Efficient and Secure Quantile Aggregation of Private Data Streams

Xiao Lan [*] Hongjian Jin [†] Hui Guo [‡] Xiao Wang [§]

May 12, 2023

Abstract

Computing the quantile of a massive data stream has been a crucial task in networking and data management. However, existing solutions assume a centralized model where one data owner has access to all data. In this paper, we put forward a study of secure quantile aggregation between *private* data streams, where data streams owned by different parties would like to obtain a quantile of the union of their data without revealing anything else about their inputs. To this end, we designed efficient cryptographic protocols that are secure in the semi-honest setting as well as the malicious setting. By incorporating differential privacy, we further improve the efficiency by $1.1\times$ to $73.1\times$. We implemented our protocol, which shows practical efficiency to aggregate real-world data streams efficiently.

1 Introduction

Processing high-speed, large volumes of data is one of the core tasks in networking analysis and database systems. The seminal work by Alon, Matias, and Szegedy [1] formalized and popularized the concept of data stream algorithms, where the algorithms can only make one pass of the data. These algorithms are particularly useful in practice due to their small memory footprint and high processing speed. They have been designed for the aggregation of various data, such as sum, max/min, count, quantile, cardinality estimation, histogram, and heavy hitter [2, 3, 4, 5, 6]. These algorithms can be applied to large volumes of physical data, medical data, activity data, business data, and more. Based on these algorithms, more complex tasks, such as service quality management [7], DoS attack detection [8], large area environmental/wildlife habitat monitoring deployed in sensor networks [9], and query optimization [10], can also be tackled.

However, almost all these algorithms are proposed under the assumption that all data streams are within the same system or owned by one party, and thus the aggregation of all data could be performed in a central place. It may not be the case when multiple data streams from different sites with different security requirements need to be aggregated. A simple illustration of this scenario would be the joint analysis of hospital visit records from multiple hospitals. In this case, each of the data owners (i.e., the hospitals) is reluctant to share their raw data, which contains sensitive information about their patients, with each other. However, they need to aggregate their data for joint analysis. A simple example would be the joint analysis of hospital visit records from multiple hospitals. In this case, multiple mutually distrusting data owners (i.e., the hospitals) need to aggregate their data, but none of the data owners would agree to share their raw data that contains highly sensitive information about their patients. This could be the cause of policy restrictions or the fact that the data is too valuable to share with other parties directly.

[*] Sichuan University. E-mail:lanxiao@scu.edu.cn.
[†] Sichuan University. E-mail:jinhongjian545@163.com.
[‡] the State Key Laboratory of Cryptology. E-mail:guohtech@foxmail.com.
[§] Northwestern University. E-mail: wangxiao1254@gmail.com.

Several efforts have been made to realize aggregation in privacy-related applications involving multiple data owners. There are two broad approaches to this problem: enhancing existing aggregation algorithms in a centralized manner (such as in [11, 12, 13, 14, 15, 16, 17, 18] or using cryptographic tools or designing dedicated distributed cryptographic protocols (such as in [19, 20, 21, 22, 23, 24, 25, 26, 27, 28]) for specific computation tasks. In a centralized manner, differential privacy, the *de facto* standard of data privacy proposed by Dwork [29], is a common approach to enhance privacy. However, the data still needs to be given out to the aggregator though the data has a certain degree of privacy protection, which would lead to privacy leakage. Besides, a trusted aggregator is not always easy to find. In the distributed manner, the cryptographic protocols take the role of the trusted aggregator, but dedicated protocols are highly dependent on complex cryptographic tools and hard to follow and extend. Moreover, most cryptographic protocols under the category of secure multi-party computation (MPC) do not actually ensure the privacy of personal data in the input, even though their goal is to allow a set of mutually distrusting parties to compute any function of their choice while ensuring that nothing is revealed except the outcome of the computation. Therefore, the privacy of the input must be guaranteed by other means. Additionally, many existing works in the distributed setting require the participants to outsource their data, which results in significant communication and computation overhead for the applications.

Furthermore, existing works on secure aggregation mainly focus on secure heavy hitter [21, 12, 25, 28] histogram [27], cardinality estimation problem [23]. Few of them are on secure quantile computation [26, 17]. Quantiles, as known to us, are elements in a sorted dataset that represent specific ranks. They provide reasonably accurate information about the distribution of a dataset and are less sensitive to outliers than the moments. As an order statistic, it can also be used to resolve histograms, heavy hitters, frequency problems, and geometric computations, therefore it is a very basic and important kind of statistical calculation. Meanwhile, there is plenty of research focusing on efficient approximate quantile computation [30, 31]. Based on our careful research, we found that most of the quantile algorithms were proposed more than ten or twenty years ago, while the recent works mainly focus on the improvement of algorithms' performance in some extreme cases. Among them, the algorithm proposed by Shrivastava et al. [9], named as Q-Digest, is the first famous deterministic algorithm designed in a distributed model. Although Q-Digest is not the best choice for computing quantiles, it is the only deterministic mergeable summary for quantiles. Therefore, Q-Digest itself provides a merge algorithm that makes the distributed quantile computation possible.

To better motivate secure quantile aggregation in distributed privacy-preservation applications, we conduct our work. First, we choose a candidate algorithm (i.e., Q-Digest in this paper) that is mergeable so that each data owner can first locally aggregate their own data streams. Then, we design an MPC protocol to jointly compute a summary over the aggregation of all data. Furthermore, different from previous work, we use a Differential Privacy mechanism as a tool to protect the input of the MPC protocol without reducing the accuracy of the data.

However, the use of MPC introduces several challenges: 1) MPC protocols, especially those with malicious security, are highly expensive and thus we would like to minimize the use of MPC as much as possible; 2) MPC protocol requires the function to be represented in a way such that the execution and data access trace are all independent of the input (a.k.a. obliviousness), this means that the input size and the computation have to incur the worst-case cost to prevent leakage; 3) in the malicious setting, the data owners could behave arbitrarily and input an invalid local aggregation. Therefore, extra mechanisms are needed to efficiently check that all inputs are valid. The first challenge in secure quantile aggregation in distributed privacy-preservation applications can be addressed in two ways. Firstly, more efficient general MPC protocols [32, 33, 34] can help. Secondly, by using aggregation sketch to reduce the computation overhead brought to MPC. Our choice of Q-Digest is motivated by this consideration. The second challenge, regarding obliviousness, is faced by all MPC applications and is usually addressed by transforming algorithms with ORAM [35, 36, 37]. However, this approach increases both computation and communication complexity, making it unfeasible for complex algorithms. The last challenge is the key to realizing malicious security

but varies from problem to problem. In this paper, we design a dedicated input validity check to address this challenge.

In this paper, we design and implement a system for the secure aggregation of multiple private data streams efficiently. We focus on the problem of quantile estimation based on the Q-Digest algorithm [9], which is an algorithm designed in the distributed setting. Our system is based on provably secure protocol and can aggregate a dataset with size 10^9 and universe 2^{32} under 996 seconds in the semi-honest setting and 6596 seconds in the malicious setting, a much stronger security guarantee. Such impressive performance is enabled by designing new algorithms at the intersection of approximation algorithms, cryptography, and privacy. In summary, our contributions are efficient MPC protocols for quantile aggregation with privacy protection on distributed streams. Concretely,

1. Our first task is to support data aggregation in the semi-honest setting without any leakage. To this end, we design an efficient data-oblivious Q-digest merge algorithm called OMerge. Compared to a generic baseline, this protocol is significantly more efficient both asymptotically and concretely.
2. To explore further efficiency improvement, we propose a formal definition for secure two-party Q-digest aggregation that allows provable trade-offs between security and efficiency. We observe that the bottleneck in the above algorithm is the unnecessary padding and thus proposed an improved algorithm that can achieve better performance by revealing parties' actual data size but protected by differential privacy. To minimize the amount of noise needed, we proved that the global sensitivity of the digest size is 2, independent of all other parameters in the system. As a result, the overhead of differentially private leakage is only 40.5% under the universe of 2^8 and 0.8% under the universe of 2^{32}. See Section 4.
3. To achieve malicious security, we design an oblivious validity check algorithm OCheck that guarantees that the input is calculated correctly based on some data stream. The check can be represented efficiently as a circuit, and at the same time, can be checked using a zero-knowledge proof protocol rather than a full-blown malicious two-party computation, which further reduces its overhead. See Section 5.
4. Finally, we implement our system and benchmark its efficiency in both the semi-honest and malicious setting, using both real-world data and also synthetic data.

2 Preliminaries and Model

2.1 Notation

We use $:=$ to denote assignment and $=$ to denote equivalence. We use D to denote a dataset, where the elements of D are drawn from a universe $[U] = \{1, 2, \ldots, U\}$. We use a frequency vector $\boldsymbol{f}$ to represent D such that $\boldsymbol{f}[i]$ is the number of occurrences of i in D, where $i \in [U]$. We use $|D|$ to denote the size of D so $|D| = \sum_{i\in[U]} \boldsymbol{f}[i]$. We use T to denote a complete binary tree with U leaves. By convention, we use $[a, b]$ to denote $\{x | a \leq x \leq b\}$, use $A \cap B$ to join two sets A and B, use $A \setminus B$ to denote the subset which contains all the elements only in A but not in B, and use $A \cup B$ to denote the union of two sets A and B.

2.2 Q-Digest

Q-Digest [9] is a popular data structure mainly used to answer quantile queries in database and sensor network systems and is based on a complete binary tree built over the universe of the datasets. By denoting the universe of our dataset as $[U]$, we can build a complete binary tree T over the entire universe with depth $\log U$, in which every node is associated with a tuple $\langle \mathsf{id}, \mathsf{c} \rangle$ where id is the identifier of the node in the tree and c is a counter used to record the number of elements which lie in the certain range corresponding to the identifier of this node. We start with a dataset D which has a frequency vector $\boldsymbol{f}$ and the size $n = \sum_{i\in[U]} \boldsymbol{f}[i]$. In order to calculate the Q-Digest for D, a tree T is initialized such that the counter for the i-th leaf node is set to the frequency $\boldsymbol{f}[i]$, while the counters of all non-leaf nodes are set to 0. The

next step is to group the counter values into higher-level nodes through a compression procedure. In their original paper, the compression condition is referred to as digest properties, which depend on a compression parameter k and determines not only the size of Q-Digest but also the approximation error. To simplify, letting $\theta = \lfloor n/k \rfloor$ to be the threshold, the compression condition checked against any node with $\langle \mathsf{id}, \mathsf{c} \rangle$ is whether $\mathsf{c} + \mathsf{c_p} + \mathsf{c_s} \leq \theta$, where $\mathsf{c_p}$ and $\mathsf{c_s}$ are the counter values of the parent and sibling of node id. When $\mathsf{c} + \mathsf{c_p} + \mathsf{c_s} \leq \theta$, the counter $\mathsf{c_p}$ is replaced with $\mathsf{c} + \mathsf{c_p} + \mathsf{c_s}$, and the counter value c and $\mathsf{c_s}$ are set to 0. The check is executed from bottom to top, from left to right, and then the final Q-Digest can be constructed. A more detailed description of the original Q-Digest is given in Appendix A.

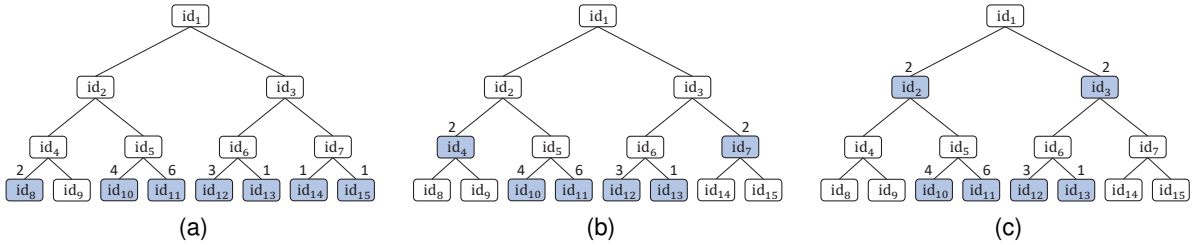


Figure 1: Building the Q-Digest on a dataset where $n = 18$, $U = 8$, $\boldsymbol{f} = (2, 0, 4, 6, 3, 1, 1, 1)$ when the compression parameter $k = 5$. In this case, $\theta = \lfloor n/k \rfloor = \lfloor 18/5 \rfloor = 3$. In Figure 1a, the compression condition of id_8, id_{14} (and id_{15}) is met so the nodes will be compressed with their parents id_4 and id_7 respectively. Thus we arrive at the state in Figure 1b. At this point, the compression condition of id_4 and id_7 is still met, so the nodes are compressed and we arrive at Figure 1c. Since the compression condition of id_2 and id_3 is not met, the compression procedure finishes and we arrive at the final Q-Digest in Figure 1c.

In Figure 1, we provide an example to show the Q-Digest building procedure for a dataset of size $n = 18$ and $U = 8$. For every node in the tree, we mark the identifier in the box and the non-zero counter value above the box. The boxes in blue color are the nodes with non-zero counter values. In Figure 1a, the tree is constructed on the dataset with frequency vector $\boldsymbol{f} = (2, 0, 4, 6, 3, 1, 1, 1)$; Figure 1b and Figure 1c show the intermediate compression results executed on each level bottom up; finally, the Q-Digest of the dataset is a set which contains all the nodes of blue color in Figure 1c.[1]

2.3 Differential Privacy

Differential privacy is a rigorous and composable privacy notion in the context of data analysis. It provides a generic mechanism to protect the privacy of any individual whose data is contained in the dataset and aims at realizing privacy-preserving data analysis.

Following the definition of Dwork and Roth [29], we define the distance between datasets based on the ℓ_1 norm. Given two datasets D and D', their distance is defined as $\|D - D'\|_1 := \sum_{i \in [U]} |\boldsymbol{f}[i] - \boldsymbol{f}'[i]|$. Then, we say that two datasets D and D' are neighboring if their distance is at most 1.

Definition 1. (ϵ, δ)**-Differential Privacy.** *A randomized algorithm M with domain $\mathbb{N}^U$ is (ϵ, δ)-differentially private if for any two neighboring datasets D and D' and for any set S of possible outputs, we have*

$$\Pr[M(D) \in S] \leq e^{\epsilon} \Pr[M(D') \in S] + \delta. \tag{1}$$

Definition 2. Global Sensitivity. *The global sensitivity of a function $f : \mathbb{N}^U \to \mathbb{R}^m$ is:*

$$\Delta f = \max_{D, D' \in \mathbb{N}^U, \|D - D'\|_1 = 1} \|f(D) - f(D')\|_1. \tag{2}$$

[1] In the rest of this paper, we regard the data structure Q as a set, where the element is a tuple $\langle \mathsf{id}, \mathsf{c} \rangle$, and the set can be ordered by the tuple's identifier id.

Theorem 1. Post-Processing. *Let $f^* : \mathbb{N}^U \to R$ be a randomized algorithm that is (ϵ, δ)-differentially private, $g : R \to R'$ be an arbitrary randomized algorithm. Then $g \circ f^* : \mathbb{N}^U \to R'$ is (ϵ, δ)-differentially private.*

The property of post-processing was proved by Dwork and Roth [29, Proposition 2.1].

2.4 MPC Security

2.4.1 Adversary Model and Security Definitions

In this paper, we consider two different security settings: the semi-honest model and the malicious model. In the semi-honest model, the adversaries are called semi-honest adversaries, who control one of the parties at the onset of the computation and follow the protocol specification exactly. In the malicious model, the adversaries are called malicious adversaries, who may arbitrarily deviate from the protocol specification. The goal of all kinds of adversaries is to learn more information than is allowed by taking any action they want and are allowed during the protocol execution.

To formally define the security of MPC, we use the real-ideal paradigm. In this paradigm, a clear "ideal" world is introduced that implicitly captures all security guarantees and defines security in relation to this ideal world. In real-world execution, parties interact with semi-honest/malicious adversaries and execute an MPC protocol. In the ideal world, parties interact with a simulator and ideal functionality, which plays the role of a trusted party that is not corruptible and captures the security properties of the MPC protocol. A real-world protocol is considered secure in the semi-honest/malicious setting if any effect that an adversary can achieve in the real world can also be achieved by an adversary in the ideal world, by carefully constructing a simulator in the ideal world to simulate the real-world view.

For more formal definitions and detailed discussions, readers can refer to the books of Lindell [38] and Evan et al. [39].

2.4.2 Security Model of Two-Party Computation with Leakage

In this paper, we follow the security model of two-party computation with leakage proposed by Groce *et al.* [40]. In their work, the ideal functionality with leakage provides a class $\mathcal{L}$ of allowable leakage functions such that the adversary is allowed to choose the leakage function from $\mathcal{L}$. Besides, the adversary is also able to corrupt a party and obtain the leakage of the honest party's input both before and after choosing his input. In their setting, a protocol π securely realizes function $f = (f_1, f_2)$ with $\mathcal{L}$ leakage if π is a UC-secure protocol for this functionality. When the leakage function chosen from $\mathcal{L}$ is (ϵ, δ)-differentially private under the definition of neighboring on the input, the protocol π is said to realize f with (ϵ, δ)-DP leakage. For more details, please refer to their paper [40].

In our work, we consider a special case of the leakage functionality for two-party Q-Digest computation, where the adversary is only given one chance to query the leakage function. The description of our functionality is shown in the next section.

3 Setup and Functionality

3.1 Overview

As we mentioned in Section 2.2, two or more Q-Digests can be merged to get a new one by running the Merge algorithm. When two parties want to merge their locally computed Q-Digests, the Merge algorithm (Algorithm 4) needs to be run by one of the participants or a third party. In this case, the local Q-Digests need to leak to the executor. To prevent the local Q-Digest from being revealed, we apply generic secure

Functionality $\mathcal{F}_{\mathsf{QDigest}}$

Parameters: $[U]$ is the universe, $L^*(\cdot)$ is a leakage function.

1. Upon receiving $(\mathsf{input}, i, \boldsymbol{f}_i)$ from P_i, store $\boldsymbol{f}_i$ in the memory and send $(\mathsf{received}, i)$ to both parties. Ignore all subsequent messages of the form $(\mathsf{input}, i, \cdot)$. If both $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ appear in the memory, compute a merged Q-Digest $Q = \mathsf{Merge}(\mathsf{Compress}(\boldsymbol{f}_1), \mathsf{Compress}(\boldsymbol{f}_2))$ and send Q to both parties.
2. Upon receiving $(\mathsf{leakageQuery}, i)$ from the corrupted party $\mathcal{S}$, send $L^*(|\mathsf{Compress}(\boldsymbol{f}_i)|)$ to $\mathcal{S}$. Ignore all subsequent messages of the form $(\mathsf{leakageQuery}, \cdot)$.

Figure 2: The functionality of two-party Q-Digest with leakage. To simplify our functionality, we omit the compression parameter k of the Compress algorithm defined in Appendix A, and also the procedure of transforming the frequency vector to the form of input described in Algorithm 3.

computation to this task by expressing the Merge algorithm as a circuit. However, if we directly take the local Q-Digests of two parties as inputs of the secure computation, whether specific nodes are in the Q-Digests would be leaked from the size of Q-Digest, which would further reflect the distribution of the input datasets. So we require adding some dummy nodes to hide the actual size of the local Q-Digest such that any party cannot get the exact information about the size of another party's local Q-Digest. Differentially privacy in this setting would ensure that one party cannot learn whether a particular node is in the other party's Q-Digest or not.

In this setting, we have two parties, and each of them has a dataset presented by a frequency vector. They want to merge their locally computed Q-Digest, with the security guarantee that any one of them cannot get more than differentially private information about another party's dataset from the execution of secure computation between them.

3.2 Functionality of Two-party Q-Digest with Leakage

In this section, we formally define the functionality $\mathcal{F}_{\mathsf{QDigest}}$ as shown in Figure 2. $\mathcal{F}_{\mathsf{QDigest}}$ securely evaluates Q-Digest of two parties with (differentially private) leakage of the size of the local Q-Digest following the two-party computation with a leakage model [40]. Different from the original functionality proposed by them, the adversary in our functionality only has one chance to query the leakage function. So our functionality is a special case of their original model.

4 Our Protocol in the Semi-honest Setting

In the semi-honest setting, the adversary is one who corrupts parties but follows the protocol as specified. The security of the protocol is following a real-ideal paradigm. In this section, we show the construction of a semi-honest protocol Π_{sQDigest} to realize $\mathcal{F}_{\mathsf{QDigest}}$. Recall that the Merge algorithm itself should be data oblivious to be used in secure computation. In the following, we design an oblivious version of Merge algorithm called OMerge in Section 4.1 and show how the algorithm can be used for secure computation in Section 4.2. We also give a rigorous analysis of the global sensitivity of the output of the Compress algorithm in Section 4.3.

4.1 OMerge Algorithm

Before the introduction of the OMerge algorithm, let us explain why the original Merge algorithm is not data oblivious. In fact, as shown in Algorithm 4, the Merge algorithm is constructed on the Compress algorithm. In the Compress algorithm, every node will be checked, and the non-empty ones will be input to the CompressNode algorithm to do further operations such as compression. No matter whether the node is compressed or not, the counter value of its sibling and parent will be accessed. Different from the compression procedure of taking the original dataset as input, which contains all the empty nodes. The merging of two Q-Digest only takes the non-empty nodes as input. Therefore, when a specific node is checked, even if the nodes are in order[2], we are still not sure the node's parent is in which place because the position of the node's parent is data-dependent. That is to say, the structure of the Q-Digest can reflect the distribution of the input dataset to some extent. Therefore, by observing the access pattern, some information about the original dataset would be leaked. In this section, we show how to enhance the Merge algorithm to obtain a data oblivious version algorithm OMerge.

4.1.1 Algorithm Description

Input and Initialization. The inputs of the OMerge algorithm are two sets of parameters such that each contains a Q-Digest Q_i (with each tuple defined below), the size n_i ($n_i = \sum_{j\in[U]} \boldsymbol{f}_i[j]$) of the dataset corresponding to Q_i, the size q_i of Q-Digest Q_i (i.e., the number of tuples in Q_i), the compression parameter k used to calculate Q_i, and the depth d of the q-digest tree used to calculate Q_i, where $i \in \{1, 2\}$.[3] Without loss of generality, we explain the set Q_1. Q_1 contains all the nodes output by Compress algorithm (Algorithm 3) and also a certain number of dummy nodes (i.e., tuples). Different from the setting in Compress algorithm, now we define every tuple in Q_1 as $\langle \mathsf{id}, \mathsf{c}, \mathsf{isDummy}, \mathsf{isParent}\rangle$, where id and c are defined as in Section 2.2, isDummy is a binary flag to indicate whether the tuple is valid or not, and isParent is also a binary flag to indicate whether the tuple is mapped from its parent or not. The initial settings for each tuple are as follows. First, based on the observation that the compression operation on each node will check the sibling and parent of the node, we modify the identifier of every non-leaf node to the identifier of its left child and call this operation a push operation. This operation is the key to realizing data obliviousness because by doing this, the node and its sibling and parent are next to each other when we order them by identifier. Therefore, there is no need to search for the whole set to find them. To distinguish the pushed tuple, the flag isParent of the pushed tuple is set to 1. Therefore, at the very beginning, all non-leaf nodes are identified with the identifier of its left child, and the binary flag isParent is set to 1. Second, for all nodes output by Compress algorithm, the counter c remains unchanged and the binary flag isDummy is set to 0. In contrast, the binary flag isDummy is set to 1 for all the dummy nodes being added to hide the actual size of the Q-Digest.

Intuition. The high-level idea of the OMerge algorithm is first taking the union of the two sets and adding the counter value for the tuples with the same identifier and the same parent flag (to distinguish the original tuple and the pushed tuple) to get a merged list Q, and then repeating the compression process node by node from bottom level to up level hierarchically by scanning the whole merged list Q each time. The list Q is resorted to updating the intermediate merging result once the compression procedure of each level finishes. Finally, the set Q is output, which contains all the nodes updated by the algorithm, while only the nodes with predicates isParent and isDummy set to 0 at the same time are the valid output of the OMerge algorithm.

[2]By regarding the data structure as a set, every tuple as an element, the set can be ordered by the identifier in each tuple, and this could help to execute the Merge algorithm.

[3]As with the original Merge algorithm, OMerge algorithm requires two Q-Digests to have the same universe and the same compression parameter.

For the compression of the node with identifier id, we do the following operations. We first check 1) if the node is at the level we are handling now, 2) if the dummy flag is 0 which means that the node is not a dummy or not processed before, 3) if the next node is not the sibling or parent of the node id, where we define a function $\mathsf{Parent}(x) := x/2$ to do the check because only the node and its sibling and parent would get the same output from $\mathsf{Parent}(\cdot)$ with their identifier as input. If all the conditions are satisfied, it means that we have scanned all the necessary nodes that are needed to execute the further compression procedure. Whether the nodes should be compressed to the upper level or remained at this level is considered separated under the following three cases:

A. neither the sibling nor the parent of id is in Q or the node itself is a parent node (with the flag isParent set to 1). When the counter of the node is not bigger than the threshold, a compression operation is executed, otherwise, the node is kept;
B. either the sibling or the parent of id is in Q. When the sum of the two nodes' counters is not bigger than the threshold, a compression operation is executed, otherwise, the nodes are kept;
C. both the sibling and the parent of id are in Q. When the sum of the three nodes' counters is not bigger than the threshold, a compression operation is executed, otherwise, the nodes are kept.

When the compression operation is executed in corresponding cases (A, B, or C), the status of the node's sibling and parent would also be changed if they exist. On the other hand, when the node should be remained in corresponding cases (A, B, or C) and there is a parent node, we need to recover the parent node by setting the node's identifier back to its parent's identifier and resetting the isParent flag to 0.

The complete description of OMerge algorithm is given in Appendix B.1.

4.1.2 Security Analysis

The security of OMerge is reflected in two aspects: correctness and obliviousness. For correctness, since the algorithm handles the node one by one level by level strictly following the line of Merge algorithm described in Section 2.2, the output should be the same as the original Merge algorithm. The only difference comes from our algorithm's design that we do not delete any empty node in Q, which is the key to guaranteeing privacy. Nevertheless, we distinguish valid nodes from invalid ones by setting some predicates, so only the nodes with predicates isParent and isDummy set to 0 at the same time are the valid output of the OMerge algorithm. For the obliviousness, from the description of Section 4.1, since all the access to Q is input independent, the oblivious nature of the OMerge algorithm is not hard to see.

4.2 Semi-Honest Two-party Q-Digest Protocol

As shown in Figure 3, we construct a two-party protocol Π_{sQDigest}, which realizes $\mathcal{F}_{\mathsf{QDigest}}$ in the semi-honest model and provides differentially private leakage of the size of local Q-Digests.

4.2.1 Protocol Description

The protocol Π_{sQDigest} consists of local computation and two-party interaction. In the first step: each party locally executes the Compress algorithm to get a Q-Digest from the frequency vector, adds a certain number of dummy nodes to their local Q-Digest Q_i such that the size of the Q-Digest equals the output of the leakage function $L^*(|Q_i|)$. In the second step, the two parties run a two-party computation to calculate the OMerge algorithm with their padded Q-Digest as inputs to get a two-party merged Q-Digest.

4.2.2 Security Analysis

We prove the security of protocol Π_{sQDigest} in the semi-honest model.

Protocol Π_{sQDigest}

Parameters: $[U]$ is the universe, k is compression parameter of Compress algorithm defined as Algorithm 3, $L^*(\cdot)$ is a leakage function.

Inputs: Party P_i for $i \in [2]$ inputs a frequency vector $\boldsymbol{f}_i$ with $n_i = \sum_{j \in [U]} \boldsymbol{f}_i[j]$.

The protocol:

1. Each party P_i locally computes the following:
   a. Generate a list Q_i based on $\boldsymbol{f}_i$, and compute $\mathsf{Compress}(Q_i, n_i, k)$ to get the local Q-Digest Q_i.
   b. Execute leakage function $L^*(q_i)$ where $q_i := |Q_i|$ to get the noised size q_i'.
   c. Add dummy nodes until the size of the Q-Digest Q_i equals q_i'. The tuple associated with each node is set according to the description of input for OMerge algorithm in Section 4.1.1. Notably, the predicate isDummy is set to 1 for every dummy node.
2. Two Parties run a secure computation of OMerge algorithm with their input Q_i to get the merged Q-Digest Q.

Figure 3: Semi-honest two-party Q-Digest with leakage.

Theorem 2. *If $L^*(\cdot)$ is (ϵ, δ)-differentially private function, then the protocol Π_{sQDigest} shown in Figure 3 securely realizes $\mathcal{F}_{\mathsf{QDigest}}$ with (ϵ, δ)-DP leakage against semi-honest adversaries in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model.*

Proof. The correctness is straightforward based on the correctness of OMerge as shown in Section 4.1.2. Now we show a simulator $\mathcal{S}$ that generates a view indistinguishable from an honest execution of the protocol in the case that a single party is corrupted. Let us assume that the corrupted party is P_c and the honest party is P_h. First, $\mathcal{S}$ has the input of the frequency vector $\boldsymbol{f}_c$ of the corrupted party. Once $\mathcal{S}$ receives a message $(\mathsf{received}, P_h)$ from the functionality, $\mathcal{S}$ sends $(\mathsf{leakageQuery}, P_h)$ to $\mathcal{F}_{\mathsf{QDigest}}$ and gets the size of the honest party's Q-Digest with dummy nodes. $\mathcal{S}$ also simulates $\mathcal{F}_{\mathsf{2PC}}$ with the input such that the size is the same as the size returned by the leakage query. $\mathcal{S}$ also sends $(\mathsf{input}, P_c, \boldsymbol{f}_c)$ to $\mathcal{F}_{\mathsf{QDigest}}$ and gets the the merged Q. Once the adversary provides input to the $\mathcal{F}_{\mathsf{2PC}}$, $\mathcal{S}$ returns Q to the corrupted party.

Also note that we assume the leakage function $L^*(\cdot)$ is (ϵ, δ)-differentially private. Therefore, Π_{sQDigest} securely realizes $\mathcal{F}_{\mathsf{QDigest}}$ with (ϵ, δ)-DP leakage in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model. □

4.3 Achieving Differentially Private Leakage for the Size of Local Q-Digest

In order to hide the size of the local Q-Digest such that any party cannot get the true size of the other participant from running the two-party secure computation of OMerge algorithm, some dummy nodes should be added to the local Q-Digests. In our setting, the local Q-Digest is computed by the Compress algorithm. By defining a function $L(\cdot)$ which outputs the size of the Q-Digest calculated by the Compress algorithm, we already know that there are multiple ways to achieve the differential privacy for $L(\cdot)$. One is by using Laplace mechanism [29], and the other one is by adopting the truncated Laplace mechanism [41]. Both of them add noise based on the parameters decided by the exact value of the sensitivity of $L(\cdot)$. Therefore, in this section, we analyze the sensitivity of $L(\cdot)$, which is the key to determining the size of the dummy nodes to be added to guarantee that the Q-Digest size with noise satisfies differential privacy.

We first list all the notations that will be used in this section in Table 1 and provide an example for two datasets D and D' in Figure 4 to help understand our notations.

Table 1: Notations.

Notations	Meanings
$L(\cdot)$	the function that outputs the size of the Q-Digest calculated by Compress;
ΔL	the global sensitivity of $L(\cdot)$;
$L^*(\cdot)$	the leakage function defined on $L(\cdot)$;
D	a dataset of size n with frequency vector $\boldsymbol{f}$ where $n = \sum_{i\in[U]} \boldsymbol{f}[i]$ and $[U]$ is the universe;
T	the complete binary tree used to calculate the Q-Digest of D, every node in the tree is associated with a tuple $\langle \mathsf{id}, \mathsf{c}\rangle$, the counter value c for the nodes in the tree would be changed during the processing procedure;
Q	the Q-Digest of D, which contains all non-empty nodes of T after the compressing procedure finishes;
q	the size of the Q-Digest Q, which is the number of nodes in Q;
Q_t	the intermediate Q-Digest after merging t times against D, where $t \in [0, \log U]$;
σ	an item in the universe $[U]$ such that for the frequencies of dataset D and D', there is $\boldsymbol{f}[\sigma] - \boldsymbol{f}'[\sigma] = 1$ and $\sum_{i\in[U]\setminus\sigma}(\boldsymbol{f}[i] - \boldsymbol{f}'[i]) = 0$;
S	the set of nodes lying on the path from the leaf node σ in T and their siblings after the whole compression procedure finishes, which contains all the nodes (both empty and non-empty nodes);
q_S	the size of the non-empty nodes of S, i.e., the size of $Q \cap S$;
m	the number of distinct levels for the non-empty nodes in S except for the root node;
s	the size of the non-empty leaf nodes in S;
r	a one-bit binary to indicate whether the root node is in Q or not, where $r = 1$ means that the root node is in Q, otherwise, $r = 0$.

The notations T', Q', q', Q'_t, S', q'_S, m', s' and r' are defined in the similar way for D'.

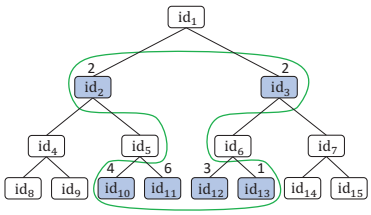
Following Definition 2, we focus on two datasets D and D' with frequency vectors $\boldsymbol{f}$ and $\boldsymbol{f}'$ such that $\sum_{i\in[U]} \left|\boldsymbol{f}[i] - \boldsymbol{f}'[i]\right| = 1$ (where $[U]$ is the universe). Without loss of generality, we consider the case that $\sum_{i\in[U]}(\boldsymbol{f}[i] - \boldsymbol{f}'[i]) = 1$ in our following analysis.

Based on our understanding of Q-Digest algorithm, we have the following two observations.

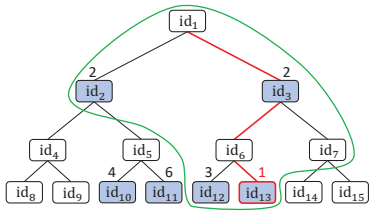
Fact 1. *For the node identified with* id *in* Q *except for the root and leaves, the sibling of* id *is also in* Q.

The fact shows that if an inter-level node id exists in Q, then its sibling id_s must also be in Q. We can prove it by contradiction. Assume id is in Q while id_s is not. It means that the counter value of node id exceeds the threshold while the counter value of id_s is 0. Note that the counter value of id is compressed from its children. So in our assumption, id's children have already violated the compression condition and should be kept but not merged to node id, which contradicts the assumption that the node id is in Q. So we have Fact 1.

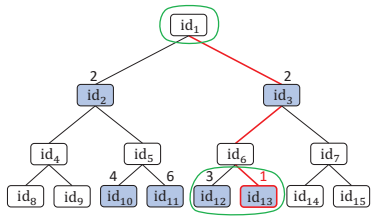
According to Fact 1, q_S could be expressed as one of the following two forms: 1). $q_S = 2(m-1)+r+s$, where there are $s \in \{1, 2\}$ leaf nodes remains in $Q \cap S$ so the leaf nodes are counted and contribute 1 to m; 2). $q_S = 2m + r$, where there is no leaf nodes in $Q \cap S$. q'_S can be expressed in the same way. The mentioned expressions of q_S and q'_S will be used in the proof of Theorem 3.



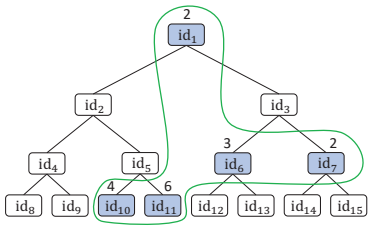
(a) The set of Q, which contains all the non-empty nodes picked out by the green coil in T.



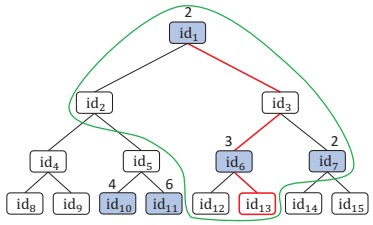
(b) The set of S, which contains all the nodes lying on the path from the leaf node σ to the root (which has been marked with red line) and their siblings in T.



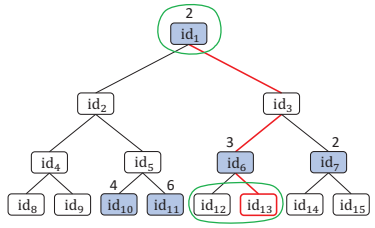
(c) $r = 0$, $s = 2$, $m = 2$ because 1) the root node is empty and the leaf nodes id_{12} and id_{13} are non-empty, so the value of $r = 0$ and $s = 2$; 2) except the root level, there are two levels in S such that the nodes id_2, id_3, id_{12} and id_{13} are non-empty, so $m = 2$.



(d) The set of Q', which contains all the non-empty nodes picked out by the green coil in T'.



(e) The set of S', which contains all the nodes lying on the path from the leaf node σ to the root (which has been marked with red line) and their siblings in T'.



(f) $r' = 1$, $s' = 0$, $m' = 1$ because 1) the root node is non-empty but the leaf nodes id_{12} and id_{13} are empty, so the value of $r' = 1$ and $s' = 0$; 2) there are only one level in S' such that the nodes id_6 and id_7 at this level are non-empty, so $m' = 1$.

Figure 4: The example of some notations in the tree T and T' constructed on D and D'. In this example, D is the dataset from Figure 1, and D' is another dataset with the universe $U = 8$, size $n' = 17$, and frequency vector $\boldsymbol{f}' = (2, 0, 4, 6, 3, 0, 1, 1)$. So $\sum_{i\in[U]}(\boldsymbol{f}[i] - \boldsymbol{f}'[i]) = 1$ and $\sigma = 6$. By setting the compression parameter as $k = 5$, we can calculate the Q-Digest Q and Q' for D and D'.

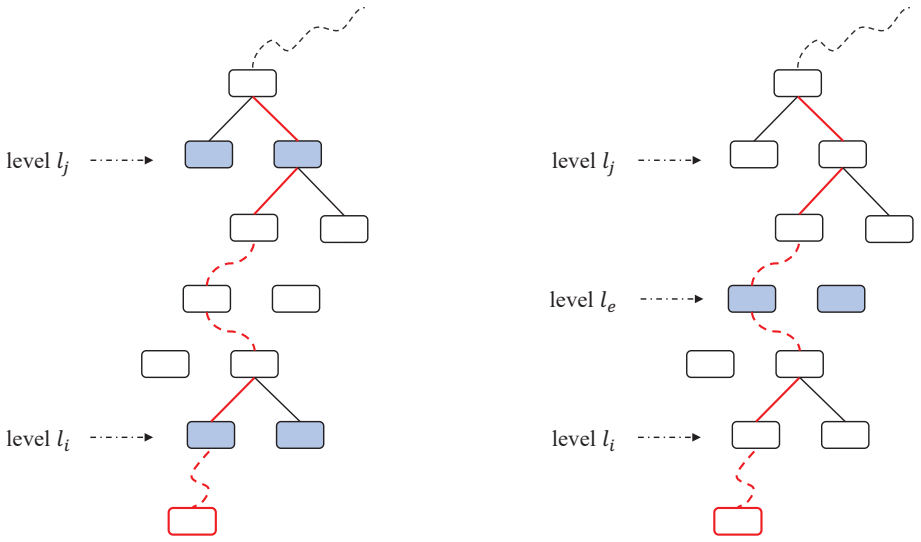
Fact 2. *Q and Q' only differ in S and S'.*

The fact shows that Q and Q' only differ in S and S'. We can also prove it by contradiction. Assume that there is a node that has different counter values in $Q \setminus S$ and $Q' \setminus S'$. It means that by tracing back to the leaf level, there must be some differences for the leaf nodes in the subtree rooted with the observed node in two trees. It contradicts the assumption that D and D' only differ at σ and $\sigma \in S$. So we have Fact 2.

By the above analysis, we know that the counter values for the node with the same identifier in S and S' would be different. In Lemma 1, we show that by ordering the nodes in S and S' with their identifiers, for any non-empty nodes next to each other at two levels in one set, if there is not any non-empty node at the two levels in another set, there must be non-empty nodes at a level between the two levels. We also draw a picture as shown in Figure 5 to give a more intuitive description of Lemma 1.

Lemma 1. *Let $l_i > l_j$ be the two levels with non-empty nodes in S such that between the two levels, there is not any non-empty node in S. Suppose that the nodes in S' at level l_i and l_j are all empty. Then there exist two nodes in S' at level l_e such that $l_i > l_e > l_j$.*

The concrete proof of Lemma 1 is given in Appendix C.1.



(a) The local view of T.

(b) The local view of T'.

Figure 5: An intuitive view of Lemma 1.The figure shows the local views for nodes in S (resp. S') between level l_i and l_j of tree T (in Figure 5a) and T' (in Figure 5b). The solid red line means that the two nodes are the relationship between parent and child, while the dotted red line indicates that several levels have been omitted between the two nodes.

We also have the following corollaries based on Lemma 1. First, without the limitation that there do not exist any non-empty nodes at level l_i and l_j, Corollary 1 is straightforward.

Corollary 1. *The compression condition is violated by nodes alternately or simultaneously in two trees, i.e., if l_i and l_j are the two levels with non-empty nodes in S such that between the two levels, there is not any non-empty node in S, then there exist nodes in S' at level l_e such that $l_i \geq l_e > l_j$ or $l_i > l_e \geq l_j$.*

Second, by comparing the counter values of the nodes at the same level where the compression procedure is to execute in two trees, we could infer that the next time for the event that the compression condition is violated by nodes (the corresponding nodes will be kept) to happen must be in the tree which has bigger counter values at the level we are observing.

Corollary 2. *Assume that the compression now proceeds at level l_p, the sum of the counter values for the nodes at level l_p in S is c and that in S' is c', if $c > c'$ (resp. $c < c'$), the next time for the compression condition be violated must happen in T (resp. T'); if $c = c'$, the two trees will be compressed in the same way from that level on.*

Third, since the compression condition is alternately violated by the nodes in two trees and the first time the nodes to violate the compression condition must happen in tree T. Every time the compression condition is violated, the corresponding nodes will be kept and the value of m will be added by 1. Therefore, we have Corollary 3.

Corollary 3. *Let D and D' be two datasets with $\sum_{i\in[U]}(\boldsymbol{f}[i] - \boldsymbol{f}'[i]) = 1$, and m and m' be defined as in Table 1, then we have $m \geq m'$.*

We now analyze the exact bound of $m - m'$ in Lemma 2, which will be used in the proof of Theorem 3.

Lemma 2. *Let D and D' be two datasets with $\sum_{i\in[U]} \boldsymbol{f}[i] - \boldsymbol{f}'[i] = 1$, and m and m' be defined as in Table 1, then we have $0 \leq m - m' \leq 1$.*

The concrete proof of Lemma 2 is given in Appendix C.2.

Finally, we are ready to analyze the bound of $q_S - q'_S$ in Theorem 3.

Theorem 3. *Given two datasets D and D', we denote the size of non-empty nodes in S (resp. S') for D (resp. D') as q_S (resp. q'_S). If $\sum_{i\in[U]}(\boldsymbol{f}[i] - \boldsymbol{f}'[i]) = 1$, then $q_S - q'_S \leq 2$.*

The concrete proof of Theorem 3 is given in Appendix C.3.

By modifying the constraint condition on D and D' such that $\sum_{i\in[U]}(\boldsymbol{f}'[i] - \boldsymbol{f}[i]) = 1$ is satisfied and repeating the whole analysis, we could get Corollary 4.

Corollary 4. *Given two datasets D and D', we denote the size of non-empty nodes in S (resp. S') for D (resp. D') as q_S (resp. q'_S). If $\sum_{i\in[U]} |\boldsymbol{f}[i] - \boldsymbol{f}'[i]| = 1$, then $|q_S - q'_S| \leq 2$.*

Obviously, the nodes in Q but out of S should be the same as that in Q' but out of S'. therefore, we can finally get the global sensitivity $\Delta L = 2$ of function $L(\cdot)$ in Corollary 5.

Corollary 5. *Given two datasets D and D', we denote the size of Q-Digest Q (resp. Q') for D (resp. D') as q (resp. q'). If $\sum_{i\in[U]} \left|\boldsymbol{f}[i] - \boldsymbol{f}'[i]\right| = 1$, then $\left|q - q'\right| \leq 2$, i.e., the global sensitivity ΔL for function $L(\cdot)$ is 2.*

5 Our Protocol with Malicious Security

In the malicious setting, the adversary is one who corrupts parties and causes corrupted parties to deviate arbitrarily from the prescribed protocol. In the previous section, there is not any input validity check of the two-party computation. Therefore, there is no way to prevent the malicious participant from inputting any invalid Q-Digest, which is a behavior that deviates from the protocol where the Q-Digest is calculated from a valid frequency vector. By providing invalid inputs, the adversary would get some information about the honest party's input.

In this section, we explore how to augment the two-party computation to achieve malicious security with differentially private leakage of the input. Our main idea is to design an algorithm OCheck in Section 5.1 to obliviously check the validity of input before running the OMerge algorithm. In Section 5.2, we show the construction of the protocol Π_{mQDigest} which integrates the OCheck algorithm, and further prove that the protocol securely realizes $\mathcal{F}_{\mathsf{QDigest}}$ against malicious adversaries.

5.1 OCheck Algorithm

5.1.1 Algorithm Description

Input and Initialization. The input of OCheck algorithm is the same as that of OMerge algorithm.

Intuition. The goal of our OCheck algorithm is just checking the validity of input and outputting the check result but not updating any tuple. Intuitively speaking, if a single leaf node is valid, its counter value should be bigger than the threshold; if two leaf nodes with the same parent are valid, their sum of the counter values should be bigger than the threshold; if any inter-level node is valid, its counter value should be no bigger than the threshold since the counter value of the node is merged from lower-level nodes, and the sibling of the node should exist (which is shown by Fact 1 in Section 4.3). The high-level idea to do the check is first recovering the non-leaf nodes to their original places by calculating a new list H, taking the union of Q and H, and setting a check flag to 1, then repeating the checking process (following the principle we have

mentioned above) node by node from bottom to up hierarchically by scanning the sorted and united list Q. Once there is a check fails, flag is set to zero. When the whole check finishes, the algorithm outputs the flag as the check result, 1 for success and 0 for failure.

Specifically, for the validity check of the node with identifier id, we do the following operations. We first check if we have scanned all the nodes with the same parent, which is realized with the help of $\mathsf{Parent}(\cdot)$ as used in the OMerge algorithm. Then, for the leaf level, we check how many valid nodes belong to the same parent, here we define predicates 1ValidLeaf and 2ValidLeaf to do the check, only when the conditions are satisfied, the corresponding predicate will be set to 1; for the intermediate level, only one kind of situation is valid, we define a predicate 2ValidInternode to reflect it; for the root level, we define predicate ValidRoot. Any other kinds of situations make the expression in line 26 set to 0 mean that the validity check fails.

The complete description of OCheck algorithm is given in Appendix B.2.

5.1.2 Security Analysis

We now show the correctness and the obliviousness of OCheck. For correctness, we show that only the Q-Digest computed from a valid frequency vector can pass the check. We construct an algorithm that is used to recover a valid frequency vector from a given valid Q-Digest as shown in Algorithm 1.

Algorithm 1 takes the Q-Digest Q as input and pushes the non-leaf nodes to the leaf level one by one from top to bottom by looking for a leaf node such that there is no valid node on the path from the current node to the leaf node. The way to look for the node is based on the definition of ranges, which has the property that the parent's range equals the union of its two children's ranges. The root node has the maximum range of the universe $[U]$, the left child of the root has the range of $[1, U/2]$, the right child with $[U/2+1, U]$, and so on. For a certain node with range $[x, y]$, we know that it has covered all the range of its successors. By subtracting the ranges of its successors in Q from range $[x, y]$, we will find at least one valid range such that we can ensure that the node with $[x, y]$ is merged from the range or these ranges (which is how the Q-Digest algorithm works).

We now show that if Q is valid, i.e. Q is calculated from a valid frequency vector $\boldsymbol{f}$, then Algorithm 1 will always return one valid frequency vector $\tilde{\boldsymbol{f}}$ such that the Q-Digest of $\tilde{\boldsymbol{f}}$ is also Q. Since the Q-Digest algorithm is an approximation algorithm, the elements with high frequency will be kept in Q while the less frequently occurred elements will be grouped in larger ranges (i.e. at the upper level) such that the exact frequency information is lost. But on the other hand, the loss of frequency information for the less frequent elements also means that for a certain Q-Digest Q, there would be more than one valid frequency vector corresponding to it. Therefore, by arbitrarily pushing the upper ranges down to the leaf nodes following Algorithm 1 will recover a valid frequency vector for Q.

Algorithm 1 frequency vector reconstruction (Q)

Output: The recovered frequency vector $\boldsymbol{f}$.

Sort Q by id
for $i := 1$ to $len(Q)$ **do**
    $[x, y] = \mathsf{GetRange}(Q[i])$
    $W = \{\}$
    for $j := 1$ to $len(Q) - i$ **do**
        $[u, v] = \mathsf{GetRange}(Q[j])$
        $W = W \cup [u, v]$
    end for
    $W = [x, y] \setminus W$
    $Q[i].\mathsf{id}' = \mathsf{SetId}(W)$
end for

12: /* GetRange(·) would calculate the range of this node according to its position (i.e. its id) in the q-digest tree. */
13: /* SetId(·) would choose an identifier according to the given range, choose the leftmost one in the range if there are multiple valid ranges, choose the leftmost one. */

For the obliviousness, similar to the OMerge algorithm, all the access to Q is input independent, and the obliviousness of OCheck is obvious.

5.2 Maliciously Secure Two-Party Q-Digest Protocol

In this section, we show the construction of our protocol Π_{mQDigest} in the malicious setting.

5.2.1 Protocol Description

In brief, protocol Π_{mQDigest} as shown in Figure 6 adds a Q-Digest input check step based on Π_{sQDigest} to resist malicious adversaries.

Protocol Π_{mQDigest}

Parameters: $[U]$ is the universe, k is the compression parameter of Compress algorithm defined as Algorithm 3, $L^*(\cdot)$ is a leakage function.

Inputs: Party P_i for $i \in [2]$ inputs a frequency vector $\boldsymbol{f}_i$ with $n_i = \sum_{j\in[U]} \boldsymbol{f}_i[j]$.

The protocol:

1. Each party P_i locally computes the following:
   a. Generate a list Q_i based on $\boldsymbol{f}_i$, and compute $f(Q_i, n_i, k)$ to get the local Q-Digest Q_i.
   b. Execute leakage function $L^*(q_i)$ where $q_i := |Q_i|$ to get the noised output q_i'.
   c. Pad the Q-Digest Q_i to the length of q_i', update Q_i such that every item is a tuple $\langle \mathsf{id}, \mathsf{c}, \mathsf{isDummy}, \mathsf{isParent}\rangle$, set isDummy to 1 for all the dummy nodes.
2. Two parties run a secure computation which calculates OCheck algorithm and OMerge algorithm. If the output of OCheck is 1, then output the merged Q-Digest Q. Otherwise, output 0.

Figure 6: Maliciously secure two-party Q-Digest with leakage.

5.2.2 Security Analysis

Theorem 4. *If $L^*(\cdot)$ is (ϵ, δ)-differentially private function, then the protocol Π_{mQDigest} shown in Figure 6 securely realizes $\mathcal{F}_{\mathsf{QDigest}}$ with (ϵ, δ)-DP leakage against malicious adversaries in the $\mathcal{F}_{\mathsf{2PC}}$-hybrid model.*

Proof. Let the corrupted party is P_c and the honest party is P_h. We construct a simulator $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{QDigest}}$. First, once $\mathcal{S}$ receives a message $(\mathsf{received}, P_h)$ from the functionality, $\mathcal{S}$ could send $(\mathsf{leakageQuery}, P_h)$ to $\mathcal{F}_{\mathsf{QDigest}}$ and get the size of the honest party's Q-Digest with dummy nodes. $\mathcal{S}$ also simulates $\mathcal{F}_{\mathsf{2PC}}$ with the input such that the size is the same as the message returned by the leakage query. $\mathcal{S}$ also receives the Q-Digest with dummy nodes Q_c from corrupted party P_c by the simulation of $\mathcal{F}_{\mathsf{2PC}}$. If $\mathcal{F}_{\mathsf{2PC}}$ aborts, then $\mathcal{S}$ also aborts. Otherwise, it means that the input validity check of P_c passes. In this case, in order to query $\mathcal{F}_{\mathsf{QDigest}}$, $\mathcal{S}$ needs to extract P_c's frequency vector from Q_c. $\mathcal{S}$ runs the frequency vector

reconstruction algorithm as shown in Algorithm 1. Once a valid frequency vector $\boldsymbol{f}_c$ which coincides with Q_c is recovered, $\mathcal{S}$ invokes $\mathcal{F}_{\mathsf{QDigest}}$ with input $(\mathsf{input}, P_c, \boldsymbol{f}_c)$, and gets the merged Q-Digest Q. $\mathcal{S}$ returns Q to P_c in the simulation of the two-party computation functionality.

The same as in the semi-honest case, if the leakage function $L^*(\cdot)$ is (ϵ, δ)-differentially private, Π_{mQDigest} securely realize $\mathcal{F}_{\mathsf{QDigest}}$ in the two-party secure computation hybrid with (ϵ, δ)-DP leakage. □

6 Experiments

In this section, we first give some discussion on how we choose the privacy parameter ϵ and present the results of several experiments on the implementation of our protocols. Specifically, we evaluate the concrete performance of our protocol under different security guarantees, with different dataset sizes and universe sizes of the synthetic datasets we generate. Based on this, we also show the performance of our protocol when handling real datasets.

Privacy parameter selection. First, there are two kinds of commonly used mechanisms to achieve differential privacy, i.e. Laplace mechanism and truncated Laplace mechanism [41]. The analysis in [41] shows that the truncated Laplace mechanism is with better privacy properties. So we choose to use the truncated Laplace mechanism in our later experiments. In our evaluation, we focus on the case that $\delta = 2^{-40}$, which is the acceptable statistical security parameter in most applications. As for the private budget ϵ, we follow the analysis of Ninghui Li et al. [42] that the meaningful range in most applications is $0.01 \leq \epsilon \leq 10$. More concretely, $\epsilon = 0.1$ can usually offer reasonably strong privacy protection, and $\epsilon = 1$ may be acceptable in a lot of cases. Readers who are interested in the detailed analysis can refer to their original book. To get the expected noise to be added, we calculate the expectation for the truncated Laplace mechanism under different parameters and get Table 2.

Table 2: Expectation of Truncated Laplace mechanism [41] under different privacy parameters.

PDF	$p \cdot e^{(-\lvert x-\eta\rvert/b)}$					
Expectation	$E(x) = 2p\eta b$					
δ	2^{-40}					
ϵ	0.01	0.1	0.5	1	5	10
b	200	20	4	2	0.4	0.2
$E(x)$	5408.04	542.06	109.03	54.37	8.18	2.58

In this table, $b = \frac{\Delta L}{\epsilon}$, $p = \frac{e^{\epsilon/\Delta L}-1}{e^{\epsilon/\Delta L}+1}$, $\eta = -\frac{\Delta L \cdot \ln((e^{\epsilon/\Delta L}+1)\delta)}{\epsilon} + \Delta L$. Also note that, in our analysis of Section 4.3, $\Delta L = 2$.

Implementation and system. We implement OMerge and OCheck algorithms in C++ and run them as two-party protocol under the framework of EMP tools [43]. We also execute our experiment on an Amazon EC2 machine of type c5.2xlarge.

Evaluation of Performance. Following the way to handle generic secure computation protocols, the algorithms OMerge and OCheck are required to be converted into circuits where the circuit layout is independent of the input values. The evaluation of the performance of the two-party protocol depends critically on the size of the circuit. Through our modification, obliviousness is achieved by the OMerge algorithm and the OCheck algorithm. Besides, the circuit of the two-party protocol can be converted by EMP tools. The execution time of the two-party protocol is measured by us and it mainly depends on the circuit size, which is

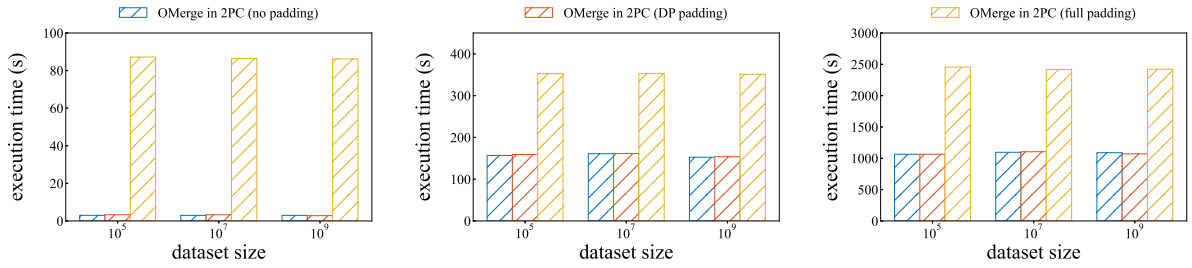
also recorded in our experiments.

Datasets. The synthetic datasets with a certain universe and size used in Section 6.1 are generated such that every value is sampled from a Gaussian distribution with mean 0 and variance 1, scaled to a certain range (for example, if the universe $U = 2^{32}$, the range to be scaled is $[0, 2^{32} - 1]$) and then rounded to an integer since q-digest cannot handle floating-point numbers. To demonstrate the practicality of our protocol, we also evaluate the performance of our protocol on two real datasets in Section 6.3, namely the TCP/UDP packets and web page click data.

6.1 Performances in the Semi-Honest Setting

All our experiments are carried out under the setting that $\delta = 2^{-40}$. In the experiments, there are two participants Alice and Bob. We consider different cases where Alice and Bob each hold a dataset with size $n \in \{10^5, 10^7, 10^9\}$ and universe $U \in \{2^8, 2^{16}, 2^{32}\}$, under the cases that $\epsilon \in \{0.01, 0.5, 1\}$. We also assume that the OMerge algorithm is executed with the error of 0.01, which is also the common acceptable error rate in most applications. Following Theorem 1 in [9], we then set the compression parameter k to be $\log U/0.01$.

Alice and Bob first calculate their local merge separately and then run our protocol under different privacy guarantees of the two participants' local merge. In the case of no padding, there is not any privacy protection of the local merge inputs, i.e., we do not add any noise to the inputs; for DP padding, there is a certain differential privacy guarantee of the local merge inputs, i.e., we add certain noise based on the definition of truncated Laplace mechanism; for full padding, there is full privacy guarantee of the inputs, i.e., we add noise to hide the local merge inputs such that the input length is the upper bound of the size according to the analysis of Lemma 1 in [9].



(a) universe is 2^8 (b) universe is 2^{16} (c) universe is 2^{32}

Figure 7: The performances for synthetic datasets in semi-honest setting. All time is reported in the LAN setting.

We get the execution time of protocol Π_{sQDigest} as shown in Figure 7 and also record the detailed circuit sizes for the synthetic datasets as shown in Table 3. We can see that the running time is linear with the circuit size and the circuit size is always the same under the cases of the same universe with full padding. This is because the previous paper of Q-Digest analyzed the upper bound of the data structure's size as $3k$, so the trivial way to hide the information that would be leaked by the size of the local merge result is to add dummy data until the theoretic maximum size. Our goal is to reduce the size of dummy data as much as possible with the help of differential privacy.

Our experiments also show that the privacy of two parties who execute the Q-Digest algorithm can be achieved at almost no cost since the execution time is almost the same as the case where no privacy is provided, and also insignificant compared to achieving input privacy protection in a trivial way. This can also be shown by the expectation of truncated Laplace mechanism as we have calculated in Table 2, where

the expectation is independent with compression parameter k. Therefore, the percentage of dummy data is decided only by the universe and the error setting of the algorithm. In the setting that error is 0.01, with universe grows, the compression parameter k and also the maximum local merge size would grow, while the expectation is kept the same, so the percentage of dummy data is decreased. under the same universe, when the error decreases, the compression parameter k and also the maximum local merge size would grow. Therefore, the percentage of dummy data will also decrease with the error decreases.

Table 3: The circuit size for synthetic datasets under different conditions.

U	N	circuit size				
		no padding	DP padding			full padding
			$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$	
2^8	10^5	11,683,312	112,647,328	28,812,592	19,619,672	560,018,336
	10^7	10,465,248	111,226,320	27,280,384	18,348,784	560,018,336
	10^9	11,097,024	111,955,400	28,059,504	19,077,864	560,018,336
2^{16}	10^5	936,658,560	1,199,216,208	1,093,988,856	961,336,440	2,341,210,752
	10^7	979,268,976	1,250,883,864	1,028,975,016	1,002,510,712	2,341,210,752
	10^9	980,395,776	1,252,526,080	1,030,682,280	1,003,702,560	2,341,210,752
2^{32}	10^5	7,615,775,336	8,597,491,448	7,821,858,200	7,714,318,400	17,007,918,336
	10^7	6,656,984,984	7,523,980,672	6,815,910,864	6,742,216,704	17,007,918,336
	10^9	6,409,268,992	7,183,261,080	6,527,893,312	6,461,997,392	17,007,918,336

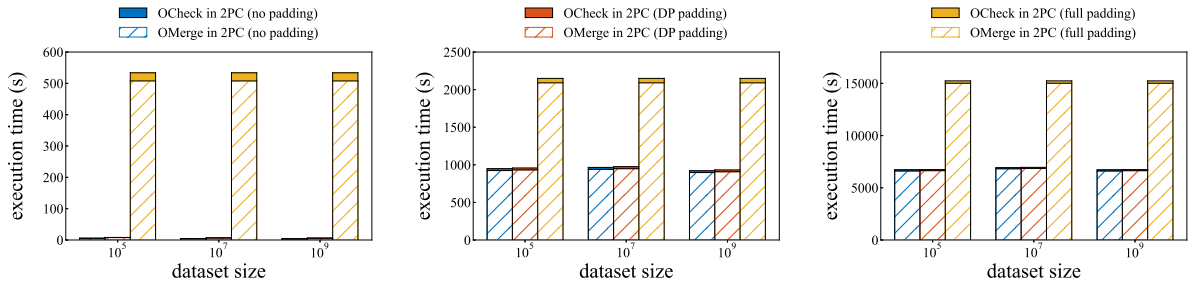
Concretely, the cases of inputs with DP padding is only 0.457-11.796 seconds slower compared with the cases that no privacy is provided, while it is $1.1\times$ to $73.1\times$ faster than the cases of full padding. It should be noted that the improvement is highly dependent on the input size, and with the universe changes, to guarantee the same error rate of the Q-Digest computation, the compression parameter is changed. Therefore, the input size of our protocol is not linear to the universe size.

6.2 Performances in the Malicious Case

We also evaluate the execution time of protocol Π_{mQDigest} under different cases and get Figure 8, which consists of two parts: the execution of OCheck in zero-knowledge proof and the execution of OMerge in two-party computation.

Concretely, the time for the execution of OCheck is measured according to the result of Yang *et al.* [44] that Boolean circuits can be proved with the speed of 7.7 million AND gates per second (with one thread); the time for the execution of OMerge is measured according to the result of [45] that per-authenticated AND gate can be generated with 0.239 μs (with eight threads).

The evaluation shows that our protocol can achieve security against malicious adversaries with the time used to do the input validity check almost negligible, compared with the time spent by the two-party com-



(a) universe is 2^8 (b) universe is 2^{16} (c) universe is 2^{32}

Figure 8: The performances for synthetic datasets in the malicious setting. All time is reported in the LAN setting.

putation of OMerge.

In detail, the cases of inputs with DP padding are only 2-32 seconds slower than the cases in that no privacy is provided, while it is 2.1× to 74.2× faster than the cases of full padding.

6.3 Performances on Real Dataset

First, we choose the daily network traffic dataset of WIDE project [46] and arbitrarily choose the collected data on Nov. 1st, 2020, which contains 88937609 packets in total. We extract the destination port information of the TCP/UDP packets from the dataset to do the further experiment. After the filtration and extraction, we get a dataset with size $n = 49,282,099$. Since the maximum port number is 65535, the universe size $U = 2^{16}$. Besides, we also conduct our experiments on the Kosarak[4] data set. It consists of 990,000 sequences of click-stream data over 41,270 unique web page ids, a total of 8,019,015 counters. Therefore, the size of the dataset is $n = 8,019,015$. We randomly split them into two parts and compute the click frequency of each page and do our further experiment.

Here we only consider the situation $\epsilon = 1/300$ to show a more extreme case that the accuracy requirements are very high. We want to show that the performance of our protocol is still acceptable in such a situation. The experiment results are given in Table 4.

Table 4: The performances of the real dataset.

setting	TCP/UDP packets		Kosarak data	
	circuit size	execution time (s)	circuit size	execution time (s)
no padding	1,255,561,824	45.664	1,650,536,968	24.852
DP padding	1,287,095,184	46.846	1,606,637,472	25.747
full padding	8,746,648,704	318.295	8,746,648,704	317.014

The compression parameter is set such that the error rate is $1/300$.

[4] `http://fimi.ua.ac.b/e/data/`http://fimi.ua.ac.be/data/.

The scale of the real data is comparable to the experimental data with a dataset of 10^7 and a universe of 2^{16} in Section 6.1. The execution time is faster because the input data is smaller, which can be reflected in the circuit size and also shows that the real data set does not obey Gaussian distribution.

6.4 Comparison with Direct Approach

As there is no work on secure quantile aggregation in the distributed setting. The only comparison we can make is with the direct way that computes quantile from the beginning by two parties under some framework that can program the original quantile algorithm in a secure way. By saying secure, we mean that the whole computation procedure is obliviousness, so as to guarantee that no private information is leaked. We follow the result of GraphSC [47], where the main overhead for any kind of secure computation is sorting, and compare the circuit size when handling the data with the same magnitude. The detailed comparison results are shown in Figure 9. As the execution time is proportional to the circuit size, it gives us confidence, especially in big data scenarios.

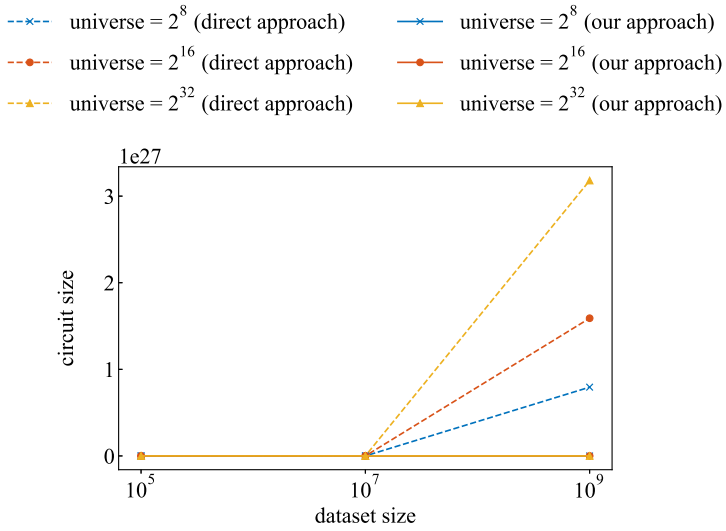


Figure 9: Comparison with direct approach.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very valuable comments. Xiao Lan and Hongjian Jin are supported by the National Key Research and Development Program of China (Grant Nos. 2022YFB3102500), the National Natural Science Foundation of China (Grant Nos. 61802270, U19A2081). Hui Guo is supported by the National Key Research and Development Program of China (Grant Nos. 2022YFB2702000), the National Natural Science Foundation of China (Grant Nos. 62022018, 61932019, and 61802021).

References

[1] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. Annu. ACM Symp. Theory Comput. (STOC)*, 1996, pp. 20–29.

[2] R. Morris, "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[3] J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.

[4] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita, "Improved histograms for selectivity estimation of range predicates," in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, 1996, pp. 294–305.

[5] M. Charikar, K. C. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. Int. Colloq. Autom., Lang. Program. (ICALP)*, 2002, pp. 693–703.

[6] M. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, 2004, pp. 275–285.

[7] G. Cormode and K. Yi, *Small summaries for big data*. Cambridge University Press, 2020.

[8] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proc. ACM Conf. Appl., Technol., Archit., Protoc. Comput. Commun. (SIGCOMM)*, 2003, p. 99–110.

[9] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: New aggregation techniques for sensor networks," in *Proc. ACM Int. Conf. Embed. Networked Sens. Syst. (SenSys)*, 2004, pp. 239–249.

[10] Y. Tao, K. Yi, C. Sheng, J. Pei, and F. Li, "Logging every footstep: quantile summaries for the entire history," in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, 2010, pp. 639–650.

[11] T.-H. H. Chan, M. Li, E. Shi, and W. Xu, "Differentially private continual monitoring of heavy hitters from distributed streams," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp. (PETS)*, 2012, pp. 140–159.

[12] 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 *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 192–203.

[13] M. Aumüller, C. J. Lebeda, and R. Pagh, "Differentially private sparse vectors with low error, optimal space, and fast access," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2021, pp. 1223–1236.

[14] T. Wang, N. Li, and S. Jha, "Locally differentially private heavy hitter identification," *IEEE Trans. Dependable Secur. Comput. (TDSC)*, vol. 18, no. 2, pp. 982–993, 2021.

[15] M. Zhou, T. Wang, T. H. Chan, G. Fanti, and E. Shi, "Locally differentially private sparse vector aggregation," in *Proc. IEEE Symp. Secur. Priv. (S&P)*, 2022, pp. 422–439.

[16] Z. Huang, Y. Qiu, K. Yi, and G. Cormode, "Frequency estimation under multiparty differential privacy: one-shot and streaming," *Proc. VLDB Endow.*, vol. 15, no. 10, pp. 2058–2070, 2022.

[17] D. Alabi, O. Ben-Eliezer, and A. Chaturvedi, "Bounded space differentially private quantiles," *CoRR*, 2022.

[18] C. J. Lebeda and J. Tetek, "Better differentially private approximate histograms and heavy hitters using the Misra-Gries sketch," *CoRR*, 2023.

[19] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient private statistics with succinct sketches," in *Network and Distributed System Secur. Symp.*, 2016.

[20] X. Liu, R. H. Deng, K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 11, pp. 2401–2414, 2016.

[21] M. Naor, B. Pinkas, and E. Ronen, "How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2019, pp. 1369–1386.

[22] X. Liu, R. H. Deng, K. R. Choo, Y. Yang, and H. Pang, "Privacy-preserving outsourced calculation toolkit in the cloud," *IEEE Trans. Dependable Secur. Comput.*, vol. 17, no. 5, pp. 898–911, 2020.

[23] S. G. Choi, D. Dachman-Soled, M. Kulkarni, and A. Yerukhimovich, "Differentially-private multi-party sketching for large-scale statistics," Cryptology ePrint Archive, Report 2020/029, 2020, https://eprint.iacr.org/2020/029.

[24] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2021, pp. 762–776.

[25] J. Böhler and F. Kerschbaum, "Secure multi-party computation of differentially private heavy hitters," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2021, pp. 2361–2377.

[26] A. Aseeri and R. Zhang, "SecQSA: secure sampling-based quantile summary aggregation in wireless sensor networks," in *Proc. IEEE Int. Conf. Mobil., Sens. Netw.(MSN)*, 2021, pp. 454–461.

[27] J. Bell, A. Gascón, B. Ghazi, R. Kumar, P. Manurangsi, M. Raykova, and P. Schoppmann, "Distributed, private, sparse histograms in the two-server model," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2022, pp. 307–321.

[28] P. Jangir, N. Koti, V. B. Kukkala, A. Patra, B. R. Gopal, and S. Sangal, "Poster: Vogue: Faster computation of private heavy hitters," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2022, pp. 3371–3373.

[29] C. Dwork and A. Roth, *The algorithmic foundations of differential privacy*. Now Publishers Inc., 2014.

[30] L. Wang, G. Luo, K. Yi, and G. Cormode, "Quantiles over data streams: an experimental study," in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, 2013, pp. 737–748.

[31] Z. Chen and A. Zhang, "A survey of approximate quantile computation on large-scale data," *IEEE Access*, vol. 8, pp. 34 585–34 597, 2020.

[32] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 830–842.

[33] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang, "Optimizing authenticated garbling for faster secure two-party computation," in *Advances in Cryptology (CRYPTO)*, 2018, pp. 365–391.

[34] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2020, pp. 1607–1626.

[35] M. Keller and P. Scholl, "Efficient, oblivious data structures for MPC," in *Advances in Cryptology (ASIACRYPT)*, 2014, pp. 506–525.

[36] M. Keller, "The oblivious machine - or: How to put the C into MPC," in *Int. Conf. Cryptology and Infor. Secur. Latin America*, 2017, pp. 271–288.

[37] M. Keller and A. Yanai, "Efficient maliciously secure multiparty computation for RAM," in *Annu. Int. Conf. Theory Appl. Cryptographic Tech. (EUROCRYPT)*, 2018, pp. 91–124.

[38] Y. Lindell, *How to simulate it - a tutorial on the simulation proof technique*. Springer International Publishing, 2017.

[39] D. Evans, V. Kolesnikov, and M. Rosulek, *A pragmatic introduction to secure multi-party computation*. NOW Publishers, 2018.

[40] A. Groce, P. Rindal, and M. Rosulek, "Cheaper private set intersection via differentially private leakage," *Proc. Int. Symp. Privacy Enhancing Technol. Symp. (PETS)*, vol. 2019, no. 3, pp. 6–25, Jul. 2019.

[41] X. He, A. Machanavajjhala, C. J. Flynn, and D. Srivastava, "Composing differential privacy and secure computation: A case study on scaling private record linkage," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1389–1406.

[42] N. Li, M. Lyu, D. Su, and W. Yang, *Differential privacy: from theory to practice*, ser. Synthesis Lectures on Information Security, Privacy, & Trust. Morgan & Claypool Publishers, 2016.

[43] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: efficient multiparty computation toolkit," https://github.com/emp-toolkit, 2016.

[44] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," Cryptology ePrint Archive, Report 2021/076, 2021, https://eprint.iacr.org/2021/076.

[45] K. Yang, X. Wang, and J. Zhang, "More efficient MPC from improved triple generation and authenticated garbling," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2020, pp. 1627–1646.

[46] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the WIDE project," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2000, pp. 263–270.

[47] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, "Graphsc: Parallel secure computation made easy," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, 2015, pp. 377–394.

A Q-Digest algorithm

In Q-Digest, the range associated with every node is defined by the position of the node in the tree. To be specific, the root has a range $[1, U]$, and given any non-leaf node with range $[a, b]$, its left child and right child are associated with ranges $[a, a + \frac{b-a+1}{2} - 1]$ and $[a + \frac{b-a+1}{2}, b]$, respectively. With such a definition, the range associated with the i-th leaf contains only i. Besides, the identifier of the node is defined in a way that the root is identified by 1 and given any non-leaf node identified by i, its left child and right child are identified by $2i$ and $2i + 1$, respectively. Also, we call the node an empty node when its counter value is zero, otherwise, we call the node a non-empty node.

The compression procedure on a single node is described as Algorithm 2, which determines whether a node should be compressed to its parent by checking the compression condition as introduced in Section 2.2. However, there is an exception to the root node. Since the compression procedure is bottom-up, if the root

Algorithm 2 CompressNode $(\mathsf{id}, \mathsf{c}, \mathsf{c_p}, \mathsf{c_s}, \theta)$

if $0 < \mathsf{c} + \mathsf{c_p} + \mathsf{c_s} \leq \theta$ **then**
  $\mathsf{c_p} = \mathsf{c_p} + \mathsf{c} + \mathsf{c_s}$
  $\mathsf{c} = 0$
  $\mathsf{c_s} = 0$
end if

Algorithm 3 Compress (Q, n, k)

$l = \log U - 1, \theta = \lfloor n/k \rfloor$
while $l > 0$ **do**
  for all id in level l **do**
    if $\mathsf{c} \neq 0$ or $\mathsf{c_s} \neq 0$ **then**
      CompressNode$(\mathsf{id}, \mathsf{c}, \mathsf{c_p}, \mathsf{c_s}, \theta)$
    end if
    Delete empty nodes
  end for
  $l = l - 1$
end while

node is non-empty, its counter value must be no bigger than θ. But there is no parent or sibling for the root, so the root node will not be compressed even though the compression condition is met.

Based on Algorithm 2, the Compress algorithm (Algorithm 3) used to calculate the final Q-Digest can be constructed. In the beginning, the input to the Compress algorithm is the set Q, which contains all the nodes in the tree after initialization. After the execution of Compress algorithm, a node with $\langle \mathsf{id}, \mathsf{c} \rangle$ still remains in Q if and only if the condition in Algorithm 2 is not met; otherwise, a compression procedure will be called such that the counters of the node and its sibling will be compressed to their parent and the two nodes will be deleted from Q. Therefore, all the nodes that remain in Q are non-empty nodes since all the empty nodes have been deleted from Q in Algorithm 3.

Algorithm 4 Merge $(Q_1(n_1, k), Q_2(n_2, k))$

$Q \leftarrow Q_1 \cup Q_2$
Compress $(Q, n_1 + n_2, k)$

Besides, multiple Q-Digests can also be merged to get a new Q-Digest by running the Merge algorithm. The merging of two Q-Digests is shown as Algorithm 4, where the two Q-Digests should have the same compression parameter k and the same universe (i.e., the same q-digest depth d). There needs an involved party or a trusted party to execute the Merge algorithm to get a merged Q-Digest.

B Oblivious Algorithms

B.1 OMerge Algorithm

Algorithm 5 OMerge $(Q_1(n_1, q_1, k, d), Q_2(n_2, q_2, k, d))$

$\theta = \lfloor (n_1 + n_2)/k \rfloor$
$Q = Q_1 \cup Q_2$
$len(Q) = q_1 + q_2$
Sort Q by $(-\mathsf{id}, \mathsf{isParent})$[5]
/ merge two input to a single list */*

[5]It means that the items in set Q are sorted first in descending order of id first, and when id is the same, the items are sorted in ascending order of isParent.

for $i := 1$ to $len(Q) - 1$ **do**
  if $\neg Q[i]$.isDummy and $\neg Q[i+1]$.isDummy and $Q[i]$.id $==$ $Q[i+1]$.id and $Q[i]$.isParent $==$ $Q[i+1]$.isParent **then**
    $Q[i]$.c $= Q[i]$.c $+ Q[i+1]$.c
    $Q[i+1]$.isDummy $= 1$
  end if
end for
$Q = Q \cup \{\langle 0, 0, 0, 0\rangle\}$
/ start merge q-digest */*
for $j := d$ to 1 **do**
  for $i := 1$ to $len(Q) - 1$ **do**
    if $2^j \leq Q[i]$.id $< 2^{j+1}$ and $\neg Q[i]$.isDummy and the parent of $Q[i+1]$ is not the same as $Q[i]$'s **then**
      if there is no node which has the same parent as $Q[i]$ **then**
        if $Q[i]$.c $\leq \theta$ **then**
          Set $Q[i]$.id to be its parent's id, and reset isParent to 0.
        end if
      else if there is a node $Q[i-1]$ which has the parent as $Q[i]$ **then**
        if $Q[i]$.c $+ Q[i-1]$.c $\leq \theta$ **then**
          Merge two nodes by setting $Q[i]$.id to be its parent's id and $Q[i]$.c to $Q[i]$.c $+ Q[i-1]$.c, reset $Q[i]$.isParent to 0, set $Q[i-1]$.isDummy to 1.
        else
          if $Q[i]$.isParent is 1 **then**
            Set $Q[i]$.id to be its parent's id, reset $Q[i]$.isParent to 0.
          end if
        end if
      else
        if $Q[i]$.c $+ Q[i-1]$.c $+ Q[i-2]$.c $\leq \theta$ **then**
          Merge three nodes by setting $Q[i]$.id to be its parent's id and $Q[i]$.c to $Q[i]$.c $+ Q[i-1]$.c $+ Q[i-2]$.c, reset $Q[i]$.isParent to 0, set $Q[i-1]$.isDummy and $Q[i-2]$.isDummy to 1.
        else
          Set $Q[i]$.id to be its parent's id, reset $Q[i]$.isParent to 0.
        end if
      end if
    end if
  end for
  / resort and update Q for the next level */*
  Sort Q by (isDummy, $-$id, isParent)
end for

Output: List Q which is the merge of Q_1 and Q_2.

B.2 OCheck Algorithm

Algorithm 6 OCheck(Q, n, q, k, d)

if $q > 3k$ **then**
  abort
end if
$\theta = \lfloor n/k \rfloor$
/ recover the inter-level nodes */*
$H = \{\}$
for $i := 1$ to q **do**
  if $Q[i]$.isParent **then**
    $H[i] = \langle \text{Parent}(Q[i].\text{id}), Q[i].\text{c}, 0, 0\rangle$
  else
    $H[i] = \langle 0, 0, 0, 0\rangle$
  end if
end for
$Q = Q \cup H$
Sort Q by (id, $-$isParent)
$Q = Q \cup \{\langle 0, 0, 0, 0\rangle\}$

17: $len(Q) = |Q|$
18: */* start to check */*
19: flag $= 1$
20: **for** $i := 1$ to $len(Q) - 1$ **do**
21: **if** $\neg Q[i]$.isParent and the parent of $Q[i+1]$ is not the same as $Q[i]$'s **then**
22: 1ValidLeaf = $Q[i]$.id $\geq 2^d$ and $Q[i]$.c $> \theta$ and there is not a node $Q[i-1]$ which has the same parent as $Q[i]$
23: 2ValidLeaf = $Q[i]$.id $\geq 2^d$ and $Q[i-1]$.id $\geq 2^d$ and $Q[i]$.c $+ Q[i-1]$.c $> \theta$ and there is only a node $Q[i-1]$ which has the same parent as $Q[i]$ and $\neg Q[i-1]$.isParent
24: 2ValidInternode = $Q[i]$.id $< 2^d$ and $Q[i-1]$.id $< 2^d$ and $Q[i]$.c $\leq \theta$ and $Q[i-1]$.c $\leq \theta$ and $Q[i]$.c $+ Q[i-1]$.c $> \theta$ and there is only a node $Q[i-1]$ which has the same parent as $Q[i]$ and $\neg Q[i-1]$.isParent
25: ValidRoot = $Q[i]$.id $== 1$ and $Q[i]$.c $\leq \theta$ and there is not a node $Q[i-1]$ which has the same parent as $Q[i]$
26: **if** $\neg$(1ValidLeaf or 2ValidLeaf or 2ValidInternode or ValidRoot) **then**
27: flag $= 0$
28: **end if**
29: **end if**
30: **end for**

Output: The check result flag, 1 means check passes, otherwise, check fails.

C Proofs

C.1 The proof of Lemma 1

Proof. We prove by contradiction and assume that all nodes located between level l_i and l_j in S' are all empty after the compression procedure finishes. We first look at the compression procedure of T from level l_i to level l_j. Since there are nodes at level l_i in Q, the parent of them must be not in Q. Besides, all the nodes at level l_x are empty where $l_i > l_x > l_j$ while the nodes at level l_j are in Q. It means that the compression condition is met by all the nodes at level l_x, and all the counter values are summed up to the counter value of the node at level l_j in the red path. Now we look at the compression procedure of T' from level l_i to level l_j. Since we assume that the nodes located at level l_y in S' are empty, where $l_i \geq l_y \geq l_j$. It means that the compression procedures from at least level l_i to level l_j have not reached the threshold yet. Therefore, the counter value of the node at level l_j in the red path is at least the sum of counter values of the siblings for the nodes in the red path from l_i to l_j. That is to say, there is at least one more level being counted in T' than in T for the node in the red path at level l_j. Besides, from Fact 2 we know that the nodes outsides S and S' will always be the same in two trees every time the compression procedure on the nodes of the same level finishes. Therefore, the counter value of the sibling of σ's ancestor at level l_j must be the same in two trees after the compression procedure of this level finishes. Hence, the sum of the counter values for nodes at level l_j in T' should be bigger than that of nodes at the same level in T because the non-zero counter value of the node at level l_i has also been counted in T'. Since the nodes at level l_j are in Q while the nodes at the same level are not in Q', it means that before the compression procedure of level l_j, the compression condition is violated by at least one time in Q'. Therefore, there must be a node in S' at level l_e which is between l_i and l_j, i.e. $l_i > l_e > l_j$. This causes a contradiction with our assumption. Additionally, we know that there are two nodes at level l_e from Fact 1.

□

C.2 The proof of Lemma 2

Proof. According to Corollary 3, we already know that $m - m' \geq 0$. Therefore, in the following analysis, we only prove that $m - m' \leq 1$. We prove by contradiction and assume that $m - m' > 1$. Since there is $\boldsymbol{f}[\sigma] - \boldsymbol{f}'[\sigma] = 1$, we have three cases to consider.

Case 1. the leaf node in both S and S' violates the compression condition. In this case, the leaf nodes in both S and S' remain. Therefore, m and m' are added by 1 at the same time. Besides, all the nodes

Table 5: The values of $q_S - q'_S$ in three cases under different conditions.

		Case 1	$q_S - q'_S$	Case 2	$q_S - q'_S$	Case 3	$q_S - q'_S$
$m - m' = 0$	$r - r' = 0$	$s - s' = 0$	0	$s = 1$	-1	$s = s' = 0$	0
		$s - s' = 1$	1	$s = 2$	0		
	$r - r' = 1$	$s - s' = 0$	1	$s = 1$	0		1
		$s - s' = 1$	2	$s = 2$	1		
$m - m' = 1$	$r - r' = 0$	$s - s' = 0$	2	$s = 1$	1		2
		$s - s' = 1$	**3**	$s = 2$	2		
	$r - r' = 1$	$s - s' = 0$	**3**	$s = 1$	2		**3**
		$s - s' = 1$	**4**	$s = 2$	**3**		

The compression parameter is set such that the error rate is the same as the setting in Section 6.1. In all three cases, from Lemma 2, we need to consider the case that $m - m' = 0$ and $m - m' = 1$; in each case, we do not know whether the root node is in Q (and Q') or not, so we also need to consider them separately. To be more specific, $r - r' = 0$ means that the root node is in or not in Q and Q' simultaneously, and $r - r' = 1$ means that the root node is in Q but not in Q'. However, the relationship of s and s' in the three cases is a little complex, which is caused by the different expression of $q_S - q'_S$ in Equation 3. In Case 1, the leaf nodes in S and S' will be kept in Q and Q', and the number of non-empty leaf nodes in S could be equal to that in S' (where $\boldsymbol{f}'[\sigma] > 0$), or bigger than that in S' by 1 (where $\boldsymbol{f}'[\sigma] = 0$), so we need to consider the cases of $s - s' = 0$ and $s - s' = 1$. In Case 2, the leaf nodes in S and S' will be kept in Q but not in Q'. So the non-empty leaf node number in S will be considered. Since $\boldsymbol{f}[\sigma] - \boldsymbol{f}'[\sigma] = 1$, there are at least one leaf nodes in S, therefore, we need to consider the case of $s = 1$ and $s = 2$. In Case 3, from Equation 3, we do not need to consider s and s', while in fact, in this case, both the leaf nodes in S and S' are empty, so we always have $s = s' = 0$.

outside S and S' are the same in the two trees and they will be updated in the same way. Therefore, once the compression procedure finishes at the leaf level, all the nodes on the next level (i.e. the level where the leaf nodes' parents reside in) are the same in the two trees. Later on, all the compression procedures for the upper levels are the same in two trees, so there should be $m = m'$.

Case 2. the leaf node in S violates the compression condition while that in S' does not. In this case, the leaf nodes in S remain while the leaf nodes in S' do not. Therefore, m is added by 1 while m' is not. If in the end, we have $m - m' > 1$, it must mean that in S, there exist two levels of non-empty nodes which are in Q while there is not any non-empty node of S' between the two levels that belong to Q', this contradicts with Lemma 1. Therefore, we have $m - m' \leq 1$.

Case 3. the leaf node in both S and S' meets the compression condition. In this case, the leaf nodes in both S and S' are compressed to their parent and will not be in Q and Q'. Also due to the fact that the merged counter value for the parent of these leaf nodes in T is bigger than that in T' by 1 (due to $\boldsymbol{f}[\sigma] - \boldsymbol{f}'[\sigma] = 1$), according to Corollary 2, the first time for the nodes to violate the compression condition must happen in T, this case is the same as Case 2. Therefore, we still have $m - m' \leq 1$.

To sum up, we proved that $0 \leq m - m' \leq 1$. □

C.3 The proof of Theorem 3

Proof. Following the analysis of Lemma 2, we need to consider the same three cases and the expressions of $q_S - q'_S$ in corresponding cases are shown in Equation 3, which comes from Fact 1.

$$q_S - q'_S = \begin{cases} 2(m - m') + (r - r') + (s - s'), & \text{Case 1} \\ 2(m - m') + (r - r') + (s - 2), & \text{Case 2} \\ 2(m - m') + (r - r'), & \text{Case 3} \end{cases} \tag{3}$$

We could analyze the bound of $q_S - q'_S$ case by case and get Table 5 with the exact value of $q_S - q'_S$ under certain conditions. But it is easy to see that no matter how to set these parameters, $q_S - q'_S$ is always bigger than minus 1. To simplify our analysis, we only show that the following subcases that cause $q_S - q'_S > 2$ are impossible (which have been emphasized in boldface in Table 5).

– (SC1). In Case 1, when $m - m' = 1$, $r - r' = 0$, and $s - s' = 1$; or $m - m' = 1$, $r - r' = 1$, and $s - s' = 0$; or $m - m' = 1$, $r - r' = 1$, and $s - s' = 1$.
– (SC2). In Case 2, when $m - m' = 1$, $r - r' = 1$, and $s = 2$.
– (SC3). In Case 3, when $m - m' = 1$ and $r - r' = 1$.

For case SC1, the leaf nodes of S and S' are in Q and Q' because they violate the compression condition. According to our analysis of Case 1 in Lemma 2, there is $m = m'$, so all the subcases in SC1 cannot happen.

For case SC2, since $s = 2$, there are two leaf nodes in S, and the two nodes are also in Q because in Case 2 they violate the compression condition; since $r - r' = 1$, the root node is in Q ($r = 1$) but not in Q' ($r' = 0$). Now we introduce a virtual extension method to the trees T and T' and use the extended trees T_{ex} and T'_{ex} to analyze the subcases. The idea of virtual extension is to extend the trees T and T' from their root as if we are expanding the universe of the dataset from $[U]$ to $[U + E]$ such that all the frequencies of elements in the extended range are the same in both trees. Through virtual extension, we would guarantee that the next time the node violates the compression condition must happen in T_{ex} since the root node is with a non-zero counter value in T but with zero-counter value in T', and all other nodes outside T and T' are the same provided the compression procedure finishes on the same level in two extended trees, which is also shown by Corollary 2. From Lemma 1, we know that the nodes in two extended trees would alternatively violate the compression condition according to their levels. So the last time for the same thing to happen before the virtual extension must be in T'. Therefore, we should have $m = m'$ and the subcase SC2 cannot happen.

For case SC3, the leaf node(s) of S and S' are not in Q and Q', and the root node is in Q ($r = 1$) but not in Q' ($r' = 0$). By using the same virtual extension technique, similar to in case SC2, we can also infer that the last time for the node to violate the compression condition should happen in T'. Therefore, we should have $m = m'$ and the case cannot happen too.

In conclusion, we have $q_S - q'_S \leq 2$. □