Towards Vertically Distributed Differentially Private Synthetic Data Generation

Yucheng Fu, Tiaoyao Gu†, Elaine Shi†, Tianhao Wang
University of Virginia, †Carnegie Mellon University

Abstract

Ensuring individuals' privacy while allowing for collaborative data sharing is crucial for organizations. One approach to this is synthetic data generation, which generates synthetic data that fits the statistical properties of private data. Although many techniques have been developed under differential privacy (DP), most of them are based on the assumption that the data is centralized. In practice, data is often distributed among multiple clients in a distributed setup. To achieve distributed DP data synthesis, we designed a secure multi-party computation (MPC) protocol. We identify that estimating the joint distribution of distributed datasets is a key efficiency bottleneck. To overcome this, our MPC protocol takes advantage of information already leaked by DP to achieve a linear communication complexity in the number of records and constant rounds of communication. Then, we extend this protocol to an end-to-end framework that supports the classical "Select-Measure-Generate" paradigm for DP data synthesis. We instantiate the distributed version of two state-of-the-art central methods, PrivSyn and AIM, to demonstrate the flexibility of our framework. Our experiments also demonstrate that our solution achieves up to four orders of magnitude better efficiency than the existing distributed method.

1 Introduction

Differentially private (DP) synthetic data generation (SDG) aims to create a realistic artificial dataset that shares a similar statistical property as the original dataset. Such datasets can be used to support arbitrary downstream data analysis tasks without additional privacy loss since they are post-processing. Almost all DP SDG methods focus on the centralized setting [14, 15, 21, 22], where each data owner can only generate synthetic datasets on their own. But in practice, data is often distributed among multiple clients in a distributed setup. For example, a patient's blood pressure and their economical situation may be held by two different organizations (hospitals and banks), and requiring the two organizations to separately generate datasets would result in the correlation between the two metrics being completely lost. In this work, we focus on this kind of vertical setting where different clients hold different attributes of the same individuals.

Several methods have been proposed to address this issue for the distributed DP SDG problem [13, 17, 18]. One branch of these methods follows the idea of federated learning that modifies the local generation and aggregation rules to overcome heterogeneity but this results in reduced utility of generated datasets [13]. Another branch is MPC-based, mirroring the entire generation process in the centralized setting with MPC protocol [17, 18], but they suffer from efficiency bottlenecks of computing two-way marginals, i.e., the joint distribution under two different attributes. The protocol to compute a two-way marginal has a communication complexity of $O(nu)$, where n is the number of records, and u is the domain size of this two-way marginal. We follow the MPC line of work and propose a significantly more efficient method that reduces the communication complexity to $O(n)$.

Specifically, we first make the observation that to compute a two-way marginal (the efficiency bottleneck), we can leverage the sort-and-count strategy to first sort the two columns to group the same value combination together and then count the frequencies with a single pass. This can reduce the communication to $O(n \log u)$. Next, we observe that the information leakage in the one-way marginals allows us to reveal the shuffled columns to servers without any privacy loss. This observation further improves the efficiency of secure sorting. Finally, the complexity of the communication is reduced to $O(n)$. Empirically, on the real-world dataset UCI Adult [3], we can estimate a two-way marginal within 5 seconds and about 59 MB of communication while the existing method CaPS [17] takes 80 seconds and 14,082 MB of

communication. As for end-to-end performance, we can generate synthetic datasets with 91 two-way marginals in about 11 minutes.

2 Preliminaries

We consider a dataset D with n rows and m columns. Each row i in D represents a user indexed by a unique identifier id_i ($1 \le i \le n$). Each column j in D represents an attribute $\vec{\mathsf{Att}}_j$ ($1 \le j \le m$). The domain of attribute $\vec{\mathsf{Att}}_j$ is denoted as $\mathcal{U}_i$, with size $|\mathcal{U}_i| = u_i$ (wlog, each domain $\mathcal{U}_i$ has been encoded as integers $\{1, \ldots, u_i\}$).

Marginal-based Data Synthesis. The synthesis of tabular data relies on a set of *marginals* with DP guarantee to guide the generation of a synthetic dataset. A marginal is a projection of D into low-dimensional synopses. A one-way marginal $\vec{\mathsf{M}}_a$ is the histogram over the domain $\mathcal{U}_a$ of attribute $\vec{\mathsf{Att}}_a$ in column a. A two-way marginal $\vec{\mathsf{M}}_{a,b}$ is the histogram over all the possible value combinations under the joint domain $\mathcal{U}_a \times \mathcal{U}_b$ of attributes $\vec{\mathsf{Att}}_a$ and $\vec{\mathsf{Att}}_b$. Typically, one-way and two-way marginals are sufficient to depict the distribution of the original dataset [14, 15, 22].

Select-Measure-Generate Workflow. Recent methods [14, 15, 22] for private tabular data synthesis follow a three-step "Select-Measure-Generate" workflow to achieve a good balance between privacy and utility. **(1) Select.** Due to DP constraints, we focus on marginals that give the most information. Typically, this involves selection from all one-way marginals $\vec{\mathsf{M}}_a$ and two-way marginals $\vec{\mathsf{M}}_{a,b}$ for each pair of columns $a \neq b$. A *score* is defined to quantify how much information each marginal can contribute. **(2) Measure.** Apply the Gaussian mechanism to measure each selected marginal. Since marginals are histograms, adding DP noise to it is 'DP-friendly'. **(3) Generate.** Run an algorithm to generate a synthetic dataset that follows the distribution of the measured marginals, which is a post-processing.

In our task, we assume the vertical distributed setting, where each of the m clients $\{C_1, ..., C_m\}$ only holds a single column corresponding to an attribute, denoted as $\vec{\mathsf{Att}}_i$. The clients hope to generate a synthetic table D' that preserves the statistical distribution of D while ensuring that their respective private columns remain private. Note that the domain of each column is assumed to be public (e.g., the possible genders and income ranges), whereas only the values within each column are viewed as private. We assume three non-colluded semi-honest servers P_1, P_2, P_3 to collect the secret shares of columns provided by clients. The servers then run the MPC protocols that securely compute the process of **Select** and **Measure** and output the selected DP marginals.

3 Our Approach

We start with a protocol to estimate the secret shares of two-way marginals. Then, we extend this core protocol to support two SDG methods in the distributed setting to demonstrate its scalability in the "Select-Measure-Generate" workflow. In the following part, we denote $\langle x \rangle$ as the secret shares of variable x, and $\langle \vec{X} \rangle$ as a vector containing secret-shared variables. All the operators applied to these variables refer to an MPC sub-protocol, achieving the same functionality as these operators. We describe the MPC primitives used in this paper in Appendix A.

3.1 Efficient Marginal Estimation

Existing work [17] computes each two-way marginal $\vec{\mathsf{M}}_{a,b}$ by traversing two columns $\langle \vec{\mathsf{Att}}_a \rangle$ and $\langle \vec{\mathsf{Att}}_b \rangle$. For each record, it enumerates all the entries in the two-way marginal $\langle \vec{\mathsf{M}}_{a,b} \rangle$ to update the frequencies. Assuming the size of cross-domain $\mathcal{U}_a \times \mathcal{U}_b$ is $u_{a,b}$, this MPC protocol has a communication complexity of $O(nu_{a,b})$.

Estimation with Sort-and-count Strategy. To avoid heavy linear-scan on $\vec{\mathsf{M}}_{a,b}$, we can leverage the protocol for frequency estimation in [2], namely, the Sort-and-count protocol. The main observation is that if a vector containing n elements is already sorted, then the same elements are grouped together, and we can count the frequency of each value with only $O(n)$ secure operations. In our task, the key for sorting is set as $\vec{K}_{a,b}[i] = \vec{\mathsf{Att}}_b[i] \cdot u_a + \vec{\mathsf{Att}}_a[i]$, for $1 \le i \le n$, which means we treat $\vec{\mathsf{Att}}_a$ as the lower bits of the key and $\vec{\mathsf{Att}}_b$ as the higher bits of the key $\vec{K}$. Then, we obtain the permutation $p_{a,b}$ of sorting by key $\vec{K}_{a,b}$ and apply it to obtain $p_{a,b}(\vec{\mathsf{Att}}_a)$ and $p_{a,b}(\vec{\mathsf{Att}}_b)$. By doing so, we can group the same value combination together and then count the frequency with only $O(n)$ communication. For the sorting part, if we use the existing state-of-the-art radix sorting protocol [1] to implement the sorting part, we can achieve $O(n \log u_{a,b})$ communication complexity for estimating a two-way marginal $\vec{\mathsf{M}}_{a,b}$.

Improved Secure Sorting with DP Leakage. It is obvious that the efficiency bottleneck of the Sort-and-count strategy lies in the secure sorting. We further improve this part to achieve $O(n)$ communication complexity. Our intuition is that if the histogram on a vector is already known, then revealing this vector to a server only leaks the order information. Following this observation, we require each client C_i to append dummy elements to his local column $\vec{\mathsf{Att}}_i$ to make the histogram (one-way marginal) on this vector satisfy DP. Then, when estimating two-way marginal $\vec{\mathsf{M}}_{a,b}$, it is secure to reveal the column $\vec{\mathsf{Att}}_a$ or $\vec{\mathsf{Att}}_b$ to a server if the elements in this

column have been shuffled. Since the "Select-Measure-Generate" workflow itself requires publishing the DP one-way marginals, revealing the shuffled columns $\vec{\mathsf{Att}}_a$ and $\vec{\mathsf{Att}}_b$ is identical to publish the DP one-way marginals, i.e., the server cannot know more information than what "Select-Measure-Generate" workflow allows.

Below, we describe the process of our protocol.

1. **Setup.** Before sharing the private columns, each client C_i samples u_i noise samples from the discrete Gaussian distribution: for $1 \le j \le u_i$, $\eta_{i,j} \sim \mathcal{N}(\sigma_1)$ with standard deviation σ_1 and appends $\eta_{i,j} + s$ lines of record $\{j\}$ to the end of the column $\vec{\mathsf{Att}}_i$. Here, s is a sufficiently large positive number, which serves as a public parameter to ensure that $\eta_{i,j} + s$ is a positive number since we can only append items to a column. Otherwise, the frequencies of some values will become negative and cannot be represented in a vector. Meanwhile, C_i creates a flag vector $\vec{\mathsf{F}}_i$. If the j-th element in $\vec{\mathsf{Att}}_i$ is dummy, then $\vec{\mathsf{F}}_i = 0$, else $\vec{\mathsf{F}}_i = 1$. At last, each client C_i secret-shares $\vec{\mathsf{Att}}_i$ and $\vec{\mathsf{F}}_i$ and sends to servers P_1, P_2, P_3.

2. **DP One-way Marginals**. Each client C_i first locally computes the one-way marginal $\vec{\mathsf{M}}_i$. Then, C_i adds the noise vector $\vec{\eta}_j = \{\eta_{i,1}, \ldots, \eta_{i,u_i}\}$ to marginal $\vec{\mathsf{M}}_i$ and obtain the DP one-way marginal $\tilde{\mathsf{M}}_i$. Finally, the client C_i publish $\tilde{\mathsf{M}}_i$ to all the servers.

3. **Shuffle.** To compute a marginal $\vec{\mathsf{M}}_{a,b}$, servers first pad additional dummies to ensure $\vec{\mathsf{Att}}_a$ and $\vec{\mathsf{Att}}_b$ have the same length. We use n_a to denote the length of $\vec{\mathsf{Att}}_a$ and n_b to denote the length of $\vec{\mathsf{Att}}_b$. For example, if $n_a < n_b$, we append $(n_b - n_a) \times \{\perp\}$ at the end of $\langle \vec{\mathsf{Att}}_a \rangle$ and $(n_b - n_a) \times \{0\}$ at the end of $\langle \vec{\mathsf{F}}_a \rangle$. Then, the servers run a secure shuffling protocol to shuffle $\langle \vec{\mathsf{Att}}_a \rangle$, $\langle \vec{\mathsf{Att}}_b \rangle$, $\langle \vec{\mathsf{F}}_a \rangle$ with a common random permutation (now all the entries in $\langle \vec{\mathsf{F}}_a \rangle$ and $\langle \vec{\mathsf{F}}_b \rangle$ are equal, so we only use $\langle \vec{\mathsf{F}}_a \rangle$). Now we have $n = n_a = n_b$.

4. **Local sort and Secure Permutation.** We first reveal the column $\pi(\vec{\mathsf{Att}}_a)$ to server P_1, who then locally computes the permutation p_a that can stably sort $\pi(\vec{\mathsf{Att}}_a)$. P_1 then secret-shares the sorted column $p_a \circ \pi(\vec{\mathsf{Att}}_a)$ to other servers and treat p_a as the input of a secure permutation protocol [1], which securely permutes $\pi(\vec{\mathsf{Att}}_b)$ and $\pi(\vec{\mathsf{F}}_a)$. Now, we already have three permuted secret vectors $\langle p_a \circ \pi(\vec{\mathsf{Att}}_a) \rangle$, $\langle p_a \circ \pi(\vec{\mathsf{Att}}_b) \rangle$ and $\langle p_a \circ \pi(\vec{\mathsf{F}}_a) \rangle$. The next step is revealing $p_a \circ \pi(\vec{\mathsf{Att}}_b)$ to server P_2. Similarly, P_2 compute the permutation p_b that securely sorts $p_a \circ \pi(\vec{\mathsf{Att}}_b)$ and then use p_b to securely permute $\langle p_a \circ \pi(\vec{\mathsf{Att}}_a) \rangle$ and $\langle p_a \circ \pi(\vec{\mathsf{F}}_a) \rangle$. At the end of this step, we obtain $\langle p_b \circ p_a \circ \pi(\vec{\mathsf{Att}}_a) \rangle$, $\langle p_b \circ p_a \circ \pi(\vec{\mathsf{Att}}_b) \rangle$ and $\langle p_b \circ p_a \circ \pi(\vec{\mathsf{F}}_a) \rangle$, which are identical to sort $\vec{\mathsf{Att}}_a$, $\vec{\mathsf{Att}}_b$ and $\vec{\mathsf{F}}_a$ using $\vec{\mathsf{Att}}_b$ as the first key and $\vec{\mathsf{Att}}_a$ as the second key. Now, we have grouped the same value combination together.

5. **Count and Compaction.** In this step, we compute the two-way marginal $\langle \vec{\mathsf{M}}_{a,b} \rangle$ given the sorted columns $\langle \vec{\mathsf{Att}}_a \rangle$, $\langle \vec{\mathsf{Att}}_b \rangle$ and flag vector $\langle \vec{\mathsf{F}}_a \rangle$. We initialize a binary vector $\langle \vec{B} \rangle$ of size n and compute the prefix sum as: $\langle \vec{y}[i] \rangle \leftarrow \sum_{j=1}^{i} \langle \vec{\mathsf{F}}_a[j] \rangle$. Then, for $1 \le i \le n-1$ in parallel, we compute $\langle B[i] \rangle \leftarrow 1 - \mathtt{EQ}(\langle \vec{\mathsf{Att}}_a[i] \rangle, \langle \vec{\mathsf{Att}}_a[i+1] \rangle)$ and set $\langle B[n] \rangle \leftarrow \langle 1 \rangle$. Now, the vector $\langle \vec{B} \rangle$ marks all the "last elements" at the end of their groups. At last, we run a secure compaction protocol. For each i such that $\langle \vec{B}[i] \rangle = 1$, the compaction moves the corresponding prefix sum $\langle \vec{y}[i] \rangle$ to the head of vector $\langle \vec{y} \rangle$. Finally, for $2 \le i \le u_{a,b}$, we set $\langle \vec{\mathsf{M}}_{a,b}[i] \rangle$ as: $\langle \vec{\mathsf{M}}_{a,b}[i] \rangle \leftarrow \langle \vec{y}'[i] \rangle - \langle \vec{y}'[i-1] \rangle$ and set $\langle \vec{\mathsf{M}}_{a,b}[1] \rangle \leftarrow \langle \vec{y}'[1] \rangle$.

Communication Cost. In step 2, we rely on calling the secure shuffling protocol three times to shuffle the vectors $\vec{\mathsf{Att}}_a$, $\vec{\mathsf{Att}}_b$, and $\vec{\mathsf{F}}_a$. After locally sorting each column, we use secure permutation twice to apply the permutation to another attribute column and the flag vector. In the three-server semi-honest setting, the secure shuffling has a communication complexity of $O(n)$ and a constant number of rounds. For the secure permutation, it can be implemented with two times of secure shuffling [1]. In step 3, all the prefix sums can be computed with the local addition of secret shares. All the computations of the secure equality test $\mathtt{EQ}(\langle \vec{\mathsf{Att}}_a[i] \rangle, \langle \vec{\mathsf{Att}}_a[i+1] \rangle)$ is performed in parallel. At last, we can set the key as $\langle B \rangle$ to sort the payload $\langle \vec{y} \rangle$. Since $\langle B \rangle$ is a boolean vector, applying radix sorting here only incurs $O(n)$ communication and constant rounds. At last, we can assume that the number of dummy items padded to each column is much smaller than the original length of the column. Thus, all the linear time operation on the column still has a communication complexity of $O(n)$. To sum up, the secure estimation of two-way marginals has a communication complexity of $O(n)$ and a constant number of rounds.

Security and Privacy Proof Sketch. Our security proof relies the ideal-real world paradigm [9]. In the ideal world, the functionality outs the DP one-way marginals and the shares of two-way marginals. In the real world, the additional views are the shuffled vectors $\pi(\vec{\mathsf{Att}}_a)$ revealed to P_1 and the $p_a \circ \pi(\vec{\mathsf{Att}}_b)$ revealed to P_2. These views can be simulated by the DP one-way marginals $\tilde{\mathsf{M}}_a$ and $\tilde{\mathsf{M}}_b$ output in Step 2. To simulate $\pi(\vec{\mathsf{Att}}_a)$, the simulator can construct a vector $\vec{\mathsf{Att}}_a'$ based on the one-way marginal $\tilde{\mathsf{M}}_a$ and then shuffle it with a random permutation π'. At last, we have $\pi'(\vec{\mathsf{Att}}_a') \approx \pi(\vec{\mathsf{Att}}_a)$. With a sim-

ilar idea, we can also simulate $p_a \circ \pi(\vec{\mathsf{Att}}_b)$. The proof of DP is standard and similar to the proof for PrivSyn and AIM [14, 22]. The additional statistical distance from standard Gaussian caused by MPC can be absorbed into the δ terms of approximated zCDP [5].

3.2 Secure Select-Measure

Our final protocol can compute an ideal functionality that can compute all the two-way marginals and one-way marginals, as well as the "Select" and "Measure" steps, which is formally defined in Appendix B. To achieve this, we combine the marginal estimation protocol and the following steps to implement the "Select-Measure" in the "Select-Measure-Generate" workflow. We discuss two instantiations for PrivSyn and AIM separately.

Marginal Selection for PrivSyn. PrivSyn computes the metric $\mathsf{InDif}_{a,b} = \|\vec{\mathsf{M}}_{a,b} - \vec{\mathsf{M}}_a \times \vec{\mathsf{M}}_b\|_1 + \mathcal{N}(0,\sigma^2), 1 \le a < b \le m$ for each two-way marginals [22]. To compute InDif in MPC, we need the secret shares of one-way marginals. Note that we can compute $\langle \vec{\mathsf{M}}_a \rangle$ by $\langle \vec{\mathsf{M}}_a[i] \rangle = \sum_{j=1}^{u_b} \langle \vec{\mathsf{M}}_{a,b}[u_a * i + j] \rangle$ and $\langle \vec{\mathsf{M}}_b \rangle$ by $\langle \vec{\mathsf{M}}_b[j] \rangle = \sum_{i=1}^{u_a} \langle \vec{\mathsf{M}}_{a,b}[u_a * i + j] \rangle$ with only local addition of shares.

In the original definition of PrivSyn, all the marginals are normalized into $[0,1]$. Here, to ensure the correctness, we compute the score in MPC as $\langle \mathsf{InDif}_{a,b} \rangle = \|\langle \vec{\mathsf{M}}_{a,b} \rangle \cdot n - \langle \vec{\mathsf{M}}_a \rangle \times \langle \vec{\mathsf{M}}_b \rangle\|_1 + \langle \mathcal{N}(0,\sigma^2) \rangle$, i.e., we multiply each entry of $\langle \vec{\mathsf{M}}_{a,b} \rangle$ by n. At last, we can reveal all the $\mathsf{InDif}_{a,b}$ to server P_1 and treat the selection on InDif as the post-processing. After the selection, P_1 returns the set of indexes for candidate two-way marginals X.

Marginal Selection for AIM. The computation cost of utility $score_{a,b} = w_{a,b} \cdot \|\vec{\mathsf{M}}_{a,b} - \vec{\mathsf{M}}'_{a,b}\|_1 + \Theta_{a,b}$ for AIM [14] is similar to PrivSyn, which is also $O(u_a u_b)$. Because $w_{a,b}$,$\Theta_{a,b}$ and $\vec{\mathsf{M}}'_{a,b}$ are all public values, it only takes $u_a u_b$ times of secure absolute value computations. However, selecting the final index x requires running an exponential mechanism [8] in MPC. Previous work relies on secure exponential operations [17]. In our instantiation, we implement the report noisy max mechanism in MPC to achieve the same result as the exponential mechanism, thus avoiding the heavy secure exponential operations. Specifically, we rely on a sub-protocol Π_{dLap} to sample noise shares from the discrete Laplace distribution [7]. After adding noise, we search for the maximum noise score $\langle score_x \rangle + \Pi_{\mathsf{dLap}}(0,b)$ and set $X = \{x\}$.

Measure. For $x \in X$, we add noise the selected two-way marginals as $\langle \tilde{\mathsf{M}}_x \rangle \leftarrow \langle \vec{\mathsf{M}}_x \rangle + \Pi_{\mathsf{dGauss}}(\sigma_2)$, where Π_{dGauss} is a protocol to generate discrete Gaussian noise sample in MPC [19]. The final output of our protocol is a set of noisy two-way marginals sent to server P_1. The later "Generate" step can be done by P_1 as a post-processing, without any MPC operations.

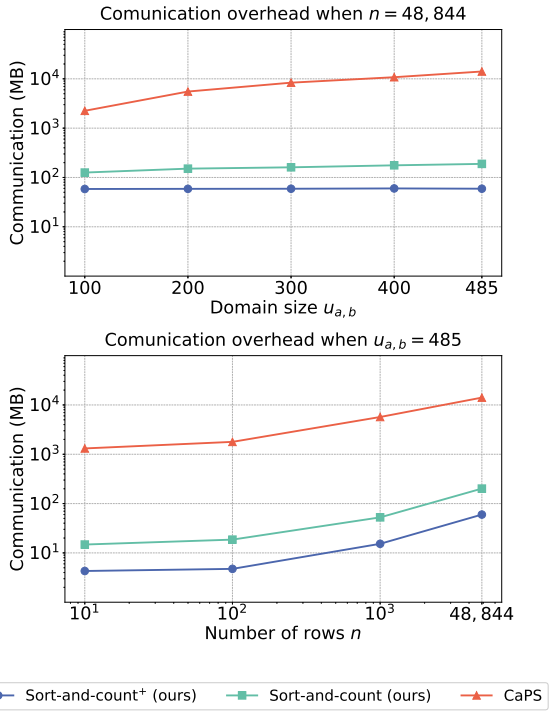

Figure 1: The communication overhead under different number of rows n and two-way marginal size $u_{a,b}$.

4 Experiment

We implement our protocol that computes the two-way marginals in the MP-SPDZ framework [11], a library for MPC in Python. We run all three servers P_1, P_2, P_3 on a single server with Intel Xeon Platinum 8369B 2.7GHz, Ubuntu 20.04, and 4GB memory.

We compare our protocol with two baselines, the CaPS protocol in [17], our solution with Sort-and-count strategy, and the improved Sort-and-count protocol with linear communication sorting (Sort-and-count$^+$). Our evaluation is conducted on the Adult dataset, with 48,844 rows of records. We use the first two columns, the domain size of this two-way marginal is 485. We vary the size of two-way marginal by manually splitting the domain into subranges of size $u \in \{100, 200, 300, 400, 485\}$ and vary the number of record $n \in \{10^2, 10^3, 10^4, 48,844\}$ by truncating the rows of the table.

As shown in Figure 1, both of our solutions significantly outperform existing marginal estimation protocol CaPS. Also, the overhead of our final protocol Sort-and-count$^+$ almost does not increase with domain size $u_{a,b}$, which is consistent with our expected communication complexity $O(n)$. Therefore, we conclude that our protocol for secure marginal estimation concretely outperforms the existing method CaPS.

References

[1] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 125–138, 2022.

[2] Gilad Asharov, Koki Hamada, Ryo Kikuchi, Ariel Nof, Benny Pinkas, and Junichi Tomida. Secure statistical analysis on multiple datasets: Join and group-by. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3298–3312, 2023.

[3] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[4] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

[5] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of cryptography conference*, pages 635–658. Springer, 2016.

[6] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282, 2017.

[7] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 486–503. Springer, 2006.

[8] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[9] Oded Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.

[10] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[11] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020.

[12] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of cryptology*, 22:161–188, 2009.

[13] Samuel Maddock, Graham Cormode, and Carsten Maple. Flaim: Aim-based synthetic data generation in the federated setting. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2165–2176, 2024.

[14] Ryan McKenna, Brett Mullins, Daniel Sheldon, and Gerome Miklau. Aim: an adaptive and iterative mechanism for differentially private synthetic data. *Proceedings of the VLDB Endowment*, 15(11), 2022.

[15] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*, pages 4435–4444. PMLR, 2019.

[16] Silvio Micali, Oded Goldreich, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM New York, NY, USA, 1987.

[17] Sikha Pentyala, Mayana Pereira, and Martine De Cock. Caps: Collaborative and private synthetic data generation from distributed sources. In *Forty-first International Conference on Machine Learning*, 2024.

[18] Mayana Pereira, Sikha Pentyala, Anderson Nascimento, Rafael T de Sousa Jr, and Martine De Cock. Secure multiparty computation for synthetic data generation from distributed data. *arXiv preprint arXiv:2210.07332*, 2022.

[19] Chengkun Wei, Ruijing Yu, Yuan Fan, Wenzhi Chen, and Tianhao Wang. Securely sampling discrete gaussian noise for multi-party differential privacy. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[20] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[21] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

[22] Zhikun Zhang, Tianhao Wang, Ninghui Li, Jean Honorio, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. Privsyn: Differentially private data synthesis. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 929–946. USENIX Association, 2021.

A MPC Primitives

Secure Multiparty Computation (MPC) [10, 20] allows a set of parties to jointly compute a function $y = f(D_1, ..., D_m)$ without revealing their inputs D_i $(1 \le i \le m)$. After the computation, all the parties can only know the result y. Currently, the two main paradigms to implement MPC are *garble circuits* [12, 16] and *secret sharing* [4, 6]. In this paper, we focus on a secret sharing scheme that offers better scalability in the multi-party setting.

Client-server Model. In this paper, we consider a classical way to enable privacy-preserving aggregation in the distributed setting, which is called the client-server model. In the client-server model, a set of clients owns the private data and is only responsible for some lightweight local computation. After creating the required inputs, they secret-share their private data into several helper servers. Upon receiving the private data (secret-shared) from clients, servers jointly run an MPC protocol to obtain the aggregation results. We assume the ground truth table D is distributed over multiple clients and build our system on three help servers P_1, P_2, P_3.

Security Model. We assume the help servers P_1, P_2, P_3 are semi-honest and non-colluded. In this assumption, a single corrupted server might attempt to infer the values in clients' columns, but follows the execution of the protocol. As for the clients, we first assume them to be semi-honest and show how to extend the protocol to be robust against malicious clients who might input invalid values to break the correct execution and skewed randomness to violate DP protection. For the proof of security, we rely on the ideal-real world paradigm [9] and work on the hybrid model. The servers can access the sub-protocol as a sub-functionality executed by a trusted party. We say that a protocol is constructed in a f-hybrid model if it accesses a sub-functionality f.

Functionality $\mathcal{F}_{\mathsf{Marginals}^2}$

Input: For $1 \le i \le m$, client C_i inputs the attribute column $\vec{\mathsf{Att}}_i$. The synthetic algorithm is PrivSyn or AIM.
Size leakage: Upon receiving $(\mathsf{QuerySize}, c)$ from the corrupted server P_c, send the length of columns n to P_c.
Marginal Estimation: Prepare the ground true marginals.

1. For $1 \le a < b \le m$, computes the two-way marginal $\vec{\mathsf{M}}_{a,b}$ over the value combinations under $\mathcal{U}_a \times \mathcal{U}_b$, based on column pairs $(\vec{\mathsf{Att}}_a, \vec{\mathsf{Att}}_b)$.
2. For $1 \le a \le m$, compute the one-way marginal $\vec{\mathsf{M}}_a$ for each column $\vec{\mathsf{Att}}_a$.
3. For $1 \le i \le m$, compute the noisy one-way marginal $\tilde{\mathsf{M}}_i = \vec{\mathsf{M}}_i + \mathcal{N}(0, \sigma^2 \mathbf{I})$ and send to P_1.

Select: Select a subset of two-way marginals.

- **PrivSyn:** Compute $\tilde{\mathsf{InDif}}_{a,b} = \|\vec{\mathsf{M}}_{a,b} - \vec{\mathsf{M}}_a \times \vec{\mathsf{M}}_b\|_1 + \eta_\sigma$, where η_σ is a value sampled from discrete Gaussian with standard deviation σ. Send all the $\tilde{\mathsf{InDif}}_{a,b}$ to P_1 and select a set of pairwise indexes X based on the marginal selection algorithm in [22].
- **AIM:** Apply exponential mechanism on $\vec{\mathsf{M}}_{a,b}$ (for $1 \le a < b \le m$) to select one pairwise index $X = \{x\}$. The utility function is defined as $u(a,b) = w_{a,b} \cdot \|\vec{\mathsf{M}}_{a,b} - \vec{\mathsf{M}}'_{a,b}\|_1 + \Theta_{a,b}$, where $w_{a,b}$ and $\Theta_{a,b}$ are public parameter defined in [14], $\vec{\mathsf{M}}'_{a,b}$ is the marginal on previously generated dataset.

Measure: Measure the selected two-way marginals. For each selected pairwise index $x \in X$, compute the noisy two-way marginal $\tilde{\mathsf{M}}_x = \vec{\mathsf{M}}_x + \mathcal{N}(0, \sigma^2 \mathbf{I})$ and send to P_1.

Figure 2: The ideal functionality $\mathcal{F}_{\mathsf{Marginal}^2}$ to select and measure the two-way marginals. The final generation is a post-processing executed by server P_1.

Linear Secret Shares. Our system is mainly built on the linear secret share (LSS). Specifically, we use three-party replicated secret shares [] on ring elements $\mathcal{R}$. To share secret value x, a client can choose three random elements $x_1, x_2, x_3 \in \mathcal{R}$. The server P_1 receives tuple (x_1, x_2), server P_2 receive (x_2, x_3), and server P_3 receive (x_3, x_1). We use

$\langle x \rangle$ to denote the secret shares of x. LSS requires no communication for the addition with a public value $a + \langle x \rangle$, multiplication with a public value $a \cdot \langle x \rangle$, and addition of two values $\langle x \rangle + \langle y \rangle$. We also rely on three operations that require communication among parties: the multiplication of shares $\langle x \rangle \cdot \langle y \rangle$, equality test $\texttt{EQ}(\langle x \rangle, \langle y \rangle)$, and conditional swap $\texttt{MUX}(\langle c \rangle, \langle x \rangle, \langle y \rangle)$, which returns $\langle x \rangle$ if condition $\langle c \rangle = \langle 1 \rangle$ else $\langle y \rangle$.

B Full Ideal Functionality

In this section, we formally define the functionality that achieves distributed "Select-Measure". We also discuss alternative approaches to show why the functionality should faithfully follow the "Select-Measure" process in the centralized setting.