CaPS: Collaborative and Private
Synthetic Data Generation from Distributed Sources

Sikha Pentyala [1]   **Mayana Pereira** [2]   **Martine De Cock** [1,3]

Abstract

Data is the lifeblood of the modern world, forming a fundamental part of AI, decision-making, and research advances. With increase in interest in data, governments have taken important steps towards a regulated data world, drastically impacting data sharing and data usability and resulting in massive amounts of data confined within the walls of organizations. While synthetic data generation (SDG) is an appealing solution to break down these walls and enable data sharing, the main drawback of existing solutions is the assumption of a trusted aggregator for generative model training. Given that many data holders may not want to, or be legally allowed to, entrust a central entity with their raw data, we propose a framework for collaborative and private generation of synthetic tabular data from distributed data holders. Our solution is general, applicable to any marginal-based SDG, and provides input privacy by replacing the trusted aggregator with secure multi-party computation (MPC) protocols and output privacy via differential privacy (DP). We demonstrate the applicability and scalability of our approach for the state-of-the-art select-measure-generate SDG algorithms MWEM+PGM and AIM.

1. Introduction

The success of data-driven applications in a variety of domains – including but not limited to healthcare, finance, and automation – can be attributed to the availability of data sharing for analysis. However, such sharing of data raises many privacy concerns. Laws and guidelines to protect user privacy in AI applications – such as the GDPR[1] in Europe and the AI Bill of Rights[2] in the United States – impose substantial barriers to data sharing across organisations, especially in highly regulated domains like healthcare and finance.
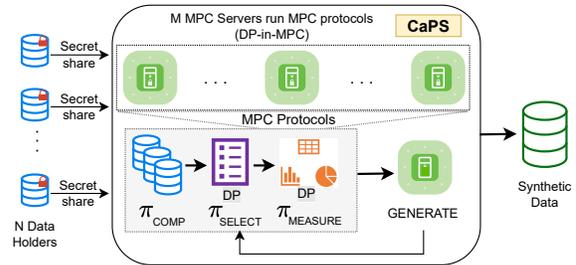

Figure 1. **CaPS**: A framework that leverages 'DP-in-MPC' to collaboratively and privately generate tabular synthetic data using marginal-based SDG techniques with the 'select-measure-generate' template. Servers run MPC protocols for 'select' and 'measure'. The 'generate' step is performed over differentially private measurements.

Synthetic data, i.e. artificial data generated using a purpose-built synthesizer trained on real data, offers an appealing solution to share microdata while mitigating privacy concerns. Synthetic data generation (SDG) with differential privacy (DP) has received considerable attention from the research community (see e.g. (Zhang et al., 2017; Jordon et al., 2019; McKenna et al., 2021; Tao et al., 2022) and references therein). This generated data fits any data processing workflow designed for the original data, while providing formal privacy guarantees.

Tabular data is ubiquitous in many domains ranging from healthcare, to humanitarian action, education, and socioeconomic studies Differentially private synthetic tabular data can be used for machine learning tasks and data analysis, as well as enable answering an arbitrary number of statistical queries. While many methods have been proposed in the literature to generate differentially private synthetic tabular

---

[1]School of Engineering and Technology, University of Washington Tacoma, USA [2]Department of Electrical Engineering, Universidade de Brasilia, Brazil [3]Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium. Correspondence to: Sikha Pentyala <sikha@uw.edu>.

[1]European General Data Protection Regulation
https://gdpr-info.eu/
[2]https://www.whitehouse.gov/ostp/ai-bill-of-rights/

data (DP-SD), such as GAN-based (e.g. (Jordon et al., 2019; Xu et al., 2019)) and marginal-based (e.g. (McKenna et al., 2021; 2022)) approaches, benchmarking studies have shown that marginal-based synthetic data generators often generate high quality tabular data with strong privacy guarantees (Tao et al., 2022). These marginal-based synthesizers often follow the common template of 'select-measure-generate' to generate synthetic data. The 'select' step involves choosing which marginals to measure from the data. The 'measure' step adds noise to perturb the measured marginals to preserve privacy. Finally, the 'generate' step employs algorithms to estimate a data distribution from the perturbed marginal measurements, and to sample synthetic data that closely preserves these measured marginals.

Problem. Existing SDG approaches operate in *the centralized paradigm*, i.e. they assume that the real data needed to train the synthesizer resides with one data holder, or, if the data originates from different data holders, that the latter are able to send their data to a trusted aggregator who in turn will use it as input for SDG algorithms. Large amounts of valuable data however are under the control of data holders (companies, hospitals, biomedical research institutes etc.) who cannot show their data to each other or to a central aggregator without raising privacy concerns. This is the bottleneck that we address, namely *how to generate synthetic data based on the combined data from distributed data holders who are not able to disclose their data*. This includes data that is horizontally distributed, such as data across different hospitals who each have their own patients, as well as data that is vertically distributed such as in scenarios where a patient's data is distributed across multiple entities (hospitals, companies, labs, etc.). In addition to the cross-silo scenarios described above, our proposed solution would also make a scenario practical where millions of users (patients) could provide their data to produce a synthetic dataset in a way that private, individual data would never be exposed in the clear to any entity – practically implementing a 'synthetic data as a service' model.

There is a substantial gap in the literature on solutions for SDG from distributed data sources. While approaches such as (Xin et al., 2020b; Behera et al., 2022), which are based on federated learning (FL) (Kairouz et al., 2021b), seem to provide a viable solution for this problem, with the caveat that they work either for horizontal (HFL) or for vertical (VFL) data partitioning, not both. Moreover, they rely on a single honest-but-curious aggregator, hence a single point of failure. Decentralized paradigms such as peer-to-peer solutions that remove the need of central entity, adapt approaches based on local differential privacy (LDP) (Kasiviswanathan et al., 2011) or distributed differential privacy (Kairouz et al., 2021a) which are known to result in utility loss (Pasquini et al., 2023). Ramesh et al., 2023 proposed to use secure multi-party computation combined with specialized hardware (a trusted execution environment) to generate synthetic

data from distributed sources. To the best of our knowledge, there are currently no solutions that can generate differentially private synthetic data from distributed data sources using marginal-based synthesizers that work for (a) arbitrary data partitioning (horizontal, vertical, or mixed), (b) provide utility at par with that of the centralized paradigm (c) do not rely on a single trusted entity and (d) do not need specialized hardware.

Our Contributions. We observe that SDG techniques based on the 'select-measure-generate' template only need to perform computations on the private data in the 'select' and 'measure' steps. The outputs of these steps are protected with DP guarantees. Computations thereafter, such as the 'generate' step, are *post-processing*[3] steps. Following this observation, it is sensible to (1) secret share the data, (2) carry out the 'select' and 'measure' steps with differential privacy in secure multi-party computation protocols (DP-in-MPC) over the secret shared data, and (3) publish the perturbed marginals for consumption by the 'generate' step.

We capture this idea in a framework called **CaPS** (Fig. 1) to generate differentially private synthetic tabular data from distributed data holders that works for any kind of data partitioning (horizontal, vertical, or mixed). **CaPS** uses MPC to provide *input privacy* by replacing the trusted entity from the centralized paradigm with MPC servers. Unlike in the traditional central paradigm, these computing servers that execute the MPC protocols never see the private data of the data holders in the clear. To provide *output privacy* guarantees, **CaPS** uses differential privacy as in the centralized paradigm. The crucial difference however is that perturbations to provide DP are performed *within* the MPC protocols, effectively having the MPC servers simulate the role of a trusted curator implementing differential privacy mechanisms. This 'DP-in-MPC' approach achieves utility at par with the centralized paradigm as shown in our experiments.

We designed **CaPS** to be modular. The individual algorithms for each of the select, measure, and generate steps can be replaced. Thus, the MPC protocols for the select and measure steps can be replaced or switched. This means that **CaPS** can be easily adapted to any SDG algorithm by simply replacing or designing the MPC protocols for the given select and measure steps. Moreover, the underlying MPC primitives in the protocols can also be replaced with primitives for different adversarial models and schemes or with much more efficient primitives in the future. To illustrate this, we apply **CaPS** to create privacy-preserving collaborative counterparts of the state-of-the-art SDG algorithms MWEM+PGM (Mckenna et al., 2019) and AIM (McKenna et al., 2022). We design MPC protocols for the select and measure steps of these algorithms and develop pipelines to

---

[3]Outputs of differentially private mechanisms are immune to post-processing (Dwork & Roth, 2014).

generate tabular synthetic data. We evaluate these pipelines for performance, measuring runtime and communication cost. We assess the quality of the synthetic data by reporting the average workload error on benchmark datasets. We also assess the utility of the generated synthetic data for downstream machine learning (ML) tasks by reporting AUC and F1 scores obtained with logistic regression and random forest models trained on the benchmark data. Our evaluations demonstrate that **CaPS** generates differentially private synthetic data of the same level of quality as in the centralized paradigm, while in addition providing input privacy, making it suitable for collaborative SDG from distributed data holders. We summarize our main contributions as:

- We are the first to adapt a 'DP-in-MPC' approach to generate synthetic data from arbitrarily partitioned private data sources.
- We propose a framework, **CaPS**, to generate differentially private synthetic tabular data from distributed private data sources. Our framework is applicable to any data distribution without the need for a trusted entity or specialized hardware. This framework achieves utility as in the centralized paradigm.
- We design MPC protocols for the select and measure steps of two state-of-the art marginal based synthetic data generators (MWEM+PGM and AIM), and showcase the applicability of **CaPS** to create privacy-preserving versions of these SDG algorithms.
- We demonstrate the effectiveness of **CaPS** via utility and performance experiments over benchmark datasets.

This paper surpasses a prior non-archival workshop paper in which we explored an early DP-in-MPC version of an older SDG algorithm (Hardt & Rothblum, 2010) on horizontally distributed data.

2. Preliminaries

Consider a dataset D with n instances. D has a set of d attributes denoted by $x = \{x_1, x_2, \ldots, x_d\}$. The domain for the attribute x_i is finite and is given by Ω_i, i.e. $|\Omega_i| = \omega_i$. The domain for x is the cartesian product of the domains of the individual attributes, i.e. $\Omega = \prod_{i=1}^{d} \Omega_i$.

Synthetic Data Generation. By a marginal on a set of attributes $q \subseteq x$ we mean a histogram computed over D for the attributes in q, i.e. it counts the number of occurrences in D for each $t \in \Omega_q$ where $\Omega_q = \prod_{x_i \in q} \Omega_i$. A k-way marginal is a marginal computed on k attributes, i.e. $|q| = k$. We denote the result of the computation of a marginal on q over D as $\mu_q(D)$, which is basically a vector of counts.[4]

In what follows, we refer to a set of attributes as a *query*. Marginal-based SDG algorithms aim to maintain the values of marginals for a predefined set of queries Q (called a *work-*

_____
[4]We consider $\mu_q(D)$ to be a flattened vector for any k-way marginal.

load) when computed over the synthetic data versus over the original data. The algorithm typically repeats three steps in an iterative fashion: select, measure, and generate synthetic data $\hat{D}$. In the the select step the algorithm usually takes the answers to the workload Q on real data and intermediate synthetic data, i.e. $\{\mu_q(D) \,|\, q \in Q\}$ and $\{\mu_q(\hat{D}) \,|\, q \in Q\}$ respectively, and selects the query q_s that gives the maximum error in a differentially private manner. The measure step computes the noisy answer $\hat{\mu}_s = \mu_s(D) + N(.)$ where $N(.)$ is the noise added to satisfy DP. The generate step takes the noisy answer $\hat{\mu}_s$ as input to provide an new estimate of $\hat{D}$.

Secure Multiparty Computation (MPC). MPC is an umbrella term for cryptographic approaches that enable private computations among mutually distrustful parties (Cramer et al., 2000; Evans et al., 2018). In this paper, we follow the "MPC as a service" paradigm, where data holders delegate their computation to a set of non-colluding, but otherwise non-trusted servers. Moreover, we will work with secret shared-based MPC protocols. In such protocols, each data holder maps their secret inputs z in secret shares according to the given MPC scheme in use. We denote secret shares of z as $[\![z]\!]$. The data holders then distribute the shares to the set of servers which acts as the computing parties in MPC. Though z can be reconstructed when all shares are combined, none of the MPC servers by themselves learn anything about the value of z. These servers run MPC protocols to jointly perform computations over the secret shares, in our case computations for generating synthetic data. As all computations are done over the secret shares, so that the servers do not learn the values of the inputs nor of intermediate results, i.e. MPC provides *input privacy*.

Threat Model. MPC protocols are designed to prevent and detect attacks by an adversary corrupting one or more parties to learn private information or to cause the result of the computation to be incorrect. The adversary – which we assume to be static – can have different levels of adversarial power. In the *semi-honest* model, even corrupted parties follow the instructions of the protocol, but the adversary attempts to learn private information from the internal state of the corrupted parties and the messages that they receive. MPC protocols that are secure against semi-honest or *"passive"* adversaries prevent such leakage of information. In the *malicious* adversarial model, the corrupted parties can arbitrarily deviate from the protocol specification. Providing security in the presence of malicious or *"active"* adversaries, i.e. ensuring that no such adversarial attack can succeed, comes at a higher computational cost than in the passive case.

Chosen MPC Schemes. The protocols that we propose in Sec. 3 are sufficiently generic to be used in settings with passive or active adversaries. This is achieved by changing the underlying MPC scheme to align with the desired security setting. We evaluate our protocols in an honest-majority 3-party computing setting out of which at most one party

can be corrupted (3PC) (Araki et al., 2016; Dalskov et al., 2021), an honest-majority 4-party computing setting with one corruption (4PC) (Dalskov et al., 2021), and a dishonest-majority 2-party computation setting where each party can only trust itself (2PC) (Cramer et al., 2018).

We now give a brief introduction to MPC where we closely follow Section 3 in (Pentyala et al., 2021).

Data Representation in MPC. MPC works for inputs defined over a finite field or ring. Since inputs for synthetic data set generation algorithms are finite precision real numbers, we convert all of our inputs to fixed precision (Catrina & Saxena, 2010) and map these to integers modulo q, i.e., to the ring $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$, with q a power of 2. In fixed-point representations with a fractional bits, each multiplication generates a additional fractional bits. To securely remove these extra bits, we use the deterministic truncation protocol by (Dalskov et al., 2019).

Replicated sharing-based 3PC. To give the reader an understanding of how MPC protocols work, we give a brief description of a specific MPC protocol based on replicated secret sharing with three computing parties (Araki et al., 2016). A private value x in $\mathbb{Z}_q$ is secret shared among servers (parties) S_1, S_2, and S_3 by sselecting uniformly random shares $x_1, x_2, x_3 \in \mathbb{Z}_q$ such that $x_1 + x_2 + x_3 = x \mod q$, and giving (x_1, x_2) to S_1, (x_2, x_3) to S_2, and (x_3, x_1) to S_3. We denote a secret sharing of x by $[\![x]\!]$.

From now on, servers will compute on the secret shares of x rather than on x itself. In order to proceed with the computation of a function, we need a representation of such function as a circuit consisting of addition and multiplication gates. The servers will compute the function gate by gate, by using specific protocols for computing addition of a publicly known constant to a secret shared value, addition of two secret shared values, multiplication of a secret shared value times a public constant, and multiplication of two secret shared values. After all the gates representing the function have been evaluated, the servers will hold secret shares of the desired final result of the computation.

The three servers are capable of performing operations such as the addition of a constant, summing of secret shared values, and multiplication by a publicly known constant by doing local computations on their respective shares.

To multiply secret shared values $[\![x]\!]$ and $[\![y]\!]$, we can use the fact that $x \cdot y = (x_1 + x_2 + x_3)(y_1 + y_2 + y_3) \mod q$. This means that S_1 computes $z_1 = x_1 \cdot y_1 + x_1 \cdot y_2 + x_2 \cdot y_1 \mod q$, S_2 computes $z_2 = x_2 \cdot y_2 + x_2 \cdot y_3 + x_3 \cdot y_2 \mod q$, and S_3 computes $z_3 = x_3 \cdot y_3 + x_3 \cdot y_1 + x_1 \cdot y_3 \mod q$. The next step is for the servers to obtain an additive secret sharing of 0 by choosing random values u_1, u_2, u_3 such that $u_1 + u_2 + u_3 = 0 \mod q$. This can be done using pseudorandom functions. Each server S_i then computes $v_i = z_i + u_i \mod q$. Finally, S_1 sends v_1 to S_3, S_2 sends v_2 to S_1, and S_3 sends v_3 to S_2. This allows the servers

S_1, S_2, and S_3 to obtain the replicated secret shares (v_1, v_2), (v_2, v_3), and (v_3, v_1), respectively, of the value $v = x \cdot y$.

This protocol can be proven secure against honest-but-curious adversaries. In such case, corrupted players follow the protocol instructions but try to obtain as much knowledge as possible about the secret inputs from the protocol messages, and locally stored information. We can adapt this protocol to be secure even in the case of malicious adversaries. Those can arbitrarily deviate from the protocol in order to break its privacy. For the malicious case, we use the MPC scheme proposed by (Dalskov et al., 2021) as implemented in MP-SPDZ (Keller, 2020a).

Differential Privacy (DP). Consider two neighboring datasets D and D' that differ in a single instance, i.e. D' can be obtained either by adding or removing an instance from D or vice-versa. A randomized algorithm F is differentially private if it generates similar output probability distributions on D and D' (Dwork & Roth, 2014). This implies that the presence or absence of an instance does not affect the output probability distribution of F, thus providing *output privacy* by limiting the amount of information that the output reveals about any instance. Formally, a randomized algorithm F is called (ϵ, δ)-DP if for all pairs of neighboring sets D and D', and for all subsets O of F's range,

$$\Pr(F(D) \in O) \le e^\epsilon \cdot \Pr(F(D') \in O) + \delta. \tag{1}$$

where ϵ is the privacy budget or privacy loss and δ is the probability of violation of privacy. The smaller these values, the stronger the privacy guarantees.

3. CaPS: Collaborative and Private SDG

Problem Formulation. We consider a set of N honest data holders $\mathbf{H} = \{H_1, H_2, \ldots, H_N\}$. Each data holder H_i holds a dataset D_i such that[5] $D = \bigcup_{i=1}^N D_i$. A marginal-based synthesizer F takes a workload Q and Ω as input and outputs DP tabular synthetic data $\hat{D}$. F is as defined for the centralized paradigm. Our goal is to generate $\hat{D}$ in the distributed setting with $\mathbf{H}$. To do so, we consider a set of M independent MPC servers $\mathbf{S} = \{S_1, S_2, \ldots, S_M\}$ that perform computations over distributed private data.

Algorithm 1 outlines the flow of our proposed framework **CaPS**. The key idea is to identify the steps of the SDG algorithm F that require computations across data from multiple data holders and invoking the MPC servers to perform such computations. If the output of these computations is differentially private, the outputs can be revealed and the other steps can be done in-the-clear to avoid the overhead due to MPC.

---

[5]We do not make any assumptions on how exactly the data is partitioned among the data holders, i.e. each data holder may have (partial) rows and/or (partial) columns of D.

Algorithm 1 CaPS: Generating tabular synthetic data with DP-in-MPC in the select-measure-generate template

---

Input: Domain Ω, Queries Q
Output: Synthetic Data $\hat{D}$
Parameters: Privacy parameters ϵ, δ
Hyper-Parameters: Hyper-parameters for F

1: **H**: $[\![\mu_q^i]\!], [\![D_i]\!] \leftarrow$ LOCAL_COMPUTE( )
2: Initialization by S_1: $\hat{D} \leftarrow$ INIT( )
3: **COMPUTE ANSWERS**: $[\![\mu_q]\!] \leftarrow \pi_{\text{COMP}}([\![\mu_q^i]\!], [\![D_i]\!])$
4: **repeat**
5:     **SELECT**: $q_s \leftarrow \pi_{\text{SELECT}}([\![\mu_q]\!], \hat{\mu}_q)$ //S_1: $\hat{\mu}_q$ on $\hat{D}$
6:     **MEASURE**: $\hat{\mu}_s \leftarrow \pi_{\text{MEASURE}}([\![\mu_{q_s}]\!], q_s)$
7:     **GENERATE** by S_1: $\hat{D} \leftarrow F(\hat{\mu}_s, q_s)$
8: **until** terminate

3.1. Setup Phase

The framework begins with a setup phase in which the MPC servers $\mathbf{S}$ load and compile the MPC protocols for the chosen F, the dimensions of the dataset D, and the MPC scheme. On Line 1 in Alg. 1, the data holders $\mathbf{H}$ perform local computations as defined by LOCAL_COMPUTE(); see below for a brief description. We consider S_1 to be the initiating server that is responsible for generating $\hat{D}$ and publishing the generated synthetic data to the required stakeholder. S_1 begins with randomly initializing $\hat{D}$ as defined by INIT() on Line 2. We note that this initialization is the same as in the centralized version of F.

Description of LOCAL_COMPUTE(). Each data holder locally computes answers to the queries in the workload Q to the best of its ability. We denote the answers computed by H_i as $\mu_q^i, \forall q \in Q$. If data holder H_i does not have all the attributes contained in q, then H_i is unable to compute a local answer for q, so μ_q^i is assigned a vector of 0s. All the data holders then convert their local answers into secret shares as per the given secret-sharing scheme as $[\![\mu_q^i]\!]$. In a non-horizontal distribution scenario, each H_i additionally secret shares their private local dataset D_i as $[\![D_i]\!]$.

The setup phase is followed by the training phase where the MPC servers $\mathbf{S}$ run the MPC protocols for F to generate synthetic data $\hat{D}$ (Lines 3 – 8 in Alg. 1). This phase in general does not require the N data holders to stay online or participate in the generation process. However, to reduce MPC overhead, one can optimize **CaPS** depending on the availability of any of the data holders (See Section 3.4). The training phase proceeds with invoking MPC protocols for computing query answers μ_q over the (secret shared) real data (π_{COMP}), and then repeatedly selecting a query q_s (π_{SELECT}) that needs attention and measuring/preparing a noisy answer $\hat{\mu}_s$ (π_{MEASURE}) that can be disclosed.

3.2. Computation of Answers on Distributed Data

The MPC servers run π_{COMP} to compute secret shares of $[\![\mu_q]\!]$ (Line 3 in Alg. 1). In an arbitrary setting, π_{COMP} can compute $[\![\mu_q]\!]$ by directly performing computations on $[\![D_i]\!], \forall i$. Such computations do not require data holders to perform local computations of answers, but does increase the MPC overhead. To optimize MPC overhead, we propose for each query q that all the data holders who have all attributes in q perform one-time local computations to compute partial query answers $[\![\mu_q^i]\!]$ (see Line 1 in Alg. 1); note that for queries involving only one attribute, i.e. one-way marginals, many such local computations can be done. The first step in π_{COMP} is then to simply add the secret shares of the locally computed answers from each data holder $[\![\mu_q^i]\!], \forall q \in Q$ (Lines 1–5 in Prot. 2). This is then followed by computing secret shares of workload answers $[\![\mu_{q^*}]\!]$ on $[\![D_i]\!]$ for queries q^* that can not be computed locally, i.e. queries containing attributes that are distributed across different data holders. We call the set of these queries Q^*. After the execution of π_{COMP}, the MPC servers $\mathbf{S}$ hold the secret shares of $[\![\mu_q]\!], \forall q \in Q$.

Note that when the data is horizontally partitioned, $Q^* = \emptyset$, and the workload answers $[\![\mu_q]\!]$ can be fully computed by adding secret shares of locally computed results, i.e. Lines 1–5 in Prot. 2. This results in a total of $(N-1) \cdot |Q| \cdot \max(\omega_q)$ additions in MPC implying that π_{COMP} is linear in the number of data holders for a given Q and Ω for horizontally distributed data. When $Q^* \neq \emptyset$, secret sharings of the complete query answers can be obtained by letting the servers $\mathbf{S}$ perform more computation over the secret-shared data of the data holders, as explained below.

Protocol 2 π_{COMP}:
MPC Protocol for **COMPUTE ANSWERS**

Input: Secret shares of locally computed answers $[\![\mu_q^i]\!]$, Secret shares of data $[\![D_i]\!]$ (needed if not horizontal), Queries Q, including queries Q^* with attributes that are distributed among data holders, Domain Ω
Output: $[\![\mu_q]\!], \forall q \in Q$
1: **for** $i = 1$ to N **do**
2:     **for all** $q \in Q$ **do**
3:         $[\![\mu_q]\!] \leftarrow [\![\mu_q]\!] + [\![\mu_q^i]\!]$
4:     **end for**
5: **end for**
6: **if** $Q^* \neq \emptyset$ **then**
7:     $[\![D]\!] \leftarrow \pi_{\text{JOIN}}([\![D_i]\!] | i = 1 \ldots N)$
8: **end if**
9: **for all** $q^* = \{a_1, a_2\}$ in Q^* **do**
10:     **for** $i = 1$ to n **do**
11:         $[\![x]\!] \leftarrow [\![D_{a_1}[i]]\!]$
12:         $[\![y]\!] \leftarrow [\![D_{a_2}[i]]\!]$
13:         **for** $j \in \Omega_{a_1}$ and $k \in \Omega_{a_2}$ **do**
14:             $[\![m]\!] \leftarrow \pi_{\text{MUL}}(\pi_{\text{EQ}}([\![x]\!], j), \pi_{\text{EQ}}([\![y]\!], k))$
15:             $[\![\mu_{q^*}[j * |\Omega_{a_2}| + k]]\!] \leftarrow [\![\mu_{q^*}]\!] + [\![m]\!]$
16:         **end for**
17:     **end for**
18: **end for**
19: **return** $[\![\mu_q]\!], \forall q \in Q$

MPC protocol π_{COMP}**.** Prot. 2 computes $[\![\mu_q]\!]$ and is optimized for AIM and MWEM+PGM in an arbitrary distributed setting. The workload Q in these algorithms consists of 1-way and 2-way marginals. Lines 1–5 aggregate the workload answers from all the data holders by performing addition of secret shares of $[\![\mu_q^i]\!]$. This aggregation accounts for computation of 1-way marginals for any arbitrary setting and 2-way marginals for the horizontal setting.

If $Q^* \neq \emptyset$, then **S** proceed to assemble a secret-sharing of the joint dataset using the protocol π_{JOIN} on Line 7. π_{JOIN} initializes a matrix $[\![D]\!]$ of dimensions $n \times d$. We assume that all the data samples are aligned before π_{JOIN}.[6] π_{JOIN} then combines all the secret-sharings $[\![D_i]\!]$ into a secret-sharing $[\![D]\!]$ of the overall dataset D using simple assignment statements. In Protocol 2, π_{JOIN} requires $n.d$ assignment operations and works for any number of data holders. **S** then run Lines 9–18 to compute secret-sharings of 2-way marginals. Consider a 2-way marginal $q^* \in Q^*$ which is represented as a pair $q^* = \{a_1, a_2\}$. $[\![D_{a_1}]\!]$ denotes the secret shares of $[\![D]\!]$ for the a_1 attribute and $[\![D_{a_2}]\!]$ for a_2. On Lines 10–17, the MPC servers iterate over all the n data samples in $[\![D_{a_1}]\!]$ and $[\![D_{a_2}]\!]$ to compute the number of occurrences of all the combinations of values in the domains Ω_{a_1} and Ω_{a_2}. Line 14 in Prot. 2 relies on MPC primitives for multiplication π_{MUL} and equality testing π_{EQ} to check if a combination of attribute values occurs in the data. For details about these primitives, see Appendix A. On Line 14, the MPC servers compute $[\![m]\!]$ which is a secret sharing of 0 or 1 that adds to the the number of occurrences of the combination of attributes values held in $[\![\mu_{q^*}]\!]$. We note that a major MPC overhead is due to Line 14 performed in the loop. This indicates that for an arbitrary data setting, there are $(N-1) \cdot |Q| \cdot \max(\omega_q) + n \cdot \max(\omega_q)^2 \cdot |Q^*|$ additions, $2 \cdot n \cdot \max(\omega_q)^2 \cdot |Q^*|$ equality checks, and $n \cdot \max(\omega_q)^2 \cdot |Q^*|$ multiplications. Given a fixed set of queries Q and fixed domain Ω, Protocol 2 grows linearly in the total number of samples n and the total number of data holders N.

MPC protocols are typically designed as specific circuits, composed of a sequence of addition and multiplication gates. Therefore, changing the functionality to be implemented in MPC usually requires the design of a new circuit. Protocol 2 is specifically designed to suit algorithms that consider 1-way and 2-way marginals. One can extend Protocol 2 to compute p-way marginals as discussed in Appendix A. We consider optimizing the protocol for different scenarios as future work.

3.3. Selection of the Query

In the select step, **S** run the MPC protocol π_{SELECT} to select the query q_s based on which the estimate of $\hat{D}$ should be improved (Line 5 in Alg. 1). We note that π_{SELECT} is independent of the data distribution setting as the set of

---

[6]This can be achieved using protocols for Private Set Intersection (PSI) available in literature.

Protocol 3 π_{SELECT}: MPC Protocol for **SELECT**

Input: Secret shares of $[\![\mu_q]\!]$, Estimated answer $\hat{\mu}_q$, length of domains ω_q for each q, privacy parameter b, sensitivity w
Output: Selected query q_s
1: $[\![err]\!] \leftarrow \pi_{\mathsf{ERR}}([\![\mu_q]\!], \hat{\mu}_q)$
2: $[\![err]\!] \leftarrow \pi_{\mathsf{NORM}}([\![err]\!])$
3: $[\![prob]\!] \leftarrow \pi_{\mathsf{PROB}}([\![err]\!], b, w, \omega_q)$
4: $[\![s]\!] \leftarrow \pi_{\mathsf{RC}}([\![prob]\!])$
5: Reveal s
6: **return** q_s

MPC servers **S** already have secret shares of the computed answers $[\![\mu_q]\!]$ from all the data holders. Prot. 3 provides a general template for selecting q_s. S_1 who holds $\hat{D}$ computes $\hat{\mu}_q$ and provides it as input to Prot. 3. On Line 1 the MPC servers compute a secret-sharing of the error $[\![err]\!]$ between $[\![\mu_q]\!]$ and $\hat{\mu}_q$ by taking the sum of absolute differences of the answers using π_{ERR}. This is followed by computing secret shares of the normalized errors $[\![err]\!]$ using π_{NORM} on Line 2. A secret shared probability vector over the queries is then constructed using π_{PROB} on Line 3 which takes the secret-shared normalized errors $[\![err]\!]$, the privacy parameters – the scale b and sensitivity w – as input. We describe the MPC subprotocols π_{ERR}, π_{NORM} and π_{PROB} in Appendix A. What happens within these protocols may differ from one SDG algorithm to the next; we discuss these subprotocols specifically for AIM and MWEM+PGM in protocols 6 and 8 respectively in Appendix A. Finally on Line 4, **S** run subprotocol π_{RC} to randomly select the query using exponential mechanism, thus implementing the DP-in-MPC paradigm. π_{RC} outputs the secret shares of the randomly chosen index $[\![s]\!]$, which is revealed to S_1. The query with index s is selected as q_s. The MPC overhead due to π_{SELECT} is in the order of $|Q| \times \max(\omega_q)$ of the computations involved. This means that π_{SELECT} does not depend on the number of data holders N or the total number of samples n but on he given number of queries $|Q|$ and the domain of the dataset Ω.

MPC protocol for π_{RC}**.** Protocol 4 takes as input the secret shares of the computed probabilities, $[\![prob]\!]$ and chooses the first index of the probability vector for which the probability value satisfies a condition based on random threshold. Lines 1–8 compute the random threshold $[\![t]\!]$ using the MPC primitives for multiplication (π_{MUL}), random number generation ($\pi_{\mathsf{GR-RANDOM}}(0, 1)$). Lines 10–15 select the index conditioned on the threshold without exiting the loop, as it could reveal the value of returned index to the adversary. We design Lines 10–15 to prevent such side-channel attacks. To understand this part of the code, note that we have a list $p[1..|Q|]$ of non-decreasing values, i.e. the cumulative probability sums, and the MPC servers have to find the first index i in $p[1..|Q|]$ for which $p[i] > t$. In a mock example with $|Q| = 10$, and assuming that the first such $p[i]$ value is at position 7, the tests on Line 11 will generate the results 0,0,0,0,0,0,1,1,1,1. On Line 12, these results are accumulated in k, which eventually becomes 4, and the desired

index is computed as $N - (k-1) = 10 - 3 = 7$. Lines 14–15 take care of the edge case when $p[i] \leq$ t for all i (i.e. k is 0). We protect the value of k by employing MPC primitives for multiplication to simulate a conditional statement.

Protocol 4 π_{RC}: MPC Protocol for random selection

Input: Secret shares of probability vector $[\![prob]\!]$, length of probability vector $|Q|$
Output: Secret shares of selected index $[\![s]\!]$

1: $sum \leftarrow 0$
2: Initialize a vector $\mathbf{p}$ of length N
3: **for** $i = 1$ to $|Q|$ **do**
4: $[\![sum]\!] \leftarrow [\![sum]\!] + [\![prob[i]]\!]$
5: $[\![p[i]]\!] \leftarrow [\![sum]\!]$
6: **end for**
7: $[\![r]\!] \leftarrow \pi_{\mathsf{GR-RANDOM}}(0, 1)$ // with protocol for random number generation $\pi_{\mathsf{GR-RANDOM}}$
8: $[\![t]\!] \leftarrow \pi_{\mathsf{MUL}}([\![sum]\!], [\![r]\!])$
9: $k \leftarrow 0$
10: **for** $i = 1$ to $|Q|$ **do**
11: $[\![c]\!] \leftarrow \pi_{\mathsf{GT}}([\![p[i]]\!], [\![t]\!])$
12: $[\![k]\!] \leftarrow [\![k]\!] + [\![c]\!]$
13: **end for**
14: $[\![c]\!] \leftarrow \pi_{\mathsf{EQ}}([\![k]\!], 0)$
15: $[\![s]\!] \leftarrow N - \pi_{\mathsf{MUL}}([\![k]\!] - 1, 1 - [\![c]\!])$
16: **return** $[\![s]\!]$

3.4. Measuring Answer to the Selected Query

In the measure step, **S** run the MPC protocol π_{MEASURE} to compute the noisy answer to selected query q_s (Line 6 in Algorithm 1). As **S** holds the secrets shares $[\![\mu_s]\!]$ computed for any arbitrary data distribution, π_{MEASURE} computes $[\![\hat{\mu}_s]\!]$ by generating the secret shares of noise vector $[\![\gamma]\!]$ and adding it to the $[\![\mu_s]\!]$. Once the **S** compute $[\![\hat{\mu}_s]\!]$, it is revealed to S_1 for generation step.

The MPC overhead is mainly due to generation of $[\![\gamma]\!]$. To reduce MPC overhead in an arbitrary distributed setting, one can consider the availability of the data holder who holds the attributes defined in q_s and request for generation of the noisy vector (S_1 plays the role of sending request and receiving response). This is a direct optimization in a vertical distributed setting when the selected query is a 1-way marginal. However, the advantage of generating noise in MPC is that the noise is added in a correct and private manner, such that private inputs cannot be reconstructed back. One can also optimize the MPC primitives to generate the secret shares of noise as we consider in Protocol 5.

Protocol 5 computes Gaussian noise as required by AIM and MWEM+PGM for arbitrary distributed setting. To do so, we consider Irwin-Hall approximation to generate Gaussian samples on Lines $2 - 7$ as it reduces the MPC overhead considerably. The protocol for generating noise requires $(13.\omega_s + \max(\omega_q))$ additions and $12.\omega_s$ random bit generations. These lines can be replaced by other MPC protocols for other sampling techniques such as Box-Muller method as we show in the Prot. 9 in Appendix A.

Protocol 5 π_{MEASURE}: MPC Protocol for **MEASURE** using Gaussian noise

---

Input: Secrets shares of $[\![\mu_{q_s}]\!]$, length of domain ω_s for q_s, scale b
Output: Noisy measurement $\hat{\mu}_s$

1: Initialize vector $[\![\gamma]\!]$ of length $\max(\omega_r)$ with 0s
2: **for** i = 0 to ω_{q_s} **do**
3: $[\![sum]\!] \leftarrow 0$
4: **for** j = 0 to 12 **do**
5:    $[\![sum]\!] \leftarrow [\![sum]\!] + \pi_{\mathsf{GR-RANDOM}}(0, 1)$
6: **end for** $[\![\gamma[i]]\!] \leftarrow [\![sum]\!] - 6$
7: **end for**
8: $[\![\hat{\mu}_s]\!] \leftarrow [\![\mu_s]\!] + b.[\![\gamma]\!]$
9: **return** $\hat{\mu}_s$

3.5. Generation of Synthetic Data

S_1 takes the noisy measurement $\hat{\mu}_s$ and runs the estimate algorithm of F to generate $\hat{D}$. S_1 can run this step without the need of MPC protocols as it takes DP input which means that the privacy of D is preserved due to the post-processing property of DP.

3.6. Note on Modularity

The protocols π_{SELECT} and π_{MEASURE} can be replaced by custom MPC protocols for the select and measure steps in one's SDG algorithm of choice. In our work, we specifically design MPC protocols for AIM and MWEM+PGM. Our choice of these algorithms is based on the literature showcasing that these are the state-of-the-art synthetic tabular data generation techniques (McKenna et al., 2022; Pereira et al., 2023).

CaPS is applicable to workload-based synthetic generation algorithms too such as RAP (Vietri et al., 2022) that follow the select-measure-generate template. Our π_{MEASURE} protocol can be used as is for other algorithms such as RAP and MST (McKenna et al., 2021). If the considered distribution setting is only horizontal, π_{MEASURE} can benefit in terms of MPC overhead by employing distributed DP in case of Gaussian noise addition. We also notice that π_{COMP} is equivalent to computing joint histograms from all the data holders. This protocol can be replaced in our framework with other efficient protocols, if available, for the given scheme and datasets distributions (e.g. (Bell et al., 2022; Asharov et al., 2023)).

The implementations of MPC protocols for differentially private mechanisms can also be replaced to satisfy different project requirements. For example, the Gaussian sampling protocol π_{GSS}, which is based on the Box-Muller transform, can be replaced by other methods (Canonne et al., 2020).

4. Experimental Evaluation

Datasets. We evaluate **CaPS** on three datasets: breast-cancer (Zwitter & Soklic, 1988), prison recidivism (COM-

Table 1. **Utility evaluation**. Synthetic data was generated with $\epsilon = 1.0$. For the distributed scenario with **CaPS**, $N = 2$ and $M = 3$ (3PC passive (Araki et al., 2016)). All values are averaged across 3 runs. Abbreviations: CDP = Central Differential Privacy with trusted aggregator (no input privacy); **CaPS** = Our proposed approach for arbitrary distribution; H = Horizontal distribution; V = Vertical distribution; LR = Logistic Regression; RF = Random Forest.

| | | BREAST-CANCER (CATEGORICAL) | | |
		CDP	**CaPS**(H)	**CaPS**(V)
AIM	LR-AUC	0.49	0.49	0.51
	RF-AUC	0.58	0.45	0.53
	LR-F1	0.47	0.43	0.48
	RF-F1	0.54	0.44	0.51
	ERROR Δ	0.30	0.23	0.23
MWEM+PGM	LR-AUC	0.50	0.55	0.44
	RF-AUC	0.51	0.54	0.49
	LR-F1	0.48	0.49	0.43
	RF-F1	0.44	0.46	0.50
	ERROR Δ	0.24	0.21	0.21
		COMPAS (CATEGORICAL)		
		CDP	**CaPS**(H)	**CaPS**(V)
AIM	LR-AUC	0.67	0.66	0.68
	RF-AUC	0.65	0.65	0.67
	LR-F1	0.62	0.62	0.63
	RF-F1	0.61	0.61	0.62
	ERROR Δ	0.017	0.019	0.015
MWEM+PGM	LR-AUC	0.66	0.66	0.66
	RF-AUC	0.62	0.61	0.64
	LR-F1	0.62	0.60	0.61
	RF-F1	0.59	0.58	0.60
	ERROR Δ	0.022	0.022	0.022
		DIABETES (CONTINUOUS)		
		CDP	**CaPS**(H)	**CaPS**(V)
AIM	LR-AUC	0.76	0.77	0.73
	RF-AUC	0.74	0.72	0.71
	LR-F1	0.68	0.65	0.66
	RF-F1	0.66	0.63	0.63
	ERROR Δ	0.14	0.13	0.13
MWEM+PGM	LR-AUC	0.62	0.64	0.67
	RF-AUC	0.57	0.60	0.63
	LR-F1	0.55	0.52	0.60
	RF-F1	0.55	0.56	0.58
	ERROR Δ	0.15	0.14	0.14

PAS)[7] (Angwin et al., 2016), and diabetes (Smith et al., 1988). We randomly split all the datasets into train and test in an 80% to 20% ratio. The breast-cancer dataset has 10 categorical attributes and 285 samples. The COMPAS data consists of categorical data. We utilize the same version as in (Calmon et al., 2017), which consists of 7 categorical features and 7,214 samples. The diabetes dataset has 9 continuous attributes and 768 samples. We use the train sets

[7]https://www.propublica.org/datastore/dat
aset/compas-recidivism-risk-score-data-and
-analysis

to generate synthetic data and the test sets to evaluate the quality of the synthetic data.

Metrics. To assess utility, we train classifiers on the generated synthetic data and test the models on the real test data. For breast-cancer, the task is to predict if the cancer will recur. For COMPAS, the task is to predict whether a criminal defendant will re-offend. For the diabetes dataset, the task is to classify a patient as diabetic. We train logistic regression and random forest models on the generated synthetic datasets and report the AUC-ROC and F1 score. We also evaluate **CaPS** for statistical utility of the generated data as the workload error Δ (McKenna et al., 2022):

$$\Delta(D, \hat{D}) = \frac{1}{|Q|} \sum_{q \in Q} \|\mu_q(D) - \mu_q(\hat{D})\|$$

Evaluation Setup. We implemented the MPC protocols of **CaPS** in MP-SPDZ (Keller, 2020b). We evaluate **CaPS** for horizontal (**CaPS**(H)) and vertical (**CaPS**(V)) data distribution scenarios, and compare against the centralized paradigm (CDP) in which all data holders give their data to a trusted aggregator (i.e. no input privacy). For the horizontal setup, we distribute the samples randomly and evenly among the data holders. For the vertical setup, we distribute the attributes randomly and evenly among the data holders.

Utility Analysis. Table 1 shows that **CaPS** can generate synthetic data whose utility is at par with the CDP in terms of ML utility and statistical utility for both horizontal and vertical partitions. For the COMPAS and diabetes datasets, CDP and **CaPS** give similar ML and statistical utility. We attribute the higher variability observed in the cancer data experiments (AUC and F1) to the small data size. The variation observed in results in Table 1 are due to to randomness resulting from DP.

Table 2. **Runtime evaluation**. Synthetic data was generated with $\epsilon = 1.0$. For the distributed scenario with **CaPS**, $N = 2$ and $M = 3$ (3PC passive (Araki et al., 2016)). For **CaPS** we report runtimes for experiments done in a ***simulated environment***. All values are in seconds and averaged across 3 runs. Abbreviations: CDP = Central Differential Privacy with trusted aggregator (no input privacy); **CaPS** = Our proposed approach for arbitrary distribution; H = Horizontal distribution; V = Vertical distribution.

	$|Q| \times \max(\omega_q)$	CDP	**CaPS**(H)	**CaPS**(V)
	BREAST-CANCER ($n = 228, d = 10$)			
AIM	57×77	98.065	99.789	98.992
MWEM+PGM	45×77	81.187	82.905	100.295
	COMPAS ($n = 4120, d = 9$)			
AIM	45×9	153.383	155.559	301.390
MWEM+PGM	36×9	61.477	61.656	152.153
	DIABETES ($n = 614, d = 9$)			
AIM	45×25	97.222	113.909	97.403
MWEM+PGM	36×25	60.926	63.727	91.760

Performance Analysis. To measure the average time to

generate synthetic data in CDP and **CaPS**, we run experiments in a simulated environment on Azure D8ads_v5 8 vCPUs, 32Gib RAM. We leverage MP-SPDZ to create the simulated environment in a single machine. In Table 2, we observe that, as expected, **CaPS** takes longer than CDP due to MPC overhead. For MWEM+PGM, it takes longer to generate synthetic data in a vertical setup due to overhead of π_{COMP}. For AIM, the vertical setup takes comparatively less time. This is because AIM requires computation of one-way marginals before the selection step, which can be done by the data holders in a vertical distributed scenario. In the horizontal setup, computation of one-way marginals has to be done in MPC, generating additional overhead in computation of marginals compared to the vertical setup. We also note that $|Q| \times \max(\omega_q)$ impacts the runtime for (**CaPS**(H)), whereas $|Q| \times \max(\omega_q)$ and n impact the runtime for (**CaPS**(V)). We believe these runtimes are acceptable since fast response time is not crucial for SDG. Moreover, the benefits of generating synthetic data while preserving both input and output privacy surpass this additional cost.

In Table 3, we evaluate individual MPC subprotocols for different threat models when they are run on independent instances of Azure Standard F16s v2 (16 vcpus, 32 GiB memory) and network bandwidth of 12.5Gbps. The results are in-line with the literature (Keller & Sun, 2022). We observe that 3PC passive provides the least MPC overhead. For t iterations of the loop in Alg. 1, the additional time due to MPC in a 3PC setting is $(2.882 + t \cdot 1.1913)$ sec for AIM and $(2.882 + t \cdot 0.0463)$ sec for MWEM+PGM for the chosen parameters.[8] Given that the one-time additional cost is comparatively much lower than the time it takes to generate synthetic data itself in-the-clear, **CaPS** achieves near-practical performance. With further optimizations of MPC primitives in the future, **CaPS** has the capability to be deployed in real-world scenarios. Our results demonstrate a clear path for future research in this direction and adapting various synthetic data generation techniques smartly in the "DP-in-MPC" paradigm. The major MPC overhead due to the arbitrary distributed setting in **CaPS** is attributed to the computations in π_{COMP}. The performance of this protocol is impacted by the total number of samples n in D. We empirically evaluate the scalability for n and N in Appendix B.1.

5. Related Work

To the best of our knowledge, there are only a handful of works on generating synthetic data from multiple private data sources (Xin et al., 2020a; Behera et al., 2022; Zhao et al., 2023; Ramesh et al., 2023; Ghavamipour et al., 2023; Tajeddine et al., 2020). The work closest to ours is (Maddock et al., 2024) that focuses on loosely-coordinated

---

[8]We note that the number of parties in MPC corresponds to M servers in our framework and not N data holders.

Table 3. **Performance evaluation of MPC protocols for different threat models**. MPC protocols are run with $N = 2, n = 614, d = 9, |Q| = 45, \max(\omega_q) = 25$. We run experiments with $M = 2, 3, 4$ in a LAN setup and the mentioned threat models. We report runtimes in seconds and total communication cost (Comm.) in MB for the online phase of the MPC protocols. $\pi_{SELECT}(A)$ refers to the MPC protocol for 'select' for the AIM algorithm and $\pi_{SELECT}(M)$ for the MWEM+PGM algorithm. π_{COMP} refers to the computation of 1- and 2-way marginals in a vertical distribution.

PROTOCOL	2PC PASSIVE		3PC PASSIVE	
	TIME(S)	COMM.(MB)	TIME(S)	COMM.(MB)
$\pi_{MEASURE}$	0.020	2.214	0.0043	0.261
$\pi_{SELECT}(A)$	2.972	119.053	1.187	0.554
$\pi_{SELECT}(M)$	0.783	167.658	0.042	5.341
π_{COMP}	135.482	30137.40	2.882	696.644

PROTOCOL	3PC ACTIVE		4PC ACTIVE	
	TIME(S)	COMM.(MB)	TIME(S)	COMM.(MB)
$\pi_{MEASURE}$	0.030	2.140	0.0157	0.714
$\pi_{SELECT}(A)$	1.773	11.971	2.169	13.707
$\pi_{SELECT}(M)$	0.306	33.946	0.135	4.700
π_{COMP}	21.612	3830.53	6.00	1615.61

federated settings rather than synchronized horizontal distributed setting. All of these methods either work only for a given data distribution scenario or rely on specialized hardware. In contrast with existing work, we focus on generating synthetic data with the "DP-in-MPC" paradigm for SDG algorithms that follow the select-measure-generate template, in a manner that works for any arbitrary distribution and does not require specialized hardware. In the general modular framework that we have introduced to this end, one can plug in other protocols from the MPC literature, including protocols for sampling noise, see e.g. (Wei et al., 2023; Pettai & Laud, 2015; Zhao et al., 2019).

6. Conclusion

We introduced a general framework **CaPS** that enables collaborative and private generation of tabular synthetic data based on real data from multiple data holders. **CaPS** follows the select-measure-generate template to generate synthetic data regardless of how the real data is distibuted among the data holders, i.e. horizontally, vertically, or mixed. We leverage the idea of "DP-in-MPC" to provide input privacy by performing computations on secret shared data, and applying differential privacy mechanisms within MPC itself. Letting MPC servers emulate a trusted central entity allows us to provide the same level of output privacy and utility as in the centralized paradigm, however without having to sacrifice input privacy. We demonstrated the applicability of **CaPS** for the state-of-the-art marginal based synthetic data generators AIM and MWEM+PGM. We consider generating synthetic data using "DP-in-MPC" for other data modalities and their state-of-the art generation algorithms as a future research direction.

Acknowledgements

Impact statement

Our work advocates for responsible use of sensitive and private data while benefiting different tasks of data science. In our paper, we propose a general framework that generates synthetic data from multiple sources. We believe that synthetic data is a multi-advantageous solution for data sharing while mitigating privacy concerns. Our work contributes to generation of synthetic data based on training data from multiple private data sources, thereby improving the quality and diversity of generated data. We envision a scenario where our framework can be used to generate synthetic data in sensitive domains such as healthcare. Organizations can use the generated synthetic data to augment their local datasets, perform data imputations or train effective and fair ML models. We build upon the existing state-of-the art algorithms to generate synthetic data. Our proposed framework is limited by the disadvantages and affects of these existing algorithms (such as fairness and robustness of generated synthetic data, and vulnerability to adversarial privacy attacks). Users must be aware of such limitations/disadvantages and validate the suitability of the algorithm to generate synthetic data when considering our framework. We additionally note that our framework provides privacy and utility at the cost of runtime and communication, so choosing the framework for time-sensitive scenarios is left to the user's discretion. In our experiments we use publicly available datasets thereby adhering to the ethical guidelines for data usage.

References

Angwin, J., Larson, J., Mattu, S., and Kirchner, L. Machine bias. *ProPublica*, May 23, 2016.

Araki, T., Furukawa, J., Lindell, Y., Nof, A., and Ohara, K. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM SIGSAC Conference on Computer and Communications Security*, pp. 805–817, 2016.

Asharov, G., Hamada, K., Kikuchi, R., Nof, A., Pinkas, B., and Tomida, J. Secure statistical analysis on multiple datasets: Join and group-by. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3298–3312, 2023.

Behera, M. R., Upadhyay, S., Shetty, S., Priyadarshini, S., Patel, P., and Lee, K. F. FedSyn: Synthetic data generation using federated learning. *arXiv preprint arXiv:2203.05931*, 2022.

Bell, J., Gascon, A., Ghazi, B., Kumar, R., Manurangsi, P., Raykova, M., and Schoppmann, P. Distributed, private, sparse histograms in the two-server model. Cryptology ePrint Archive, Paper 2022/920, 2022.

Box, G. E. and Muller, M. E. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, 1958.

Calmon, F. P., Wei, D., Vinzamuri, B., Ramamurthy, K. N., and Varshney, K. R. Optimized pre-processing for discrimination prevention. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 3995–4004, 2017.

Canetti, R. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1): 143–202, 2000.

Canonne, C. L., Kamath, G., and Steinke, T. The discrete gaussian for differential privacy. *Advances in Neural Information Processing Systems*, 33:15676–15688, 2020.

Catrina, O. and Saxena, A. Secure computation with fixed-point numbers. In *14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pp. 35–50. Springer, 2010.

Cramer, R., Damgård, I., and Maurer, U. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 316–334. Springer, 2000.

Cramer, R., Damgård, I., Escudero, D., Scholl, P., and Xing, C. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *Annual International Cryptology Conference*, pp. 769–798. Springer, 2018.

Dalskov, A., Escudero, D., and Keller, M. Secure evaluation of quantized neural networks. arXiv preprint arXiv:1910.12435, 2019.

Dalskov, A., Escudero, D., and Keller, M. Fantastic four: Honest-majority four-party secure computation with malicious security. In *USENIX 2021*, pp. 2183–2200, 2021.

Dwork, C. and Roth, A. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

Evans, D., Kolesnikov, V., and Rosulek, M. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018. URL https://par.nsf.gov/servlets/purl/10099282.

Ghavamipour, A. R., Turkmen, F., Wang, R., and Liang, K. Federated synthetic data generation with stronger security guarantees. In *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, pp. 31–42, 2023.

Hardt, M. and Rothblum, G. N. A multiplicative weights mechanism for privacy-preserving data analysis. In *51st Annual Symposium on Foundations of Computer Science*, pp. 61–70. IEEE, 2010.

Jordon, J., Yoon, J., and Van Der Schaar, M. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

Kairouz, P., Liu, Z., and Steinke, T. The distributed discrete Gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*, pp. 5201–5212. PMLR, 2021a.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021b.

Kasiviswanathan, S. P., Lee, H. K., Nissim, K., Raskhodnikova, S., and Smith, A. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

Keller, M. MP-SPDZ: A versatile framework for multiparty computation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1575–1590, 2020a.

Keller, M. MP-SPDZ: A versatile framework for multiparty computation. Cryptology ePrint Archive, Report 2020/521, 2020b.

Keller, M. and Sun, K. Secure quantized training for deep learning. In *International Conference on Machine Learning*, pp. 10912–10938. PMLR, 2022.

Maddock, S., Cormode, G., and carsten maple. FLAIM: AIM-based synthetic data generation in the federated setting, 2024. URL https://openreview.net/forum?id=8hc2UvwTaL.

Mckenna, R., Sheldon, D., and Miklau, G. Graphical-model based estimation and inference for differential privacy. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 4435–4444. PMLR, 2019.

McKenna, R., Miklau, G., and Sheldon, D. Winning the NIST contest: A scalable and general approach to differentially private synthetic data. *arXiv preprint arXiv:2108.04978*, 2021.

McKenna, R., Mullins, B., Sheldon, D., and Miklau, G. AIM: an adaptive and iterative mechanism for differentially private synthetic data. *Proceedings of the VLDB Endowment*, 15(11):2599–2612, 2022.

Mironov, I. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 650–661, 2012.

Pasquini, D., Raynal, M., and Troncoso, C. On the (in)security of peer-to-peer decentralized machine learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 418–436, 2023.

Pentyala, S., Dowsley, R., and De Cock, M. Privacy-preserving video classification with convolutional neural networks. In *International Conference on Machine Learning*, pp. 8487–8499. PMLR, 2021.

Pereira, M., Kshirsagar, M., Mukherjee, S., Dodhia, R., Ferres, J. L., and de Sousa, R. Assessment of differentially private synthetic data for utility and fairness in end-to-end machine learning pipelines for tabular data. *arXiv preprint arXiv:2310.19250*, 2023.

Pettai, M. and Laud, P. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pp. 421–430, 2015.

Ramesh, V., Zhao, R., and Goel, N. Decentralised, scalable and privacy-preserving synthetic data generation. *arXiv preprint arXiv:2310.20062*, 2023.

Smith, J. W., Everhart, J. E., Dickson, W., Knowler, W. C., and Johannes, R. S. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Applications in Medical Care*, pp. 261–265. American Medical Informatics Association, 1988.

Tajeddine, R., Jälkö, J., Kaski, S., and Honkela, A. Privacy-preserving data sharing on vertically partitioned data. *arXiv preprint arXiv:2010.09293*, 2020.

Tao, Y., McKenna, R., Hay, M., Machanavajjhala, A., and Miklau, G. Benchmarking differentially private synthetic data generation algorithms. In *AAAI-22 Workshop on Privacy-Preserving Artificial Intelligence*, 2022.

Vietri, G., Archambeau, C., Aydore, S., Brown, W., Kearns, M., Roth, A., Siva, A., Tang, S., and Wu, S. Z. Private synthetic data for multitask learning and marginal queries. *Advances in Neural Information Processing Systems*, 35: 18282–18295, 2022.

Wei, C., Yu, R., Fan, Y., Chen, W., and Wang, T. Securely sampling discrete Gaussian noise for multi-party differential privacy. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2262–2276, 2023.

Xin, B., Yang, W., Geng, Y., Chen, S., Wang, S., and Huang, L. Private FL-GAN: Differential privacy synthetic data generation based on federated learning. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2927–2931, 2020a.

Xin, B., Yang, W., Geng, Y., Chen, S., Wang, S., and Huang, L. Private FL-GAN: Differential privacy synthetic data generation based on federated learning. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2927–2931. IEEE, 2020b.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, J., Cormode, G., Procopiuc, C. M., Srivastava, D., and Xiao, X. PrivBayes: Private data release via Bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

Zhao, R., Goel, N., Agrawal, N., Zhao, J., Stein, J., Verborgh, R., Binns, R., Berners-Lee, T., and Shadbolt, N. Libertas: Privacy-preserving computation for decentralised personal data stores. *arXiv preprint arXiv:2309.16365*, 2023.

Zhao, R. K., Steinfeld, R., and Sakzad, A. FACCT: fast, compact, and constant-time discrete Gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1): 126–137, 2019.

Zwitter, M. and Soklic, M. Breast Cancer. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C51P4M.

A. MPC protocols.

MPC primitives. The MPC schemes listed in Section 2 provide a mechanism for the servers to perform cryptographic primitives through the use of secret shares, namely addition of a constant, multiplication by a constant, and addition of secret shared values, and multiplication of secret shared values (denoted as π_{MUL}). Building on these cryptographic primitives, MPC protocols for other operations have been developed in the literature. We use (Keller, 2020a):

- Secure random number generation from uniform distribution $\pi_{\mathsf{GR-RANDOM}}$: In $\pi_{\mathsf{GR-RANDOM}}$, each party generates l random bits, where l is the fractional precision of the power 2 ring representation of real numbers, and then the parties define the bitwise XOR of these l bits as the binary representation of the random number jointly generated.
- Secure random bit generation $\pi_{\mathsf{GR-RNDM-BIT}}$: In $\pi_{\mathsf{GR-RNDM-BIT}}$, each party generates the secret share of a single random bit, such that the generated bit is either 0 or 1 with a probability of 0.5.
- Secure multiplication π_{MUL} : At the start of this protocol, the parties have secret sharings $[\![a]\!]$ and $[\![b]\!]$; at the end, then they have a secret share of $c = a.b$.
- Secure equality test π_{EQ} : At the start of this protocol, the parties have secret sharings $[\![a]\!]$; at the end if $a = 0$, then they have a secret share of 1, else a secret sharing of 0.
- Secure less than test π_{LT} : At the start of this protocol, the parties have secret sharings $[\![a]\!]$ and $[\![b]\!]$ of integers a and a; at the end of the protocol they have a secret sharing of 1 if $a < b$, and a secret sharing of 0 otherwise.
- Secure greater than test π_{GT} : At the start of this protocol, the parties have secret sharings $[\![a]\!]$ and $[\![b]\!]$ of integers a and b; at the end of the protocol they have a secret sharing of 1 if $a > b$, and a secret sharing of 0 otherwise.
- Secure greater than test π_{GTE} : At the start of this protocol, the parties have secret sharings $[\![a]\!]$ and $[\![b]\!]$ of integers a and b; at the end of the protocol they have a secret sharing of 1 if $a \geq b$, and a secret sharing of 0 otherwise.
- Other primitives : We use secure maximum protocol (π_{MAX}), secure exponential protocol (π_{EXP}) and secure logarithm protocol (π_{LN}) as the building blocks for our protocols. π_{LN} uses the polynomial expansion for computing logarithm and π_{EXP} in turn uses the π_{LN} to compute exponential. π_{MAX} inherently uses the π_{GT} repeatedly over a list by employing variant of Divide-n-Conquer approach. At the start of all of these primitives, parties hold the secret sharings $[\![x]\!]$ and at the end of the protocol they hold the secret shares of the corresponding computed values.
- Other generic subprotocols : π_{ERR} computes the secret shares of error between secrets shares of two vectors. π_{NORM} normalizes, either by scaling or computing L1 norm, the secret shares of the error vector. π_{PROB} computes the probabilities for a given set of secret shares of vectors. π_{SQRT} computes secret shares of the square root of $[\![x]\!]$, π_{SIN} computes secret shares of the sine of $[\![x]\!]$, π_{COS} computes secret shares of the cosine of $[\![x]\!]$. π_{SUM} computes the sum of secret shares of a given vector. π_{SOFTMAX} computes the softmax for a given vector. (See (Keller, 2020b))

MPC protocols can be mathematically proven to guarantee privacy and correctness. We follow the universal composition theorem that allows modular design where the protocols remain secure even if composed with other or the same MPC protocols (Canetti, 2000).

Description of π_{SELECT} for AIM. Protocol 6 is the straight forward implementation of the select step from the centralized algorithm of AIM. Lines 2–7 compute the secrets shares of errors between the answers from real and synthetic data and form the subprotocol π_{ERR}. This protocol relies on $\pi_{\mathsf{L1-NORM}}1$ to compute the L1-norm of the errors. Protocol 7 computes secret shares of L1-norm of the error vector. This protocol takes the input size m which is the length of the answer for for query q_i, i.e. m $= \omega_{q_i}$ (this is an implementation details where we consider the unpadded vector answer). Lines 9–12 normalize the secret shares of errors by scaling it with secret share of maximum error computed using primitive π_{MAX}. Lines 9–12 form the subprotocol for normalize π_{NORM}. Line 14 computes secret shares of the probabilities by relying on the subprotocol π_{SOFTMAX}. This is defined as π_{PROB} for AIM. Finally on Line 16, MPC protocol for exponential mechanism is called π_{RC} that outputs the selected query based on computed secret shares of the probabilities.

Description of π_{SELECT} for MWEM+PGM. Protocol 8 is the straight forward implementation of the select step from the centralized algorithm of MWEM+PGM. Lines 2–9 compute the secrets shares of errors between the answers from real and synthetic data and form the subprotocol π_{ERR}. Lines 11–14 normalize the secret shares of errors by scaling it with secret share of maximum error computed using primitive π_{MAX}. Lines 11–14 form the subprotocol for normalize π_{NORM}. Line 16 computes secret shares of the probabilities by relying on the subprotocol π_{SOFTMAX}. This is defined as π_{PROB} for MWEM+PGM. Finally on Line 18, MPC protocol for exponential mechanism is called π_{RC} that outputs the selected query based on computed secret shares of the probabilities.

Description of π_{MEASURE} using Box-Muller Method. Lines 3–6 in Protocol 5 can be replaced by the subprotocol 9 which relies on the transform by (Box & Muller, 1958) to generate samples of the Gaussian unitary distribution, namely $\lceil \omega_s/2 \rceil$ pairs of Gaussian samples. For each pair, the MPC servers securely generate secret shares of two random number u and v uniformly distributed in [0,1] using MPC primitive $\pi_{\mathsf{GR-RANDOM}}$ on Lines 3–4. On Lines 5–8, the MPC servers then

---

Protocol 6 π_{SELECT}: MPC Protocol for **SELECT** for AIM

Input: Secrets shares of $[\![\mu_q]\!]$, Estimated answer $\hat{\mu}_q$, length of domains ω_q for each q, privacy parameters ϵ, a vector bias b for each q, weight for queries w , max_sensitivity s
Output: Selected query q_s
1: ********* Compute errors - π_{ERR} *********
2: Initialize a vector err of length $max(\omega_q)$
3: **for** $i = 1$ to $|Q|$ **do**
4:      $[\![\text{diff}]\!] \leftarrow [\![\mu_{q_i}]\!] - \hat{\mu}_{q_i}$
5:      $[\![err[i]]\!] \leftarrow \pi_{\mathsf{L1-NORM}}1([\![\text{diff}]\!], \omega_{q_i})$// computes L1 norm of the errors
6:      $[\![err[i]]\!] \leftarrow \pi_{\mathsf{MUL}}(w[i], [\![err[i]]\!] - b[i])$
7: **end for**
8: ********* Normalize errors - π_{NORM} *********
9: $[\![\text{max\_err}]\!] \leftarrow \pi_{\mathsf{MAX}}([\![err]\!])$
10: **for** $i = 1$ to $|Q|$ **do**
11:      $[\![err[i]]\!] \leftarrow [\![err[i]]\!] - [\![\text{max\_err}]\!]$
12: **end for**
13: ********* Compute probabilities - π_{PROB} *********
14: $[\![prob]\!] \leftarrow \pi_{\mathsf{SOFTMAX}}(0.5 \cdot \epsilon \cdot (1/s) \cdot [\![err]\!])$
15: ********* Select random index - π_{RC} *********
16: $[\![s]\!] \leftarrow \pi_{\mathsf{RC}}([\![prob]\!])$
17: Reveal s
18: **return** q_s

Protocol 7 $\pi_{\mathsf{L1-NORM}}$: MPC Protocol to compute L1-norm

---

Input: Input vector $[\![\text{diff}]\!]$, length of vector m
Output: Normalized vector $[\![err[i]]\!]$
1: $\text{sum} \leftarrow 0$
2: **for** $i = 1$ to m **do**
3:      $[\![\text{sign}]\!] \leftarrow \pi_{\mathsf{LT}}([\![\text{diff}[i]]\!], 0)$
4:      $[\![\text{abs\_diff}]\!] \leftarrow \pi_{\mathsf{MUL}}(1 - 2 \cdot [\![\text{sign}]\!], [\![\text{diff}[i]]\!])$
5:      $[\![\text{sum}]\!] \leftarrow [\![\text{sum}]\!] + [\![\text{abs\_diff}]\!]$
6: **end for**
7: **return** $[\![\text{sum}]\!]$

Protocol 8 π_{SELECT}: MPC Protocol for **SELECT** for MWEM+PGM

Input: Secrets shares of $[\![\mu_q]\!]$, Estimated answer $\hat{\mu}_q$, length of domains ω_q for each q, privacy parameters ϵ
Output: Selected query q_s
1: ********* Compute errors - π_{ERR} *********
2: Initialize a vector err of length $max(\omega_q)$
3: **for** $i = 1$ to $|Q|$ **do**
4:      $[\![\text{diff}]\!] \leftarrow [\![\mu_{q_i}]\!] - \hat{\mu}_{q_i}$
5:      $[\![\text{sign}]\!] \leftarrow \pi_{\mathsf{LT}}([\![\text{diff}]\!], 0)$
6:      $[\![\text{abs\_diff}]\!] \leftarrow \pi_{\mathsf{MUL}}(1 - 2 \cdot [\![\text{sign}]\!], [\![\text{diff}]\!])$ // computes absolute difference of error between answers on real data and synthetic data
7:      $[\![\text{sum}]\!] \leftarrow \pi_{\mathsf{SUM}}(1 - 2 \cdot [\![\text{abs\_diff}]\!])$
8:      $[\![err[i]]\!] \leftarrow [\![\text{sum}]\!] - \omega_{q_i}$
9: **end for**
10: ********* Normalize errors - π_{NORM} *********
11: $[\![\text{max\_err}]\!] \leftarrow \pi_{\mathsf{MAX}}([\![err]\!])$
12: **for** $i = 1$ to $|Q|$ **do**
13:      $[\![err[i]]\!] \leftarrow [\![err[i]]\!] - [\![\text{max\_err}]\!]$
14: **end for**
15: ********* Compute probabilities - π_{PROB} *********
16: $[\![prob]\!] \leftarrow \pi_{\mathsf{SOFTMAX}}(0.5 \cdot \epsilon \cdot [\![err]\!])$
17: ********* Select random index - π_{RC} *********
18: $[\![s]\!] \leftarrow \pi_{\mathsf{RC}}([\![prob]\!])$
19: Reveal s
20: **return** q_s

---

compute a secret sharing of $\sqrt{-2\ln(u)} \cdot \cos(2\pi v)$ and of $\sqrt{-2\ln(u)} \cdot \sin(2\pi v)$ using MPC protocols for π_{SQRT}, π_{SIN}, π_{COS}, and π_{LN} ((Keller, 2020a)).In case d is odd, one more sample needs to be generated. The parties do so on Lines 11–12 by executing π_{GSS} to sample a vector of length 2 and only retain the first coordinate.

Protocol 9 π_{GSS}: Box-Muller to generate Guassian sample with mean 0 and variance 1

Input: Vector length ω_s.
Output: A secret-shared vector $[\![\gamma]\!]$ of length ω_s sampled from Gaussian distribution with mean 0 and variance 1

1: Initialize vector $[\![\gamma]\!]$ of length d
2: **for** $i = 0$ **to** $\omega_s/2$ **do**
3:    $[\![u]\!] \leftarrow \pi_{\mathsf{GR-RANDOM}}(0,1)$
4:    $[\![v]\!] \leftarrow \pi_{\mathsf{GR-RANDOM}}(0,1)$
5:    $[\![r]\!] \leftarrow \pi_{\mathsf{SQRT}}(-2\pi_{\mathsf{LN}}([\![u]\!]))$
6:    $[\![\theta]\!] \leftarrow 2\pi[\![v]\!]$
7:    $[\![s_{2i}]\!] \leftarrow \pi_{\mathsf{MUL}}([\![r]\!], \pi_{\mathsf{COS}}([\![\theta]\!]))$
8:    $[\![x_{2i+1}]\!] \leftarrow \pi_{\mathsf{MUL}}([\![r]\!], \pi_{\mathsf{SIN}}([\![\theta]\!]))$
9: **end for**
10: **if** d is odd **then**
11:    $[\![p]\!] \leftarrow \pi_{\mathsf{GSS}}(2)$
12:    $[\![s_{d-1}]\!] \leftarrow [\![p_0]\!]$
13: **end if**
14: **return** $[\![\gamma]\!]$

Description of π_{SELECT} with Laplacian noise generation. π_{MEASURE} can be replaced by Protocol 10 when one needs to add Laplacian noise. The noise is sampled as $b \cdot \ln l \cdot$ c where b is the scale, l is a random value drawn from the uniform distribution in [0,1] and c is a random value selected from $\{-1, 1\}$. On Lines 3–4, the MPC servers straightforwardly compute l and its natural log. To compute c, the parties, on line 5, generate secret shares of a random bit $[\![r]\!]$, i.e. a value $\in \{0, 1\}$ is chosen, where each value has a chance of 50% to be chosen. On line 6, the parties transform r to a value $\in \{-1, 1\}$ using the logic c = 2$\cdot$ r -1. Lines 7–10 is straightforward computation of the noise vector $[\![\gamma[i]]\!]$ and noisy measurement $[\![\hat{\mu}_s]\!]$.

One can also replace the lines 3–6 by the following MPC pseudocode. This pseudocode drawns Laplace noise based on $l = -b \cdot \mathrm{sgn}(u) \cdot \ln(1 - 2|u|)$, where u is a random variable drawn from a uniform distribution in $[-0.5, 0.5]$. l has distribution $Lap(0, b)$. This follows from the inverse cumulative distribution function for $Lap(0, b)$.

$[\![u]\!] \leftarrow \pi_{\mathsf{GR-RANDOM}}(-0.5, 0.5)$
$[\![\mathrm{sgn}_u]\!] \leftarrow \pi_{\mathsf{GTE}}([\![u]\!], 0)$
$[\![\mathrm{abs}_u]\!] \leftarrow \pi_{\mathsf{MUL}}([\![u]\!], [\![\mathrm{sgn}_u]\!]))$
$[\![\ln_u]\!] \leftarrow \pi_{\mathsf{LN}}(1 - 2 \cdot [\![\mathrm{abs}_u]\!])$
$[\![l]\!] \leftarrow -1 \cdot \pi_{\mathsf{MUL}}([\![\ln_u]\!], [\![\mathrm{sgn}_u]\!])$
$[\![\gamma[i]]\!] \leftarrow [\![l]\!]$

Protocol 10 π_{SELECT}: MPC Protocol for **MEASURE** using Laplacian noise

---

Input: Secrets shares of $[\![\mu_{q_s}]\!]$, length of domain ω_s for q_s, scale b
Output: Noisy measurement $\hat{\mu}_s$

1: Initialize vector $[\![\gamma]\!]$ of length $\max(\omega_r)$ with 0s
2: **for** i = 0 **to** ω_{q_s} **do**
3:    $[\![l]\!] \leftarrow \pi_{\mathsf{GR-RANDOM}}(0,1)$ // with protocol for random number generation $\pi_{\mathsf{GR-RANDOM}}$
4:    $[\![\ln\_l]\!] \leftarrow \pi_{\mathsf{LN}}([\![l]\!])$ // with secure logarithm protocol π_{LN}
5:    $[\![r]\!] \leftarrow \pi_{\mathsf{GR-RNDM-BIT}}()$ // with protocol for random bit generation $\pi_{\mathsf{GR-RANDOM}}$
6:    $[\![c]\!] \leftarrow 2 \cdot [\![r]\!] - 1$
7:    $[\![\gamma[i]]\!] \leftarrow \pi_{\mathsf{MUL}}([\![\ln\_l]\!], [\![c]\!])$ // with secure multiplication protocol π_{MUL}
8: **end for**
9: $[\![\hat{\mu}_s]\!] \leftarrow [\![\mu_s]\!] + b.[\![\gamma]\!]$
10: **return** $\hat{\mu}_s$

Extending Protocol 2 to compute p-way marginals. MPC protocols are typically designed as specific circuits, composed of a sequence of addition and multiplication gates. Therefore, changing the functionality to be implemented in MPC usually requires the design of a new circuit. Protocol 2 is specifically designed to suit algorithms that consider 1-way and 2-way marginals. Our framework enables the design of new MPC protocols or the utilization of existing ones for complex scenarios, facilitating the generation of tabular synthetic data for arbitrary distributed data.

One can extend the Protocol 2 to compute 3-way marginals in the following ways:

- A direct extension of Protocol 2 to compute p-way marginals is possible. Say the p marginals are represented by a set

$\mathcal{M}$. This requires p conditionals instead of 2 on Line 13 of Protocol 2. Instead of 2 equality checks on Line 14, this requires p equality checks. This means a total of $p \cdot n \cdot \prod_{i \in \mathcal{M}} \omega_i$ equality checks and $p \cdot \prod_{i \in \mathcal{M}} \omega_i$ multiplications. As stated on Line 328, π_{COMP} can be replaced by other protocols such as (Bell et al., 2022).

For $m = 3$, change $q^* = \{a_1, a_2, a_3\}$ and add $[\![z]\!] \leftarrow [\![D_{a_3}[i]]\!]$ after Line 12 on Protocol 2. Change Line 13 to include $l \in \Omega_{a_3}$ and add a multiplication operation with $\pi_{\mathsf{EQ}}([\![z]\!], l$ on Line 14. Line 15 should be modified to index accordingly. Please see Protocol 11.

We will add a discussion on extending protocol 2 to compute of m-way marginals in naive manner. We will state that optimizing the protocol for different scenarios is considered as future work.

- Another possible way to compute 3-way marginals is by repeated computations of 2-way marginals. We compute for $\{a_1, a_2\}$ first and then get a column with the concatenated values for the feature $a_1 || a_2$ and then compute 2-way marginal for $\{a_1 || a_2, a_3\}$. Depending the domain size of each column, this might perform better or worse than the above method.

Protocol 11 MPC Protocol to compute p-way marginals

Input: Secret shares of data $[\![D_i]\!]$ (needed if not horizontal), Queries Q, including queries Q^* with attributes that are distributed among data holders, Domain Ω
Output: $[\![\mu_q]\!], \forall q \in Q$
1: **if** $Q^* \neq \emptyset$ **then**
2:     $[\![D]\!] \leftarrow \pi_{\mathsf{JOIN}}([\![D_i]\!]|i = 1 \ldots N)$
3: **end if**
4: **for all** $q^* = \{a_1, a_2, \ldots, a_p\}$ in Q^* **do**
5:     **for** $i = 1$ to n **do**
6:         $[\![x]\!] \leftarrow [\![D_{a_1}[i]]\!]; [\![y]\!] \leftarrow [\![D_{a_2}[i]]\!]; \ldots [\![z]\!] \leftarrow [\![D_{a_p}[i]]\!]$
7:         **for** $j \in \Omega_{a_1}$ and $k \in \Omega_{a_2} \cdots$ and $l \in \Omega_{a_p}$ **do**
8:             $[\![m]\!] \leftarrow \pi_{\mathsf{MUL}}(\pi_{\mathsf{EQ}}([\![x]\!], j), \pi_{\mathsf{EQ}}([\![y]\!], k), \cdots, \pi_{\mathsf{EQ}}([\![z]\!], l))$
9:             index $\leftarrow$ compute the index for marginal vector
10:            $[\![\mu_{q^*}[\text{index}]]\!] \leftarrow [\![\mu_{q^*}]\!] + [\![m]\!]$
11:         **end for**
12:     **end for**
13: **end for**
14: **return** $[\![\mu_q]\!], \forall q \in Q$

Design and optimization of every possible MPC protocols for all scenarios of select-measure-generate paradigm is beyond the scope of our work. We note that our work provides an important baseline framework that can be adapted to particular algorithms.

Discussion on common datasets. Our framework, in principle, works for all the distributed settings due to the modularity offered. This includes scenarios where the data holders hold common data as shown in Table 4-5. To illustrate the how MPC can be leveraged, Protocol 2 for π_{COMP} considers disjoint dataset. π_{COMP} also works with very little modifications when the data is common across some of the data holders as shown in Table 4-5 for two data holders. In such case, the computations in π_{COMP} begin with executing π_{JOIN} that results in a union of all datasets. This is followed by computation of all the marginals in MPC (such as computations done on Lines 9–18 of Protocol 2 to compute 2-way marginals). This requires removing Lines 1–6 in Protocol 2 and having $Q^* = Q$. This will of course result in change in the number of computations performed in MPC.

Table 4. Data held by H_1

ID	a	b
Alice	0	1
Bob	1	1

Table 5. Data held by H_2

ID	a	b	c
Alice	0	1	2
Charlie	1	0	2

Implementing DP in MPC. Keeping in mind the dangers of implementing DP with floating point arithmetic (Mironov, 2012), we stick with the best practice of using fixed-point and integer arithmetic as recommended by, for example, OpenDP [9]. We implement all our DP mechanism using their discrete representations and use 32 bit precision to ensure correctness.
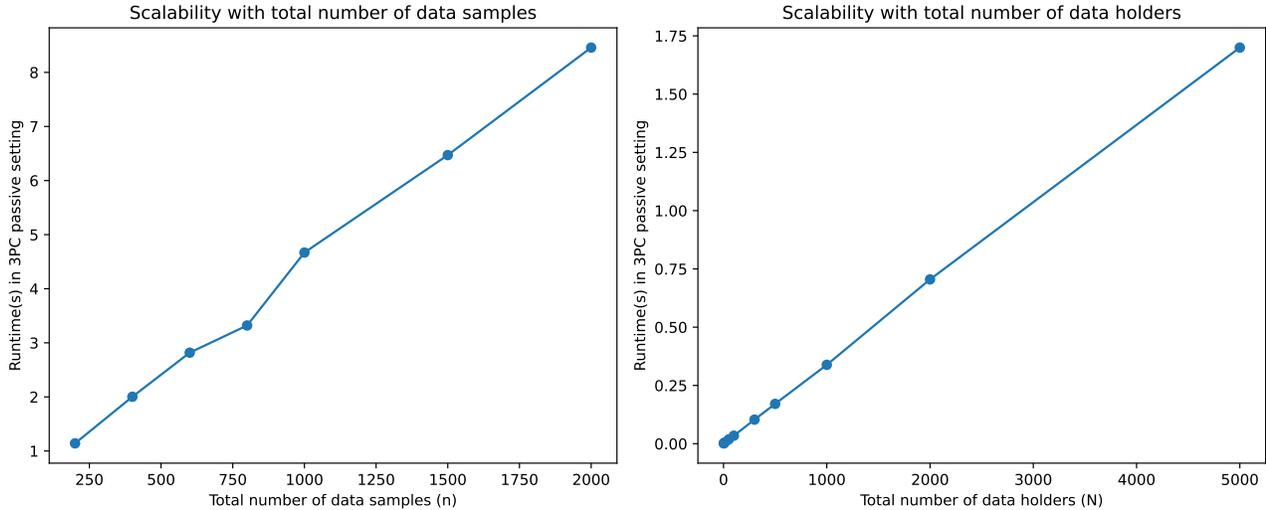
---

[9]https://opendp.org/

Figure 2. **Scalability of** π_{COMP} **in a 3PC passive setting**. MPC protocols are run with $M = 3, d = 10, |Q| = 36, max(\omega_q) = 25$. On left: Scalability of π_{COMP} for different number of total dataset size n. On right: Scalability of π_{COMP} for different number of data holders N.

We also remark that finite precision issues can also impact the exponential mechanism. If the utility of one of the classes in the exponential mechanism collapses to zero after the mapping into finite precision, pure DP becomes impossible to achieve. We can deal with such situation by using approximate DP for an appropriate value of δ. Such situation did not happen with the data sets used in our experiments.

B. Additional experiments.

B.1. Performance analysis

We run additional experiments to evaluate the scalability of `CaPS`.

Scalability with total training samples n We run experiments in a 3PC passive setting and evaluate the scalability of π_{COMP} in Figure 2. This is the only protocol that depends on the value of total number of samples over all the distributed datasets. We note that we run experiments for π_{COMP} which computes 1-way and 2-way marginals in a distributed setting. The overhead due to the number of data holders (requires only additions) in this case is negligible when compared to computation of 2-way marginal for large n. This means that our findings are independent of the number of data holders N.

Scalability with data holders N To illustrate the affect of increase in number of data holders, we consider a horizontal distributed scenario where each data holder holds equal number of samples. The overhead in this case is due to only addition of workload answers from N data holders for a given $|Q|$ and ω.

We note that the scalability of other protocols π_{SELECT} and π_{MEASURE} depend on $|Q|$ and ω. We think the impact due to $|Q|$ and ω is similar to the impact on centralized algorithms. Based on literature, runtimes for MPC protocols grow with large M, depending on the MPC scheme available for M. `CaPS` adapts MPC-as-a-service scenario, where we can choose the optimal number of MPC server, M. We think based on the current literature in MPC $M = 3$ is a better option for `CaPS`.

C. Code.

We have made the code available at https://github.com/sikhapentyala/MPC_SDG/tree/icml