp. 17 962–17 974, 2017.

[39] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, "Toward secure multikeyword top-k retrieval over encrypted cloud data," *IEEE transactions on dependable and secure computing*, vol. 10, no. 4, pp. 239–250, 2013.

[40] J. Min, X. Kui, J. Liang, and X. Ma, "Secure top-k query in edge-computing-assisted sensor-cloud systems," *Journal of Systems Architecture*, vol. 119, p. 102244, 2021.

[41] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2011, pp. 792–799.

[42] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[43] S. Li, L. Luo, and D. Guo, "Sketch for traffic measurement: design optimization application and implementation," *arXiv preprint*, 2020.

[44] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Databases. NOW publishers*, p. 15, 2011.

[45] "Source code of HyperIBLT and other supplementary materials." https://github.com/KVSAGG/KVSAGG.

[46] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, pp. 277–346, 2017.

[47] T. D. Nguyen, P. Rieger, M. Miettinen, and A.-R. Sadeghi, "Poisoning attacks on federated learning-based iot intrusion detection system," in *Proc. Workshop Decentralized IoT Syst. Secur.(DISS)*, 2020, pp. 1–7.

[48] B. Choi, J.-y. Sohn, D.-J. Han, and J. Moon, "Communication-computation efficient secure aggregation for federated learning," *arXiv preprint arXiv:2012.05433*, 2020.

[49] L. Kissner and D. Song, "Privacy-preserving set operations," in *Annual International Cryptology Conference*. Springer, 2005, pp. 241–257.

[50] J. H. Seo, J. H. Cheon, and J. Katz, "Constant-round multi-party private set union using reversed laurent series," in *International Workshop on Public Key Cryptography*. Springer, 2012, pp. 398–412.

[51] D. Many, M. Burkhart, and X. Dimitropoulos, "Fast private set operations with sepia," *ETZ G93*, 2012.

[52] A. Davidson and C. Cid, "An efficient toolkit for computing private set operations," in *Australasian Conference on Information Security and Privacy*. Springer, 2017, pp. 261–278.

[53] K. Shishido and A. Miyaji, "Efficient and quasi-accurate multiparty private set union," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2018, pp. 309–314.

[54] K. V. Jönsson, G. Kreitz, and M. Uddin, "Secure multi-party sorting and applications," *Cryptology ePrint Archive*, 2011.

[55] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang, "Privacy at scale: Local differential privacy in practice," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1655–1658.

[56] Y. Yang, Z. Zhang, G. Miklau, M. Winslett, and X. Xiao, "Differential privacy in data publication and analysis," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 601–606.

[57] A. Machanavajjhala, X. He, and M. Hay, "Differential privacy in the wild: A tutorial on current practices & open challenges," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1727–1730.

[58] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[59] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[60] Y. Wang and X. Cheng, "Prism: Prefix-sum based range queries processing method under local differential privacy," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 433–445.

[61] Y. Liu, S. Zhao, Y. Liu, D. Zhao, H. Chen, and C. Li, "Collecting triangle counts with edge relationship local differential privacy," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2008–2020.

[62] F. Jin, W. Hua, B. Ruan, and X. Zhou, "Frequency-based randomization for guaranteeing differential privacy in spatial trajectories," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 1727–1739.

[63] J. Duan, Q. Ye, and H. Hu, "Utility analysis and enhancement of ldp mechanisms in high-dimensional space," *arXiv preprint arXiv:2201.07469*, 2022.

[64] M. Zhou, T. Wang, T. H. Chan, G. Fanti, and E. Shi, "Locally differentially private sparse vector aggregation," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 422–439.

[65] T. T. Nguyên, X. Xiao, Y. Yang, S. C. Hui, H. Shin, and J. Shin, "Collecting and analyzing data from smart device users with local differential privacy," *arXiv preprint arXiv:1606.05053*, 2016.

[66] T. Wang, N. Li, and S. Jha, "Locally differentially private frequent itemset mining," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 127–143.

[67] Q. Ye, H. Hu, X. Meng, H. Zheng, K. Huang, C. Fang, and J. Shi, "Privkvm*: Revisiting key-value statistics estimation with local differential privacy," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[68] Q. Ye, H. Hu, X. Meng, and H. Zheng, "Privkv: Key-value data collection with local differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 317–331.

[69] T. Humphries, R. Akhavan Mahdavi, S. Veitch, and F. Kerschbaum, "Selective mpc: Distributed computation of differentially private key-value statistics," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1459–1472.

[70] X. Yuan, X. Wang, C. Wang, C. Qian, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 547–558.

[71] X. Gu, M. Li, Y. Cheng, L. Xiong, and Y. Cao, "{PCKV}: Locally differentially private correlated {Key-Value} data collection with optimized utility," in *29th USENIX security symposium (USENIX security 20)*, 2020, pp. 967–984.

[72] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 192–203.

[73] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[74] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "Cocosketch: high-performance sketch-based measurement over arbitrary partial key query," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 207–222.

[75] J. Li, Z. Li, Y. Xu, S. Jiang, T. Yang, B. Cui, Y. Dai, and G. Zhang, "Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams," in *KDD '20*. ACM, 2020, pp. 1574–1584.

[76] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1449–1463.

[77] Y. Wu, Z. Fan, Q. Shi, Y. Zhang, T. Yang, C. Chen, Z. Zhong, J. Li, A. Shtul, and Y. Tu, "She: A generic framework for data stream mining over sliding windows," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–12.

[78] K. Li and G. Li, "Approximate query processing: What is new and where to go? a survey on approximate query processing," *Data Science and Engineering*, vol. 3, pp. 379–397, 2018.

[79] H. Lan, Z. Bao, and Y. Peng, "A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration," *Data Science and Engineering*, vol. 6, pp. 86–101, 2021.

[80] P. Chen, Y. Wu, T. Yang, J. Jiang, and Z. Liu, "Precise error estimation for sketch-based flow measurement," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 113–121.

[81] Z. Liu, C. Kong, K. Yang, T. Yang, R. Miao, Q. Chen, Y. Zhao, Y. Tu, and B. Cui, "Hypercalm sketch: One-pass mining periodic batches in data streams," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[82] J. Gui, Y. Song, Z. Wang, C. He, and Q. Huang, "Sk-gradient: Efficient communication for distributed machine learning with data sketch," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[83] J. He, J. Zhu, and Q. Huang, "Histsketch: A compact data structure for accurate per-key distribution monitoring," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[84] P. Chen, D. Chen, L. Zheng, J. Li, and T. Yang, "Out of many we are one: Measuring item batch with clock-sketch," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 261–273.

[85] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[86] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *European Symposium on algorithms*. Springer, 2006, pp. 684–695.

[87] L. Luo, D. Guo, R. T. Ma, O. Rottenstreich, and X. Luo, “Optimizing bloom filter: Challenges, solutions, and comparisons,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2018.

[88] Y. Wu, J. He, S. Yan, J. Wu, T. Yang, O. Ruas, G. Zhang, and B. Cui, “Elastic bloom filter: deletable and expandable filter using elastic fingerprints,” *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 984–991, 2021.

[89] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, “Fetchsgd: Communication-efficient federated learning with sketching,” in *ICML 2020*, vol. 119. PMLR, 2020, pp. 8253–8265.

[90] K. G. Larsen, R. Pagh, and J. Tětek, “Countsketches, feature hashing and the median of three,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 6011–6020.

[91] H. Jowhari, M. Sağlam, and G. Tardos, “Tight bounds for lp samplers, finding duplicates in streams, and related problems,” in *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2011, pp. 49–58.

[92] G. T. Minton and E. Price, “Improved concentration bounds for count-sketch,” in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2014, pp. 669–686.

[93] Y. Li, Q. Zhu, Z. Lyu, Z. Huang, and J. Sun, “Dycuckoo: dynamic hash tables on gpus,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 744–755.

[94] Y. Wu, Z. Liu, X. Yu, J. Gui, H. Gan, Y. Han, T. Li, O. Rottenstreich, and T. Yang, “Mapembed: Perfect hashing with high load factor and fast update,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1863–1872.

[95] K. A. Ross, “Efficient hash probes on modern processors,” in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2006, pp. 1297–1301.

[96] E. Fenske, A. Mani, A. Johnson, and M. Sherr, “Distributed measurement with private set-union cardinality,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2295–2312.

[97] C. Hu, J. Li, Z. Liu, X. Guo, Y. Wei, X. Guang, G. Loukides, and C. Dong, “How to make private distributed cardinality estimation practical, and get differential privacy for free,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 965–982.

[98] “Murmur hashing v3 source codes,” https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

[99] “Bob hash,” http://burtleburtle.net/bob/hash/evahash.html.

[100] D. Ting, “Data sketches for disaggregated subset sum and frequent item estimation,” in *SIGMOD Conference 2018*. ACM, 2018, pp. 1129–1140.

[101] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, “Leaf: A benchmark for federated settings,” *arXiv preprint arXiv:1812.01097*, 2018.

[102] “The CAIDA Anonymized Internet Traces,” http://www.caida.org/data/overview/.