D3.4 Differential Privacy and Leakage Control

Prastudy Fauzi(AU), Claudio Orlandi(AU), Peter Scholl(AU), Kimberley Thissen(PHI / TUE), Berry Schoenmakers(TUE), Peter van Liesdonk(PHI), Meilof Veeningen(PHI), Paul Koster(PHI)

The project SODA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583.

Project Information

Scalable Oblivious Data Analytics

Project number:	731583
Strategic objective:	H2020-ICT-2016-1
Starting date:	2017-01-01
Ending date:	2019-12-31
Website:	https://www.soda-project.eu/

Document Information

Title: D3.4 Differential Privacy and Leakage Control

ID:	D3.4	Type:	R	Dissemination level:	PU
		Month:	M30	Release date:	30.June.2019

Contributors, Editor & Reviewer Information

Contributors (person(partner): sections)	Prastudy Fauzi(AU): Section 4 Claudio Orlandi(AU): Section 1 Peter Scholl(AU): Section 4 Kimberley Thissen(PHI / TUE): Section 2 Berry Schoenmakers(TUE): Section 2 Peter van Liesdonk(PHI): Section 2 Meilof Veeningen(PHI): Section 3 Paul Koster(PHI): Section 2, 3
Editor (person/partner)	Peter Scholl (AU)
Reviewer (person/partner)	Jonas Lindstrøm (ALX)

Release number	Date issued	Release description / changes made
1.0	June 28, 2019	First release to EU

SODA Consortium

Full Name	Abbreviated Name	Country
Philips Electronics Nederland B.V.	PHI	Netherlands
Alexandra Institute	ALX	Denmark
Aarhus University	AU	Denmark
Göttingen University	GU	Germany
Eindhoven University of Technology	TUE	Netherlands

Table 1: Consortium Members

Executive summary

This deliverable presents research on differential privacy and leakage control in the context of secure multi-party computation (MPC), which was carried out by SODA members as part of Work Package 3. The main goal of this research is to obtain a better understanding of information leakage that can occur as a result of taking part in a secure computation, and investigate practical techniques that can be used to mitigate this.

Typically, when we think about solving a problem using MPC, we assume that there is some function f that a set of parties have agreed to compute on their respective inputs, $x_1, \ldots, x_n$. We then proceed by selecting a suitable MPC protocol for solving this task, ideally guaranteeing that if all goes well, no malicious party will be able to deduce anything about the other parties' inputs, beyond what could be learnt from the output of f.

In this deliverable, we take a step back and ask the question,

> *In what settings* should *the parties compute a function $f(x_1, \ldots, x_n)$, and what else might be leaked from this function?*

Note that this is unrelated to the traditional security properties of the MPC protocol computing f, which implicitly assume that computing f is wanted by everyone. In some sense, the question as to whether this is indeed desirable is orthogonal to the design of MPC protocols, however, when looking at the bigger picture of using MPC to solve real-world privacy problems, it is essential to consider both of these aspects.

A natural topic for exploring this question is the field of *differential privacy*, which allows obtaining provable guarantees on the privacy of an individual's data, when considered as part of a larger dataset on which queries are being observed. This fits ideally into an MPC application where participants' private data is being computed upon. However, differential privacy does come with some drawbacks: it typically requires *perturbing* the output of the function f by adding *noise*, which can reduce the usefulness of the result in many settings. Therefore, in addition to differential privacy, in this deliverable we also investigate broader issues related to *analyzing and controlling leakage* in the context of outputs from a secure computation protocol. This can be related to both the types of secure computation protocols being run, and the decision to take part in such a protocol in the first place.

We now give a brief summary of the results in this document.

Balancing Leakage and Utility in Secure Multi-Party Computation. In order to decide whether it is worthwhile to perform MPC and potentially reveal information about their valuable inputs, we can look at the parties' *incentives* for taking part in the computation, based on their perceived *value* of the information gained when learning the result of the computation. To do this, we devise a formal framework for analyzing and reasoning about the amount of information that is gained (or lost) when taking part in a joint computation. We do this using mathematical models based on game theory and mechanism design.

The basic idea behind the framework is as follows. In most use cases of MPC, the *value* in any given party's output can be determined based on the overall quality of all parties' inputs. Therefore, a party can choose to provide a lower quality input (for example, one that is less fine-grained, though still truthful) to reduce the quality of the other parties' outputs, and hence, also reduce leakage on the exact input. Given a model for estimating these quantities, we can define a utility function for the parties, which balances their wish to learn with their desire for exclusivity, from not giving away too much information. We then apply this framework to several practical MPC problems of interest,

including private set intersection, set union and average. For all these tasks, we design mechanisms that incentivize parties to submit high quality inputs whilst maximizing their overall utility.

Combining Differential Privacy and MPC. To enable secure computation systems to support differentially private data analytics, we present a set of tools for combining differential privacy with MPC. We begin by designing efficient protocols for key mathematical operators like logarithms, exponentials, sine and cosine using MPC protocols that support fixed-point arithmetic. We design these protocols based on polynomial approximations to the functions, which requires taking some care in choosing and analyzing the best approximations to obtain a good level of accuracy and performance. We then turn to the task of securely generating randomness for use in a differentially private mechanism. Here, we show how to sample from the Laplace, geometric and Gaussian distributions with inverse sampling methods and a source of random bits. All of the algorithms we present have been implemented and tested in the MPyC framework.

Differentially Private Logistic Regression. As an application of our techniques for noise sampling, we consider the case of training a logistic regression classifier using multiple sensitive datasets from mutually distrusting data providers. There have been several previous works that show how to securely train logistic regression models in MPC, however, these all fall short in at least one area: they are either not differentially private, prohibitively expensive due to the type of MPC operations required, or they leak some undesirable information on intermediate values during the computation.

In this work, we take an approach which uses a hybrid of private and public computations to avoid these drawbacks. By carefully revealing intermediate values in a *differentially private* way, we can perform much of the computation in the clear, avoiding inefficiencies of the pure MPC approach. Moreover, thanks to our use of differential privacy, we do this without any loss in security, unlike previous heuristic approaches.

Controlling Leakage in MPC on Non-Integer Types. In this section, we revisit secure computation on non-integer types such as fixed and floating point values. We show that existing proposals in use do not satisfy classical security definitions, i.e., they all fail to achieve even the "input indistinguishability" notion of security. While some of the observations here might be considered "folklore", we believe that it is important to have them collected in a single resource, especially due to the increased interest in privacy-preserving machine learning techniques (which naturally uses non-integer types). Furthermore, we propose a new security definition inspired by the work of Feigenbaum et al. [20] on MPC for approximation algorithms, and analyze different approaches to constructing protocols that satisfy this definition.

About this Document

Role of the deliverable

This deliverable presents work that shows how to prevent leakage of undesired sensitive information through processing results in a big data analytics scenario. This is one of the core objectives of the SODA project, and is part of Work Package 3 in SODA. This work also provides an important step towards the final project milestone of developing demonstrators for the project's use cases, which is being carried out in Work Package 4.

More precisely, the deliverable addresses this through research in the topic of "leakage control", that is, techniques for analyzing and controlling private data leakage that can arise in a big data analytics system. These techniques include differential privacy, and how to combine it with secure multi-party computation in a private and efficient way that is suitable for the use cases of interest to SODA.

Relationship to other SODA deliverables

The starting point for the work in this deliverable is the state-of-the-art report by SODA members from deliverable D1.1, which surveyed techniques for leakage control and identified important areas for study. Many of the techniques in this deliverable such as the differentially private mechanisms for MPC and new protocol for logistic regression, have been implemented and will be applied to the use cases in the SODA project. These will be reported in deliverable D4.5 as part of Work Package 4.

Structure of this document

In Section 1, we present our framework for analyzing incentives when sharing information with competitors, and give applications to various settings of MPC. In Section 2, we describe important building blocks needed for sampling differentially private noise distributions in MPC, and apply these to several distributions of interest. Section 3 describes our study of performing differentially private logistic regression in secure computation. Finally, in Section 4, we present our analysis of existing frameworks for MPC on non-integer types, and investigate possible solutions to the leakage that occurs in these protocols.

Table of Contents

1 Balancing Leakage and Utility in Secure Multi-Party Computation

This chapter contains a summary of the results of the work performed by SODA researcher Claudio Orlandi (Aarhus University) in collaboration with Simina Brânzei (Purdue University) and Guang Yang (Chinese Academy of Sciences) which is currently under submission under the title "Sharing Information with Competitors" [7].[1]

The goal of this research is to develop a formal mathematical model in which it is possible to reason about the amount of information that parties gain (or lose) when participating in secure multi-party computation protocols. This is formalized using tools from game-theory and mechanism design.

As opposed to the regular game theoretic settings, in which parties are rewarded with money, we here consider the mechanism design problem in the setting where agents are rewarded using information only, which naturally fits the way a secure multi-party computation protocol is used. Specifically, we consider the setting of a joint computation where different agents have inputs of different quality and each agent is interested in learning as much as possible while maintaining exclusivity for information. Our high level question is how to design mechanisms that motivate all the agents (even those with high-quality inputs) to participate in the computation; we formally study natural problems of interest for MPC such as set union, intersection, and average.

1.1 Introduction

Despite the large body of research on MPC, one question that has not been addressed in the cryptographic community so far is whether parties will have any *incentive* in participating in such protocols: In traditional multi-party computation tasks, multiple agents wish to evaluate some public function on their private inputs, where all agents are equal and the evaluated result is broadcasted to all of them or at least the honest ones. However, when viewing through the game-theoretic lens, the function evaluation process can be realized as the exchanging of private information among those agents, and hence the agents are *not equal*. For example, an agent with higher influence on the function tends to have a smaller incentive in the cooperation, and in the extreme case a "dictator" would have zero incentive; or even if the function is symmetric, an agent may still be less incentivized because of a high quality private input which provides a better prior than others. An example of a dictator is an agent with input zero when the function is *AND*; such an agent already knows the output of the computation and can learn nothing from others.

To this end we suggest to consider the procedure fairness (rather than the result fairness) in terms of *information benefit*, which measures how much an agent improves the quality of her own private information by participating. We believe this is a better characterization of the agent incentives. Also from the game-theoretic point of view, it makes sense to consider the agents as rational and self-motivated individuals rather than simply "good/bad" or "honest/semi-honest/malicious" as is typically done in cryptographic scenarios.

In this work, we study mechanisms for exchanging information without monetary transfer among rational agents. These agents are rational and self-motivated in the sense that they only care about maximizing their own utility defined in terms of information. More specifically, we focus on utility functions that capture the following properties about the behavior of the agents:

- *Correctness*: The agents wish to collect information from other agents.
- *Exclusivity*: The agents wish to have exclusive access to information.

[1] All proofs that are omitted from this chapter can be found in [7].

The wish to collect information incentivizes cooperation, while the wish for exclusivity deters it. By unifying the above competing factors, agents aim to strike a balance between the two. The value of exclusivity is a concept studied in many areas of economics (e.g. labor economics, economics of the family, etc); see, e.g. [39] for a study on the role of exclusivity in contracts between buyers and sellers and [29] for platform-based information goods.

Utility functions that capture these competing factors are relevant in modeling situations where both cooperation and competition exist simultaneously, such as several companies wishing to exchange their private but probably overlapping information, e.g. training data for machine learning purpose, predictions for the stock market, etc.

We investigate specific information exchanging problems, such as Multi-party Set Union, as well as Set Intersection and Average. For example, in the set union problem there is a number of agents, each owning a set, and the goal is to find the union of the private sets held by all the agents. Set intersection is similarly defined except the goal is to find the intersection. Since for such problems the result is not Boolean and agents with different quality input should get different results, the value of result is measured by quality (accuracy) rather than by a Boolean indicator of whether it is the optimal one.

For the behavior of the agents, many of our results are for the "all-or-nothing model" where every agent either fully participates by truthfully submitting their input or refuses to participate. We also have several results for games with few agents in the richer model where agents can partially participate, by submitting some but not all of their information, as well as open questions.

The all-or-nothing model captures realistic scenarios where the inputs are authenticated by some trusted authority (e.g. using digital signatures), or where the inputs were already collected by some central entity, and the only choice of the agent is whether to allow their input to be used in the computation. Another motivation for the all-or-nothing model is when the inputs are later checked (e.g. in court or in future rounds of repeated games).

The participants send their private input to the trusted mediator (i.e. "principal") who runs a publicly known protocol (mechanism) to decide the payoff of each agent. Here the payoffs are customized pieces of information since we are studying the information exchanging mechanism without money.

As a simple example to motivate our work, consider the following scenario: Suppose there is a group of people and everyone is interested in finding a gold mine. The gold mine is situated in location t. Everyone has some estimate of where the gold mine is t_i and some uncertainty given by a radius d_i, i.e. each player i has an interval $[t_i - d_i, t_i + d_i]$. The players want to join their information to get a better approximation of the location of the gold mine and know that the gold mine lies in the intersection of all the estimates (sets). However if a player i knows that its radius d_i is much smaller than that of another player j, then player i knows that it won't learn much by interacting with player j. That is, in the worse case player i's interval is contained in player j's interval, so there is no information player i can infer from j. Since player i would rather not have player j gain free information without receiving anything in return, the problem is to design a mechanism that incentivizes the players to learn from each other (as much as possible). This is the problem that, later in the chapter, we refer to as the *set intersection problem*.

1.1.1 Summary of results

In this chapter we present a framework for non-monetary mechanism design with utility functions unifying both preferences of correctness and exclusivity. Let $N = \{1, \ldots, n\}$ be a set of agents. Suppose each agent has some piece of information, the details of which we intentionally leave informal for now. Given some mechanism M that the agents use to exchange information among themselves,

we define the information benefit v_i of an agent i to be the additional "information" gained by i after participating in M. For example, in the case of the set union problem, where each agent owns a set of elements and tries to learn additional elements from other agents, this gain could be the number of additional elements learned by an agent compared to what that agent already knew.

The utility function will capture the tension between the wish to learn and the wish for exclusivity and the simple instantiation that we focus on is $u_i = v_i - \max_{j \in N \setminus \{i\}} v_j$. Thus each agent wishes to learn as much as possible while maintaining exclusivity over the information, which is captured by minimizing the amount obtained by others. This definition is connected to the notion of envy-freeness; in particular, it captures the maximum "envy" that an agent i could have towards any another agent, and the goal is to reduce envy.

Our technical contribution is to design mechanisms for natural joint computation tasks such as *Multi-party Set Union*, as well as *Set Intersection* and *Average*. We focus on mechanisms that incentivize agents to submit the information they have as well as ensure properties such as Pareto efficiency[2] of the final allocation.

In the Multi-party Set Union Problem each agent owns a set x_i drawn from some universe $\mathscr{U}$. The utility functions are as described above. The strategy space of an agent consists of sets they can submit to the mechanism. We assume that agents can hide elements of their set, but not manufacture elements they don't have (i.e. there is a way to detect forgery). The question is to design a mechanism that incentivizes the agents to show their set of elements to others.

Theorem 1.1 *There is a truthful and Pareto efficient mechanism for set union among $n = 3$ agents. The mechanism runs in polynomial time.*

We leave open the general mechanism design question for any number of agents.

Open Problem 1 *Is there a truthful polynomial time mechanism for set union for any number of agents? Are there randomized such mechanisms?*

However, we manage to solve this problem for the special case where each agent can either submit its whole set or the empty set, i.e. cooperate or not. We call this the "all-or-nothing" model.

Theorem 1.2 *There is a truthful, Pareto efficient, and welfare maximizing mechanism[3] for set union among any number n of all-or-nothing agents. The mechanism runs in polynomial time for any fixed n.*

We further show that this mechanism satisfies several other desirable properties, such as treating equal agents equally and rewarding more agents that contribute more.

Beyond multi-party set union, we also consider two case studies of problems with sets. The first is a set intersection problem, where each agent owns a connected set (interval) on the real line. The agents have to find an element in the intersection of all the sets and are promised that such an element exists. A high level example of this problem is when the agents are trying to find a gold mine as described in the introduction.

Theorem 1.3 *There is a truthful polynomial time mechanism for interval intersection among any number n of all-or-nothing agents.*

[2] Pareto effiency ensures that no agent can be better off without making someone else worse off.

[3] Welfare maximization is achieved by maximizing over all the Pareto efficient outcomes.

The second case study is a point average problem, where each agent has a point and the goal is to compute the average value of their inputs.

Theorem 1.4 *There is a truthful polynomial time mechanism for the point average problem for any number n of all-or-nothing agents.*

We also provide a more general theorem for arbitrary value functions (e.g. that do not necessarily count the number of elements in a set).

Finally, two more high-level remarks are in order.

Why not maximize social welfare? A trivial solution to problems such as set union can be to have everyone learn everything (i.e. maximize the sum of information gains). In traditional settings such as auctions or elections it is unlikely that every agent maximizes their information benefit simultaneously since their ideal outputs are usually conflicting, e.g. there is only one indivisible good that cannot be assigned to more than one agent. However, in the world where information replaces material goods, it becomes possible to duplicate the information at (nearly) zero cost such that every agent gets all information and hence maximizes their utility at the same time. This straightforward mechanism only works if all agents are selfless and choose to report truthfully. However, it is unfair in the sense that the more one agent contributes, the less benefit they could get (since the information benefit is bounded by the whole information minus their private information). Furthermore, the straightforward mechanism fails badly when agents take exclusivity into consideration: e.g. the dominant strategy would be "revealing nothing to the mechanism but combining the output with the private input afterward" and eventually the equilibrium becomes that no exchange happens at all (similar to the "rational secret sharing" problem discussed in [22, 24, 27]) when partial participation and strategic lies are allowed; and even in the all-or-nothing model an agent may prefer not participating according to their own utility function if their advantage over other competing agents would decrease.

A Note on Mechanism Design. The intuition behind our constructions is that every agent, when participating in the cooperation, should get a benefit no less than the loss they could cause to others by not participating. At first glance it might seem that the "loss to others" inflicted by a non-participating agent would be bounded by the exclusive information of that agent. However, it turns out that agents contribute much more to the mechanism than simply their private inputs. In particular, the participation of an agent may increase social welfare by giving incentives for participation to other agents with "better" inputs. An agent i with a high quality input might choose to join the computation, or reveal more of their private information, because, by doing so, they can reduce the information benefit of some other agent j (which is rational when it reduces i's own exclusivity loss).

Therefore, the key idea behind our constructions is to characterize the marginal contribution of every agent and assign information accordingly so that nobody prefers to leave the cooperation (and in the meanwhile we aim to maximize the social welfare among all stable allocations). For example, this idea is instantiated as a round-by-round exchange mechanism for the Three-Party Set Union problem (in Section 1.3.2), such that in every single "round" of exchange each agent gains more benefit than he offers to others.

1.1.2 Related Work

Our setting is reminiscent of cooperative game theory and the well-known solution of Shapley value [40, 38, 4], except that now the agents are rewarded with information instead of money. There are two main distinctions: a) Information can be duplicated, for free or with negligible cost; b) Every piece of information is unique whereas money is fungible, e.g. the same piece of information could

have different values for different agents. The first property results in an unfixed total profit (sum of all agents' payoffs) and so breaks the intuition of "distribute the total surplus *proportionally* to each agent's contribution" used in Shapley value. The second property requires the mechanism to specify not only the *amount* of information but also the *details* of information allocation. In particular, the information already contained in an agent's input cannot be used to reward that agent. Such a property also leads to a subtle dilemma — the more an agent contributes, the less they can get as a reward from the mechanism — e.g. an agent with all information cannot get new information from other agents. Therefore, the mechanism must be able to motivate the most informed agents even though they may not benefit as much as those that know less (i.e. with lower quality inputs). A different line of work has studied the problem of sharing information when the inputs are substitutes or complements [13], which defined the value of information (and of a marginal unit of information) and instantiated it in the context of prediction markets.

Our model can be seen as an extension of the *non-cooperative computation* (NCC) framework and *informational mechanism design* (IMD) introduced in [42, 30], where they characterize Boolean functions that are computable by rational agents with non-monetary utility functions defined in terms of information. In their model, the agents are trying to compute a public Boolean function on their private inputs with the help of a trusted center. Every agent claims their type (truthfully or not) to the center, and gets a response from the center (typically but not necessarily the Boolean function evaluated on claimed types). Agents may lie or refuse to participate, and they can apply any *interpretation function* (on the response from the center and their true input, so as to correct a wrong answer possibly caused by an earlier false declaration). In the setting of [42], the agents have a two-tiered preference of correctness preceding exclusivity[4], i.e. they are interested in misleading others only if this would not hurt their own correctness, whereas we generalize this lexicographic preference to a utility function incorporating both components (The lexicographic preference is a special case when one component is assigned a very small weight). Another extension is that we consider non-Boolean functions in this work and allow distinct responses to different agents, which significantly enriches the space of candidate mechanisms.

The line of work [22, 24, 27] focuses on the cryptographic implementation of truthful mechanisms for secret sharing and multi-party computation by rational agents without a trusted mediator. In their setting there is an "issuer" who authenticates the initial shares of all agents so that the agents cannot forge a share (just as in the all-or-nothing model). Then the agents use simultaneous broadcast channels (non-simultaneous channels are also considered in [27]) to communicate in a round-by-round manner. Since all messages are broadcasted in this setting, a rational agent tends to keep silent so that they can receive others' information without revealing their own and hence possibly gain advantage in exclusivity. Therefore, much of the efforts and technical depth along this line is spent on catching dishonest agents (who do not broadcast their shares when they are supposed to), based on the key idea that in any given round the agents do not know whether this is just a test round designed to detect cheaters, or whether it is the final round for the actual information exchange. [24] achieve a fair, rational secure multi-party computation protocol which prevents coalitions and eliminates subliminal channels, despite the drawback of requiring special purpose hardware such as ideal envelopes and ballot boxes. However, all of these works assume the two-tiered preference of correctness and exclusivity as in [42], where in particular the correctness dimension is Boolean, i.e. either "correctly computed" or not. As a result, these works fall into the category of "implementing cryptographic protocols with rational agents" rather than the more game-theoretic topic "informational mechanism design" which we address with this work.

[4] [30] considers other facets, such as privacy, but still in lexicographic ordering.

There is another line of work [31, 37] on mechanism design with privacy-aware agents who care about their privacy rather than exclusivity. The consideration of privacy is relevant in many applications but technically orthogonal to what we study in this work. (In our work, the privacy of the inputs is only a tool towards limiting the loss of utility due to the exclusivity preference, not a goal in itself).

The recent works of [12] and [5] investigate non-monetary mechanisms for cooperation among competing agents. However, an essential difference is that they consider a sequential delivery of outputs to different agents, such that the utility function is not merely in terms of information but also depends on the time or order when the output is delivered. For example, the “treasure hunting problem" in [12] is in particular very similar to the multi-party set intersection problem, except that in treasure-hunting only the first agent finding the common element gets positive utility while all others get zero.

1.2 One Dimensional Search

In this section we study a problem that is also related to sharing information captured through sets, except the goal is to find a point in the intersection of all the sets. This captures e.g., the gold mine example from the introduction.

More formally, suppose each player i has an interval in $[\alpha_i, \beta_i] \subset \Re$ and the goal is to find a point contained in all the intervals. We are promised that such a point exists.

We solve this problem for all-or-nothing players, where a player can either cooperate by submitting its interval $[\alpha_i, \beta_i]$ or not cooperate by submitting the whole set of real numbers $\Re$. Given that the intersection point chosen is t, the information benefit that a player derives from learning an interval $[a,b]$ will be given by an arbitrary monotone function v such that $v(a,b) = v(b-a)$ as long as $a \leq t \leq b$ and $v(a,b) = -\infty$ otherwise. Given an allocation where the benefit to each player i is v_i, the utility of a player i is $u_i = v_i - \max_{j \neq i} v_j$.

Theorem 1.5 *There is a truthful polynomial time mechanism for interval intersection among any number n of all-or-nothing players.*

Proof: Consider the mechanism listed as Mechanism 2 at the end of the chapter. The high level intuition is that the two players contributing to the most accurate interval are equally rewarded, i.e. these two players exchange their information fairly (equal benefit). Other players get no update on what they submitted.

For players that have intervals all different from each other, note that in Mechanism 2, the interval $[\alpha_j, \beta_k]$ tracks the intersection of all processed x_i's and eventually $[\alpha_j, \beta_k] = \bigcap_{i=1}^{n} [\alpha_i, \beta_i]$. Then it is easy to verify that such a mechanism is truthful for the utility function we study. To handle players that may have identical sets as input, note that Mechanism 2 can be made to handle this case by assigning identical results to agents reporting identical types. □

1.3 Multi-party Set Union

Let $N = \{1, \ldots, n\}$ be a set of agents. There is a universe $\mathscr{U} = \{u_1, \ldots, u_m\}$ of possible numbers, from which each agent i owns a subset $S_i \subseteq \mathscr{U}$ that is private to the agent. The goal of the agents is to obtain more elements of the universe from other agents by sending elements from their own set in exchange.

We study the problem of designing mechanisms that incentivize the participants to share their information with each other. A mechanism $\mathscr{M}$ will take as input from each agent i a set $x_i \subseteq S_i$ and

output a vector $\vec{y} = \mathscr{M}(\vec{x})$, so that the i-th entry of this vector contains the set received by agent i after the exchange.

Strategies. The strategy of an agent is the set it submits to the mechanism. Agents can hide elements (i.e. submit a strict subset of their true set), but not submit elements they don't actually have. A special case we will study in more depth is when the strategies of the agents are "all-or-nothing", i.e. $x_i \in \{\emptyset, S_i\}$. The input of each agent to the mechanism is sent through a private authenticated channel to the center.

Utility. We say the "information benefit" that agent i receives from sending their set S_i to the mechanism is the number of new elements that i obtains from the exchange: $v_i(\vec{x}) = |\mathscr{M}_i(\vec{x}) \setminus x_i|$. The *utility* of the agent is then defined as the minimum difference between their own information benefit and that of any other agent, formally given by $u_i(\vec{x}) = v_i(\vec{x}) - \max_{j \in N \setminus \{i\}} v_j(\vec{x})$.

The intuition is that each agent wishes to learn as much as possible while maintaining exclusivity, which is captured by minimizing the amount of information obtained by the other agents. This utility function is closely tied with the notion of envy as it compares the value for an agent with the maximum value of any other agent and the aim is to compute (approximately) envy-free outcomes.

Incentive compatibility and Efficiency. We are interested in mechanisms that incentivize agents to share their information and will say that a mechanism is *truthful* if truth telling is a dominant strategy for each agent regardless of the strategies of the other agents. An allocation (outcome) is *Pareto efficient* (or *Pareto optimal*) if there is no other outcome where at least one agent is strictly better off and nobody is worse off.

Fairness. Some of our mechanisms also satisfy fairness and the fairness notions we consider are *symmetry* and *strong dominance*. Symmetry requires that if multiple agents report inputs of equivalent quality, then they get the same information benefit (and so the same utility). Strong dominance stipulates that if the information reported by an agent is inferior to the information reported by another agent under some partial order, then the result sent to the first agent is also (weakly) inferior to the result sent to the second agent under that order.

1.3.1 Two Agents

As a warm-up, we provide a simple solution to the exchange problem for $n = 2$ agents.

Proposition 1 *There is a truthful polynomial time mechanism for the set union problem between two agents.*

Proof: W.l.o.g., the set owned by the second agent is larger: $|x_1| \leq |x_2|$. Let $y_2 = x_1 \cup x_2$ and $y_1 = x_1 \cup y_1'$, where y_1' is a set chosen so that $y_1' \subseteq x_2 \setminus x_1$ and $|y_1'| = |x_1 \setminus x_2|$. Then agents 1 and 2 can fairly exchange their exclusive elements until one of them has used up their exclusive elements. Note this type of exchange performed over multiple rounds can in fact be done in an atomic way by the principal. It is immediate that this mechanism ensures both agents get the same information benefit: $v_1 = v_2 = |x_1 \setminus x_2| \geq 0$ and it is weakly dominant for them to report their true information. □

1.3.2 Three Agents

For three agents the problem becomes more subtle, as the mechanism must specify the order of pairwise exchanging, the number of exchanged elements, and, more importantly, which elements are exchanged.

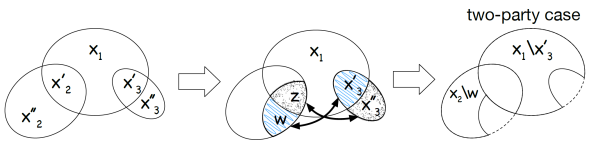
Theorem 1.6 *There is a truthful polynomial time mechanism for set union among $n = 3$ agents.*

Proof: The theorem will follow from the construction in Mechanism 3.[5] Mechanism 3 starts by removing the common elements among all three parties, since these elements will not affect the exchange; these elements are denoted by the set z_0. Then we consider the three pairwise intersections, from which the agents can exchange a number of elements bounded by the smallest intersection i.e. $s = \min\{|x_1 \cap x_2|, |x_2 \cap x_3|, |x_3 \cap x_1|\}$. Note that at the end of these exchanges at least one of these three intersections will be "used up". Thus we assume w.l.o.g. that after this step $x_2 \cap x_3 = \emptyset$ and $|x_2| \geq |x_3|$. Now we have reduced the original problem to a setting where there is no common intersection and only two pairwise intersections are non-empty, namely $x_1 \cap x_2$ and $x_1 \cap x_3$.

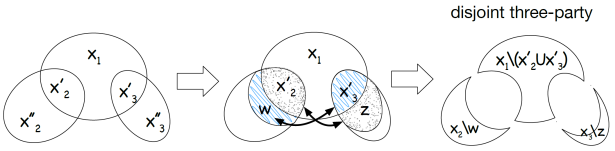
Let x_2, x_3 be partitioned into $x_2 = x_2' \cup x_2'', x_3 = x_3' \cup x_3''$ where $x_2' = x_2 \cap x_1, x_2'' = x_2 \backslash x_1$, and $x_3' = x_3 \cap x_1$, $x_3'' = x_3 \backslash x_1$. The intuition will be that elements in x_2' should be used to exchange elements in $x_3'' = x_3 \backslash (x_1 \cup x_2) = x_3 \cap \overline{x_1} \cap \overline{x_2}$, and similarly x_2'' for x_3'.

Next we discuss how exchanging looks like in different situations (as shown by the inline Figures).

Case 1: $|x_2'| \geq |x_3''|$ and $|x_2''| \geq |x_3'|$. This is the simplest case, where we can simply make agent 3 exchange all elements in $x_3 = x_3'' \cup x_3'$ with both agents 1 and 2 for an equal amount of elements in $z \subseteq x_2'$ and $w \subseteq x_2''$ respectively. Then, agent 3 used up all its elements and the problem reduces to the two-party case between agents 1 and 2 with remaining elements in $(x_1 \backslash x_3', x_2 \backslash w)$.

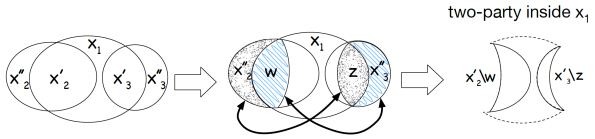


Case 2: $|x_3''| > |x_2'|$ and $|x_2''| > |x_3'|$. Then agent 2 uses $|x_3'|$ many elements in x_2'', denoted by w, to exchange all elements in x_3' with agents 1 and 3, and by symmetry agent 3 uses $|x_2'|$ many elements in z to exchange x_2' with agents 1 and 2. After this exchange all the three agents may have some elements left, but these are all exclusive elements, so the problem reduces to the easy case of three party with disjoint elements $(x_1 \backslash (x_2' \cup x_3'), x_2'' \backslash w, x_3'' \backslash z)$. Then the mechanism exchanges a number of elements equal to $\min\{|x_1 \backslash (x_2' \cup x_3')|, |x_2'' \backslash w|, |x_3'' \backslash z|\}$, further reducing the problem to the two-party case.



[5] The mechanisms are included at the end of the chapter.

Case 3: $|x_3''| < |x_2'|$ and $|x_2''| < |x_3'|$. In this case agent 2 uses x_2'' in exchange for $|x_2''|$ many elements in $z \subseteq x_3'$, and, by symmetry, agent 3 uses x_3'' to exchange $|x_3''|$ many elements in $w \subseteq x_2'$. After such an exchange the problem reduces to three parties with $(x_1 \setminus (w \cup z), x_2' \setminus w, x_3' \setminus z)$.



Finally, for improved welfare, agent 2 and agent 3 run a naïve two-agent exchange protocol with their remaining elements in $x_2' \setminus w$ and $x_3' \setminus z$. This is not optimal for agent 1, who has already collected full information and wants to end the exchange. However, agent 1 cannot prevent such exchange between agents 2 and 3 anyhow.

Mechanism 3 guarantees individual rationality because every round of exchange in its process is "fair" and "necessary". Every round is fair in the sense that all participants of that round get equal benefits — each of them gives out some elements in exchange for more new elements. Every round of such exchange is necessary because each element appears in at most one round, i.e. the mechanism does not reuse previously exchanged elements. Therefore, an agent that hides elements would suffer a loss lower bounded by the number of private elements that could have been traded, which is indeed a natural upper bound for the loss of others.

□

We note that Mechanism 3 is not Pareto efficient since the reduced problem (that is solved in the recursive call) is dealt with in a naïve way. For example, consider the last step in Case 1 i.e., after the problem has already been reduced to the two-party case. Here we could let agents 1 and 2 exchange their remaining elements without agent 3. This could be seen as fair, since agent 3 does not contribute new elements in those rounds. However, this procedure does not achieve Pareto efficiency, for that we can improve the social welfare by giving agent 3 some extra elements and, for sufficiently small number of elements, the utilities of agents 1 and 2 would not change and the solution would ensure that agents 1 and 2 remain truthful as before.

Theorem 1.7 *There is a truthful and Pareto efficient mechanism for set union among $n = 3$ agents.*

Proof: For case 1, consider the last step (of this case) in the execution of the mechanism. Mechanism 3 can be modified to assign randomly selected extra elements to player 3 so that $|y_3 \setminus x_3| = v_1 = v_2$ (recall that Mechanism 3 ensures $v_1 = v_2$). This modification achieves Pareto efficiency since any further improvement on social welfare will decrease the utility of player 1 or player 2, who already get all elements and cannot get more information benefit. Now we prove that the above modification also preserves truthfulness. This is immediate for players 1 and 2 but requires the following observations to see that it continues to hold from the point of view of player 3:

- each of player 3's exclusive elements in x_3'' leads to the same amount of marginal benefit to player 3 as to players 1 and 2, i.e. it is used to exchange for either one element in x_2', which will not be exchanged between players 1 and 2, or two elements when players 1 and 2 exchange elements in $x_2 \setminus w$ and $x_1 \setminus (x_2' \cup x_3')$, respectively.
- all of player 3's elements in x_3' do not affect others' information benefits; however, such elements can help player 3 since they might prevent the player from receiving some previously known element as the extra benefit.

We note that the same modification also works to ensure Pareto efficiency in case 2, while case 3 already ensures a Pareto efficient exchange. Thus there exists a truthful mechanism that is Pareto efficient. □

1.3.3 Any Number of Agents

We observe that the three agent mechanism above relies on a complex analysis that depends on the different intersection sets. The number of intersections increases exponentially as n grows and we leave open the question of whether it is possible to achieve an analogue of Mechanism 3 for more than three agents.

Open Problem 1. *Is there a truthful polynomial time mechanism for set union for any number of agents? Are there randomized such mechanisms?*

In Mechanism 1, we show a truthful mechanism for set union for any number of agents in the special case where each agent can either submit its whole set or the empty set, i.e. cooperate or not. We call this the "all-or-nothing" model and our main result is:

Theorem 1.8 There is a truthful, Pareto efficient, and welfare maximizing mechanism for set union among any number n of all-or-nothing agents. The mechanism runs in polynomial time for any fixed n.

The Mechanism is provided at the end of the chapter, and the proof of the theorem can be found in the original paper.

1.4 Beyond Union: Intersection and Average

Moving beyond the multi-party set union problem, we suggest two other set problems where the agents own data points and wish to share them.

Intersection. The first problem is interval intersection, where each agent owns an interval in $\mathfrak{R}$ and the goal is to find a point in the intersection of all the sets. A high level scenario motivating this problem is the gold mine example from the introduction, where there is a group of people trying to find the location of a gold mine, and each person has an estimate of where the gold mine is, given by a center and a radius. The agents would like to merge their estimates to get a better idea of where the mine is situated, but the challenge is that agents with very good estimates (i.e. small radius) will not learn much from those with worse estimates (i.e. larger radius).

Theorem 1.9 *There is a truthful polynomial time mechanism for interval intersection among any number n of all-or-nothing agents.*

Set Average. The second problem is taking the average of a set that is distributed among the agents.

Theorem 1.10 *There is a truthful polynomial time mechanism for the average point problem among any number n of all-or-nothing agents.*

The mechanisms are given at the end of the chapter. In the chapter we also discuss mechanisms for sharing problems where the value used to estimate the benefit of an agent is more general.

1.5 Discussion

Aside from our concrete open questions, the directions of generalizing the results to richer strategy spaces, allowing randomization, and more general utility functions are also interesting. And at a higher level, the problem of understanding the interplay between having value for information and having value from exclusivity remains largely open.

1.5.1 Any Number of Players

Theorem 2.3 (restated) *There is a truthful, Pareto efficient, and welfare maximizing mechanism for set union among any number n of all-or-nothing players. The mechanism runs in polynomial time for any fixed n.*

To prove the theorem we develop several lemmas.

Lemma 1 *Let M be any mechanism for set union for n all-or-nothing players. If the mechanism ensures that in every execution the information benefit of each player is the same as that of any other player, then the mechanism is truthful and Pareto efficient.*

Proof: Consider the outcome of any execution of M and suppose there is a value V so that the information benefit of each player i is $v_i = V$. An assignment with this property must satisfy truthfulness on this instance for all-or-nothing players since every player i gets utility $u_i = v_i - \max_{j \neq i} v_j = 0$ when participating and non-positive utility when not participating.

By definition, $u_i = v_i - \max_{j \in N \setminus \{i\}} v_j$ for every player i. Then we have $\sum_{i \in N} u_i = \sum_{i \in N} v_i - \sum_{i \in N} \max_{j \in N \setminus \{i\}} v_j \leq 0$ It follows that for any player i with $u_i > 0$ there must be another player j with $u_j < 0$, so no Pareto improvement is possible for this solution. □

In particular, for $V = 0$, assigning $y_i = x_i$ to every agent A_i is always a stable and Pareto optimal solution, since there is no difference between participating or not. However, this trivial solution is obviously the worst in social welfare.

To achieve maximum social welfare among all truthful and Pareto optimal solutions, we notice that by increasing the unified gain V the social welfare grows respectively while preserving truthfulness and Pareto optimality. However, V cannot be arbitrarily large since there is a systematical upper bound for the possible gain of agents with large sets, i.e. $v_i = |y_i \setminus a_i| \leq \left| (\cup_i x_i) \setminus a_i \right|$ since $y_i \subseteq \cup_i x_i$.

The full characterization of Pareto optimal solutions for the Multi-party Set Union is summarized next.

Lemma 2 *Let M be a mechanism for set union among n players. Suppose that on some execution, the information benefit of each player i is v_i.*

Let $i = \text{argmax}_{j=1}^{n} v_j$ and $V = \max_{j \neq i} v_j$. Then the allocation is Pareto optimal if and only if for every $j \neq i$, player j gets an information benefit of $v_j = \min\left\{V, \left| \bigcup_k x_k \setminus a_j \right|\right\}$. That is, every player j, except player i who gets the maximum, gets V many new elements (unless there are fewer than V that can be assigned to that agent).

Proof: Suppose that on the given instance there is a player j that gets a benefit bounded by $v_j < V$ and $v_j < \left| \cup_k x_k \setminus a_j \right|$. Then the mechanism can be modified on this instance to increase v_j by one to achieve higher social welfare. This is a Pareto improvement since no other players care about it: player i only cares about the one who already receive V, while all the other players envy player i's information gain.

On the other hand, every such allocation achieves Pareto optimality since every agent's information gain is either impossible to improve or envied by another agent and hence cannot be improved without decreasing the utility of any other agent. □

In the rest of this section, we focus on determining the unified value V that maximizes social welfare while preserving truthfulness and Pareto efficiency.

1.6 Average Point Problem

In this section we consider another well-known multi-party problem—the Average Point problem—where every agent is assigned a private input and they want to compute the average value of their private inputs. This problem is very different from set union, in the sense that the quality of private information cannot be objectively measured, since it depends on the private inputs of other agents. As a result, we resort to a mechanism where in some sense the principal treats the agents more equally.

Let the universe $\mathscr{U}$ be a metric space (e.g. $\mathscr{U} = \Re$ or $\mathscr{U} = \Re^2$). There are $N = \{1,\ldots,n\}$ players, so that each player i has a private point $a_i \in \mathscr{U}$. The goal is to compute the average point $\overline{a} = \sum_{i=1}^{n} a_i/n$. We focus on all-or-nothing players, who either submit their point a_i or nothing, the latter of which is denoted by $\perp$.

The value of each point $y \in \mathscr{U}$ is given by the square loss function: $v(y) = -|y-\overline{a}|^2$; for completeness, we define $v(\perp) = -\infty$.[6] The information benefit of a player i on receiving the value y_i from a mechanism $v_i = v(y_i) - v(a_i)$ if $x_i = a_i$ and $v_i = 0$ otherwise. The utility of player i is the same as before: $u_i = v_i - \max_{j\neq i} v_j$.

Our main result in this section is a mechanism for this problem.

Theorem 1.11 *There is a truthful polynomial time mechanism for the average point problem among any number n of all-or-nothing players.*

Proof: Let $(x_1,\ldots,x_n)$ be the reported inputs, we now design the mechanism by specifying y_i's that T assigns to the agents. Since all participants appear in equal positions, the mechanism is defined as follows:

$$y_i = f_i(x_1,\ldots,x_n) = \begin{cases} \overline{x} = \frac{\sum_{x_i\neq\perp} x_i}{M} & \text{if } x_i \neq \perp \\ \perp & \text{if } x_i = \perp \end{cases} \tag{1}$$

where $M(x_1,\ldots,x_n) = \#$ of x_i's such that $x_i \neq \perp$.

That is, the mechanism computes the average point $\overline{x}$ of all reported points, and sends $\overline{x}$ to all participants (and $\perp$ to the nonparticipants).

The above mechanism f trivially satisfies the properties of "maximal social welfare with Pareto efficiency", "symmetry" and "null agent gets zero", since all participants get identically the optimal result that the mechanism could offer. [7]

We now have to argue that f also guarantees the players submit their data points.

[6] More generally, we could define $v(y) = -d(y,\overline{a})^t$ as the t-th moment of the metric distance between y and $\overline{a}$, where $d(\cdot)$ denotes the metric equipped by U. Note also that the value function $v(\cdot)$ depends on $\overline{a}$ computed from true types $(a_1,\ldots,a_n)$ rather than the reported $(x_1,\ldots,x_n)$. This is meaningful, for instance, in applications in which the principal has already collected the types of the agents, and the only choice left to the agents is whether to allow their type to be used in the computation or not.

[7] Note the strong dominance property does not make much sense for the average point problem, because in each execution there are merely two possible outputs by (1) and the only inferior relation in results reflects participation/non-participation.

Consider the case where a subset $S \subseteq N$ of agents chooses to participate and others do not. Then, for every player $i \in S$, we have $y_i = \bar{x} = \sum_{A_i \in S} x_i / |S|$. If an agent, say $1 \in S$, deviates by switching from $x_1 = a_1$ to $\widehat{x}_1 = \perp$, then their result changes from $y_1 = \bar{x}$ to $\widehat{y}_1 = \perp$ and thus their information benefit changes from $v_1 = v(\bar{x}) - v(a_1)$ into $\widehat{v}_1 = 0$. That is,

$$\Delta v_1 = \widehat{v}_1 - v_1 = v(a_1) - v(\bar{x})$$

For every other agent $i \in S \setminus \{1\}$ the result changes from $y_i = \bar{x}$ to

$$\widehat{y}_i = \bar{x}' = \sum_{i \in S \setminus \{1\}} x_i / (|S| - 1),$$

and so $\Delta v_i = \widehat{v}_i - v_i = v(\bar{x}') - v(\bar{x})$ for every $i = 2, 3, \ldots, n$. It suffices to show that $\Delta v_1 \leq \Delta v_i$ for all $i > 1$.

When all agents participate in the computation and $S = N$, we have $\bar{x} = \bar{a}$ and hence $v(\bar{x}) = 0$, $\Delta v_i = v(\bar{x}') \leq 0$. Then, by deviating from cooperation player 1 changes the information benefit of others from $y_i = \bar{x}$ to $\widehat{y}_i = \bar{x}' = (n\bar{a} - a_1)/(n-1)$. The value of $\widehat{y}_i$ is as follows:

$$\begin{aligned} v(\widehat{y}_i) = v(\bar{x}') &= -\left|\bar{x}' - \bar{a}\right|^2 = -\left|\frac{\bar{a} - a_1}{n-1}\right|^2 \\ &\geq \frac{-|a_1 - \bar{a}|^2}{n-1} = \frac{1}{n-1} v(a_1) \end{aligned}$$

Therefore, since $v(\bar{x}) = 0$ and $\Delta v_i = v(\bar{x}') - v(\bar{x}) = v(\bar{x}') \leq 0$ for $i = 2, 3, \ldots, n$, we have

$$\begin{aligned} \Delta v_1 =& v(a_1) - v(\bar{x}) \leq (n-1) v(\bar{x}') - v(\bar{x}) \\ =& (n-1)\left(v(\bar{x}') - v(\bar{x})\right) = (n-1)\Delta v_i = \sum_{i=2}^{n} \Delta v_i \end{aligned}$$

Thus the players will act truthfully since the utility of every agent will decrease when deviating from telling the truth.

We further remark that the above argument holds not only for the square loss function $v(y) = -|y - \bar{a}|^2$, but also extends to $v(y) = -|y - \bar{a}|^p$ when $p > 0$. □

1.7 General Sharing Problems

In this section, we investigate the more general case when the value function v satisfies only minimal assumptions. More specifically, we introduce a subgroup value function $V : \{0,1\}^{|N|} \to \mathfrak{R}$ such that for every subset $S \subseteq N$, the collection value $V(S)$ is defined as the value function $v(\cdot)$ evaluated on the optimal combination of all private types of agents in S. Moreover, the function V is monotone in the sense that for every $S' \subseteq S \subseteq N$, $V(S') \leq V(S)$.

The utility of an agent is defined as before, i.e. the difference between the information gain of i and the maximum gain of any other player. In the following, we will say that a set of players cooperate if they submit their true input.

Definition 1 *Let S be the set of cooperative agents and V be the subgroup value function. The* rewardable contribution *of $i \in S$ in the coalition S is denoted by $\phi_i(S)$ such that*

$$\begin{aligned} \phi_i(S) = \max_{T \subseteq S, i \in T} \Big\{ & \min\left\{V(T) - V(\{i\}), \right. \\ & \left. V(T) - V(T \setminus \{i\})\right\} \Big\} \end{aligned}$$

Intuitively, Mechanism 4 rewards every agent with respect to his contribution within the feasible amount.

Theorem 1.12 *Mechanism 4 is truthful for any number n of all-or-nothing players. It is symmetric and satisfies strong dominance given a universal traversal function over the type set of the players.*

Corollary 1 *Mechanism 4 can be modified to a Pareto optimal mechanism for* $v = \max_i \phi_i(S)$ *by Lemma 2.*

The proof of the Theorem can be found in the original paper.

We remark that in this mechanism the most beneficial agent gets a reward bounded by his own contribution (i.e. $\max_{T \subseteq S} \{V(T) - V(T \setminus \{i\})\}$) and hence not necessarily the maximal feasible (i.e. $V(S) - V(\{i\})$).

Mechanism 1: Multiparty Set Union

Input: $(x_1, x_2, \ldots, x_n)$, where each set $x_i \subseteq \mathscr{U}$ is the input from player i.
Output: $(y_1, y_2, \ldots, y_n)$, where each set y_i is sent to player i.

Fix an ordering π of all elements in $\mathscr{U}$ /* π will be used to specify the exchanged elements */
$u = \bigcup_{i=1}^{n} x_i$
$V = \texttt{ComputeV}(x_1, \ldots, x_n)$ /* the function ComputeV is defined below */
foreach *player* $i \in [n]$ **do**
  $v_i = \max\{V, |u \backslash x_i|\}$
  Let r_i be the set of first v_i elements in $z_{-i} \backslash x_i$ according to π
  $y_i = x_i \cup r_i$
end
return $(y_1, y_2, \ldots, y_n)$.

Function $\texttt{ComputeV}(x_1, \ldots, x_n)$
  if $n \leq 1$ **then**
    return 0
  foreach *player* $i \in [n]$ **do**
    $z_{-i} = \bigcup_{j \neq i} x_j$
    $V_{-i} = \texttt{ComputeV}(x_{-i})$
  end
  $V = \min\left\{\min_{k \in [n]}\left\{|z_{-k} \backslash x_k| + V_{-k}\right\}, \max_{k \in [n]} |z_{-k} \backslash x_k|\right\}$
return V.

Mechanism 2: One Dimensional Search Mechanism

Input: $(x_1, \ldots, x_n)$, where x_i is the set submitted by each player i.
Output: $(y_1, \ldots, y_n)$, where y_i is the set received by each player i.

foreach *player* i **do**
  $y_i = x_i$
end
$j = 1; k = 1$
foreach *player* i **do**
  if $\alpha_j < \alpha_i$ **or** $(\alpha_j = \alpha_i$ **and** $\beta_j > \beta_i)$ **then**
    $j = i$
  end
  if $\beta_k > \beta_i$ **or** $(\beta_k = \beta_i$ **and** $\alpha_k < \alpha_i)$ **then**
    $k = i$
  end
end
if $j \neq k$ **then**
  Select points β_j', α_k' so that $v(y_j) - v(x_j) = v(y_k) - v(x_k) > 0$
  $y_j = [\alpha_j, \beta_j']; y_k = [\alpha_k', \beta_k]$
end
Output: $(y_1, \ldots, y_n)$

Mechanism 3: Three Party Set Union

Input: Set $x_i \subseteq \mathscr{U}$ for each player i
Output: Set $y_i \subseteq \mathscr{U}$ for each player i

$z_0 = x_1 \cap x_2 \cap x_3$
foreach *player i* **do**
  $y_i = x_i$
  $x_i = x_i \backslash z_0$
end
$s = \min\{|x_1 \cap x_2|, |x_2 \cap x_3|, |x_3 \cap x_1|\}$ /* W.l.o.g., $|x_2| \geq |x_3|$ and $s = |x_2 \cap x_3|$ */
$z_1 = x_2 \cap x_3$
Select arbitrary sets $z_2 \subseteq x_3 \cap x_1$ and $z_3 \subseteq x_1 \cap x_2$ of sizes $|z_2| = |z_3| = s = |z_1|$
foreach *player i* **do**
  $y_i = y_i \cup z_i$
  $x_i = x_i \backslash (z_1 \cup z_2 \cup z_3)$
end
$x'_2 = x_2 \cap x_1$; $x''_2 = x_2 \backslash x_1$
$x'_3 = x_3 \cap x_1$; $x''_3 = x_3 \backslash x_1$
$(y'_1, y'_2, y'_3) = (\emptyset, \emptyset, \emptyset)$ /* Sets to store elements from recursive calls, if any. */
if $|x'_2| \geq |x''_3|$ *and* $|x''_2| \geq |x'_3|$ **then**
  /* Case 1 */
  Select arbitrary sets $z \subseteq x'_2$ and $w \subseteq x''_2$ of sizes $|z| = |x''_3|$ and $|w| = |x'_3|$
  $y_2 = y_2 \cup x_3$
  $y_3 = y_3 \cup z \cup w$
  $y_1 = y_1 \cup w \cup x''_3$
  $(y'_1, y'_2) = \text{TwoPartySetUnion}(x_1 \backslash x'_3, x_2 \backslash w)$
else if $|x''_3| > |x'_2|$ *and* $|x''_2| > |x'_3|$ **then**
  /* Case 2 */
  Select arbitrary sets $w \subseteq x''_2$ and $z \subseteq x''_3$ of sizes $|w| = |x'_3|$ and $|z| = |x'_2|$
  $y_2 = y_2 \cup x'_3 \cup z$
  $y_3 = y_3 \cup x'_2 \cup w$
  $y_1 = y_1 \cup z \cup w$
  $(y'_1, y'_2, y'_3) = \text{ThreePartySetUnion}(x_1 \backslash (x'_2 \cup x'_3),\ x''_2 \backslash w, x''_3 \backslash z)$ /* Recursive call with disjoint sets. */
else
  /* Case 3: $|x''_3| < |x'_2|$ and $|x''_2| < |x'_3|$ */
  Select arbitrary sets $w \subseteq x'_2$ and $z \subseteq x'_3$ of sizes $|w| = |x''_3|$ and $|z| = |x''_2|$
  $y_2 = y_2 \cup x''_3 \cup z$
  $y_3 = y_3 \cup x''_2 \cup w$
  $y_1 = y_1 \cup x''_2 \cup x''_3$
  $(y'_2, y'_3) = \text{TwoPartySetUnion}(x'_2 \backslash w, x'_3 \backslash z)$
foreach *player i* **do**
  $y_i = y_i \cup y'_i$ /* Add elements obtained from recursive calls, if any, to the final set for each player. */
end
return (y_1, y_2, y_3)

Mechanism 4: General Mechanism

Input: x_i for each player i.

Output: $y = (y_1, \ldots, y_n)$ with y_i assigned to player i.

$S = \{i \mid x_i \neq \perp\}$
foreach $i \notin S$ **do**
  $y_i = \perp$
end
foreach $i \in S$ **do**
  find $y_i \in Y$ such that $v(y_i) = \phi_i(S)$
  [in case such y_i does not exist, select y_i such that $\mathrm{E}[v(y_i)] = \phi_i(S)$*]*
end
Output $(y_1, \ldots, y_n)$.

2 Achieving Differential Privacy in Secure Multi-Party Computation

This chapter presents building blocks and mechanisms for differential privacy in an MPC setting. While the chapter provides the overview, for details the reader is referred to [43].

While data may be used for purposes such as research, improvement of services and other good causes, disclosing sensitive data as part of this process could harm the people whose data are contained in the database(s) involved. Particularly, if results depend on whether a person is included in the data or not then information is disclosed about that person. The idea of preventing this violation of privacy, is called *differential privacy*, coined by Dwork *et. al.* [19]. Differential privacy is a solution for preserving the privacy of the individuals in the database while maintaining a certain degree of usefulness of the results. This is done by introducing noise on the results using so-called *noise generating mechanisms*.

We introduce the necessary secure protocols to implement noise generating mechanisms, including $\exp(\cdot)$, $\ln(\cdot)$, $\sqrt{(\cdot)}$, $\cos(\cdot)$ and $\sin(\cdot)$. All protocols are designed to perform optimally in an MPC setting.

The literature proposes many candidates for noise generating mechanisms (NGM) that are proven to be differential private. This chapter highlights a number of the most relevant mechanisms to investigate their potential in the MPC setting. The underlying algorithms have been optimized with respect to performance speed and complexity.

2.1 Building Blocks for Randomness Generation

To build secure protocols for noise generating mechanisms, secure protocols are needed to perform a couple of basic arithmetic operations. The following methods are designed for specifically the fixed-point number type, offering sufficient and acceptable precision for differential privacy and other applications.

2.1.1 Experimental Design

The protocols presented in the coming sections are analyzed for the following aspects. All experiments are done on a normal PC using Jupyter Notebook (Anaconda) and the MPC framework MPyC [3] in 3-party mode.

Precision and speed. The default setting of datatype `SecFxp` has a bit length of 32 bits and a fractional part of 16 bits. This means the range of such a number is approximately 10^5 and the range of the fractional part is approximately 10^{-5}. The aim is to have a precision equivalent to the limits of this range with an acceptable speed. In order to determine the precision, experiments are done by sampling from the Python package `numpy` for random floating-point numbers in a feasible interval. Then the samples are used as input for the implemented protocols and the output values are compared to the output value given by the library functions of Python package `math`. The (averaged) relative error is computed to quantify the precision. The (averaged) performance speed of the implementations is determined by running the protocol 100 times and averaging the speed. The results are given in Table 2.

Complexity. Important aspects with respect to the complexity of an algorithm in MPyC are the number of generated random bits and the number of secure multiplications. These MPC protocols are expensive to run and take a significant amount of the computer's processing power. The usage of these operations should be minimized.

All information about the complexity of the building blocks in MPyC is to be found in Table 2. In this table, $\theta = \left\lceil \log_2 \left(\frac{2f+1}{3.5} \right) \right\rceil$ is the number of iterations performed in the protocol for secure division (a protocol included in the MPyC framework), l is the bit-length of the fixed-point number and f is the number of fractional bits.

Absolute and relative error. In the literature, there are several utility metrics used for analyzing the effect of an NGM $\mathscr{K}$ evaluated on a certain true answer $q(D)$, perturbing it to $q'(D)$. Here, the following metrics are used. For the absolute error, given true answer $q(D)$ and its perturbed value $q'(D)$, the absolute error is defined as |q(D)-q'(D)|. For the relative error, given true answer $q(D)$ and its perturbed value $q'(D)$, the relative error is defined as $\left| \frac{q(D)-q'(D)}{q(D)} \right|$.

2.1.2 MPC Protocols For Selected Functions

This section presents the protocols for secure evaluations of the following functions:

$$2^x, e^x, \ln(x), \sqrt{(x)}, \cos(x), \sin(x)$$

using existing protocols in MPyC and some additional protocols stated in [43]. The design of the protocols is inspired by MPC frameworks that already have these protocols implemented like SCALE-MAMBA [2]. A recurring pattern stands out in the design of the MPC protocols presented here. The general approach is as follows:

1. The input of the protocol is scaled down to a convenient domain.
2. A suitable approximation is used to evaluate the function on the scaled input.
3. Information about the scaling is used to correct the approximation.

The approximations, defined on a certain domain and with a fixed precision, are taken from the book 'Computer Approximations' by Hart [23]. They were also used in the protocols of SCALE-MAMBA. These approximations suits this particular application the best, because only n multiplications are needed to evaluate the approximations (using Horner's rule) since these approximations are based on polynomials with degree n. Furthermore, the polynomials have sufficient precision for the application of noise. Therefore, using these approximations is an efficient and relatively cheap solution for these MPC protocols.

Evaluation of 2^n for integer input. This function is essentially a building block for other protocols mentioned in this chapter. It computes $2^n, n \in \mathbb{Z}$. The idea is to take the bit decomposition $b_0, b_1, ...$ of the input n and to record on what positions i holds $b_i = 1$. The bit decomposition in MPyC is a list of bits starting from the least significant bit. The array has $l+1$ elements of which, since the input is integer, the first f bits are 0 with f being the length of the fractional part. First, the bits of n are shifted f to the right, so that the bit decomposition method can stop at position f. Since $2^0 = 1$, the start element of this iteration is 1. Every time $b_i = 1$ for some $i > 0$, the start element gets updated by multiplication with 2^{2^i}. The end result is $\prod_{b_i=1} 2^{2^i} = 2^n$. A similar reasoning is applied to the case that n is negative. The behavior of this method is as expected and satisfies the pre-determined goals.

Algorithm 1 Protocol for $2^n, n \in \mathbb{Z}$

Input: $n \in \mathbb{Z}$ and its bit length l and fractional length f
Output: $2^n, n \in \mathbb{Z}$

$n \leftarrow n >> f$ #shift f bits to right
$bp \leftarrow 1$
$bn \leftarrow 1$
$b \leftarrow$ `to_bits` $(n, l-f)$ #bit decomposition of n to the $l-f$-th bit
$s \leftarrow b[-1]$ #signed bit
for i `in range`$(0, f)$ **do**
  $t \leftarrow b[i]$
  $bp \leftarrow bp + t \cdot (-bp + bp \cdot 2^{2^i})$
  $bn \leftarrow bn + (1-t) \cdot (-bn + bn \cdot (0.5)^{2^i})$
return $s \cdot (0.5 \cdot bn) + (1-s) \cdot bp$

Evaluation of e^x for all input. For the evaluation of e^x, the base is changed as follows:

$$e^x = 2^{x \cdot \log_2(e)} \approx 1.442695040889 \cdot 2^x$$

Then the sign $s \in \{-1, 0, 1\}$ of x is determined, and the absolute value $|x|$ of x is taken to do further computations. Now, $|x|$ is split up in an integer part $x' \in \mathbb{N}$ and fractional part $x'' \in [0, 1)$ such that $|x| = x' + x''$. In order to reduce operations, computing $2^{x'}$ is done via a separate method `twoe(n)`. Since x' is always positive, the negative case is neglected in this variant of Algorithm 1, see [43]. The evaluation of $2^{x''}$ is approximated by p_{1045} defined on domain $[0, 1]$ and whose coefficients are given in [43]. It has a relative error of $10^{-12.11}$ if computed exactly, which corresponds to a precision up to 40 fractional bits. Then, the product is taken of $2^{x'} \cdot 2^{x''} = 2^{|x|}$. Since the sign of x could be negative, the inverse is computed as $1/2^{|x|}$. In case $x = 0$, the protocol evaluates to 1. In the end the sign determines the output.

Algorithm 2 Protocol for e^x

Input: fixed-point number x
Output: e^x

$x \leftarrow 1.442695040889 \cdot x$
$s \leftarrow \mathtt{sgn}(x)$ #outputs $\{-1, 0, 1\}$ depending on sign of x
$x \leftarrow s \cdot x$
$x' \leftarrow \mathtt{trunc}(x)$ #truncates x
$x'' \leftarrow x - x'$
$f \leftarrow 2^{x'}$
$g \leftarrow p_{1045}(x'')$
$h \leftarrow f \cdot g$
$h^{-1} \leftarrow \frac{1}{h}$
$k \leftarrow s \cdot s \cdot \left(\left(-\frac{s-1}{2} \right) \cdot h^{-1} + \left(\frac{s+1}{2} \right) \cdot h \right) - 1) + 1$
return k

From sampling 500 values in $[0, 1]$, the relative error is on average 10^{-6}. This means that the relative error will likely get worse when the result of the approximation is multiplied with other numbers, which is the case here. Since e^{10} is around 10^5, the sampling is done in $[-10, 10]$ and

the resulting relative error is on average 10^{-3}. From analyzing the absolute error, it is clear that the values have at least four digits correct. This is not strange, as the radix point shifts to the left when the approximation result is multiplied with a number of maximum size 10^3. The loss of precision is therefore inherent to the computations with fixed-point numbers and their range. When considering the context of generating noise, loss of precision will probably not lead to failure.

Evaluation of $\ln(x)$ for positive input. For a secure evaluation of $\ln(x)$, a similar approach based on an approximation is used. Here, p_{2607} is defined on domain $[\frac{1}{2},1]$ and has a relative error of $10^{-7.53}$ if computed exactly, which corresponds with a precision up to 25 fractional bits. The input x has to be scaled down to a number in interval $[\frac{1}{2},1]$. Suppose there is a k such that $\frac{x}{2^k} \in [\frac{1}{2},1]$. Then it holds:

$$\begin{aligned}\ln(x) &= \ln(\frac{x}{2^k} \cdot 2^k) \\ &= \ln(e^{\ln(\frac{x}{2^k})} \cdot e^{k \cdot \ln(2)}) \\ &= \ln(\frac{x}{2^k}) + k \cdot \ln(2)\end{aligned}$$

Here, $\frac{1}{2^k}$ and k are retrieved by calling a separate method `anorm()`, see [43]. Also, $\ln(\frac{x}{2^k})$ is approximated with p_{2607}, defined on domain $[\frac{1}{2},1]$ and whose coefficients are given in [43]. Furthermore, $\ln(2)$ is given by its fixed-point number representation as given below. The corresponding protocol is given with input secret shared value x and output the evaluation of $\ln(x)$. The behavior of this method is as expected and satisfies the pre-determined goals.

Algorithm 3 Protocol for $\ln(x)$

Input: fixed-point number x
Output: $\ln(x)$

$v \leftarrow \texttt{anorm}(x)$
$g \leftarrow x \cdot v[0]$ #$v[0]$ is the scaling factor $1/2^k$
$f \leftarrow p_{2607}(g)$
$k \leftarrow v[1]$ #$v[1]$ is k from scaling factor
$g \leftarrow f + k \cdot 0.69314718055994530$
return g

Evaluation of $\sqrt{(x)}$ for positive input. The computation of the square root of a given number x is done in a similar fashion as the logarithm:

$$\begin{aligned}\sqrt{x} &= \sqrt{\frac{x}{2^k} \cdot 2^k} \\ &= \sqrt{\frac{x}{2^k}} \cdot \sqrt{2^k} \\ &= \sqrt{\frac{x}{2^k}} \cdot 2^{\frac{k}{2}}\end{aligned}$$

Here, k is chosen such that $\frac{x}{2^k} \in [\frac{1}{2},1]$. Information about the parity of k is needed since evaluating $2^{\frac{k}{2}}$ securely requires integer input. In case k is even, $\frac{k}{2}$ will still be an integer and then $2^{\frac{k}{2}}$ can be evaluated securely using Algorithm 1. In case that k is odd, the following reasoning is applied:

$$2^{\frac{k}{2}} = 2^{\frac{k-1}{2}} \cdot \sqrt{2}$$

Using a fixed-point number representation of $\sqrt{2}$, the expression can be securely computed. The base is again the approximation p_{0132}, defined on domain $[\frac{1}{2},1]$ and whose coefficients are given in [43]. The polynomial has a relative error of $10^{-5.08}$ and that corresponds with a precision up to 16 fractional bits. The behavior of this method is as expected and satisfies the pre-determined goals.

Algorithm 4 Protocol for $\sqrt{x}$

Input: fixed-point number x and fractional length f
Output: $\sqrt{x}$

$t \leftarrow \texttt{is\_zero}(x)$ #checks whether t is zero
$v \leftarrow \texttt{anorm}(x)$
$k \leftarrow v[1]$ #$v[1]$ is k from scaling factor
$p \leftarrow k\%2$
$p \leftarrow p >> f$
$a \leftarrow p_{0132}(x \cdot v[0])$ #$v[0]$ is the scaling factor $1/2^k$
$s \leftarrow (1-p)\cdot(a\cdot \texttt{two}(\frac{k}{2})) + p\cdot(a\cdot \texttt{two}(\frac{k-1}{2})\cdot 1.4142135623730950488016887242 0)$
return $(1-t)\cdot s$

Evaluation of $\cos(x)$ and $\sin(x)$ for all input. A similar approach is used to evaluatie both $\sin(x)$ and $\cos(x)$. First, the input x has to be scaled down to the interval $[0,\frac{1}{2}\pi]$. This is done in several steps:

1. Compute $x' \equiv x \mod 2\pi$.
2. Identify in what interval x' is by the use of two bits b_1 and b_2. If $x' \in [\pi, 2\pi]$, then $b_1 = 1$, else, $b_1 = 0$. If $x' \in [\frac{1}{2}\pi, \pi]$, then $b_2 = 1$, else, $b_2 = 0$. After that, x' can be reduced to interval $[0,\frac{1}{2}\pi]$.
3. $\cos(x)$ and $\sin(x)$, with reduced $x' \in [0,\frac{1}{2}\pi]$, are approximated by p_{3508} and p_{3347}, respectively. The approximations are corrected with the information from step 2.

A separate method is used for step 2, called `trigsub(x)`, which takes as input secure fixed-point number x and output $x' \in [0,\frac{1}{2}\pi]$ as well as b_1, b_2.

Algorithm 5 `trigsub(x)`

Input: fixed-point number x
Output: $z \in [0,\frac{1}{2}\pi]$, $b_1 \in \{0,1\}$, $b_2 \in \{0,1\}$

$\pi \leftarrow \texttt{secfxp}(3.1415926535897932384)$
$p \leftarrow \frac{x}{2\pi}$
$q \leftarrow \texttt{trunc}(p)$ #truncates x
$r \leftarrow x - 2\pi q$
$b_1 \leftarrow \texttt{ge}(r,\pi)$ #if $r \geq \pi$ then $b_1 = 1$ else $b_1 = 0$
$f \leftarrow 2\pi - r$
$w \leftarrow r + b_1(f-r)$
$b_2 \leftarrow \texttt{ge}(w,\frac{1}{2}\pi)$ #if $w \geq \frac{1}{2}\pi$ then $b_2 = 1$ else $b_2 = 0$
$v \leftarrow \pi - w$
$z \leftarrow w + b_2(v-w)$
return z, b_1, b_2

Algorithm	Multiplications	Scalar Multiplications	Randomness	Speed(s)
`two`	$7f$	$l-\log_2 l-1$	f	0.08
`exp`	$7l+5f+2\theta+19$	$5l-3\log_2 l-5$	$4l+(15+2\theta)f$	0.19
`ln`	$5l+8$	$2l-\log_2 l-2$	$l+9f$	0.14
`sqrt`	$8l+14f+14$	$8l-4\log_2 l-8$	$5l+13f$	0.48
`sin`	$3f+17$	$l-\log_2 l-1$	$5l+15f$	0.07
`cos`	$3f+15$	$l-\log_2 l-1$	$5l+13f$	0.07

Table 2: Complexity table: $\theta = \left\lceil \log_2\left(\frac{2f+1}{3.5}\right)\right\rceil$ is the number of iterations performed in the protocol for secure division, l is the bit-length of the fixed-point number and f is the number of fractional bits.

For the protocol of $y = \sin(x)$, the only thing that has to be done is convert the bit into the sign of y and evaluate the approximation polynomial p_{3302}, defined on domain $[0, \frac{1}{2}\pi]$ and whose coefficients are given in [43]. It has a relative error of $10^{-24.33}$ if computed exactly, which corresponds with a precision up to 80 fractional bits.

Algorithm 6 Protocol for securely evaluating $\sin(x)$

Input: fixed-point number x
Output: $\sin(x)$

$w, b_1, b_2 \leftarrow$ `trigsub`(x) #additional method, see Algorithm 5
$v \leftarrow w \cdot \frac{2}{\pi}$ #put w in appropriate domain for p_{3302}
$b \leftarrow 1 - 2 \cdot b_1$ #indicates whether $\sin(2\pi - a) = -\sin(a)$ is used or not
$y \leftarrow v \cdot p_{3302}(v \cdot v)$
return $b \cdot y$

For the protocol of $y = \cos(x)$, the only thing that has to be done is convert the bit into the sign of y and evaluate the approximation polynomial p_{3504}, defined on domain $[0, \frac{1}{2}\pi]$ and whose coefficients are given in [43]. It has a relative error of $10^{-23.06}$ if computed exactly, which corresponds with a precision up to 76 fractional bits.

Algorithm 7 Protocol for securely evaluating $\cos(x)$

Input: fixed-point number x
Output: $\cos(x)$

$w, b_1, b_2 \leftarrow$ `trigsub`(x) #additional method, see Algorithm 5
$b \leftarrow 1 - 2 \cdot b_2$ #indicates whether $\cos(\pi - a) = -\cos(a)$ is used or not
$y \leftarrow p_{3504}(w \cdot w)$
return $b \cdot y$

The behavior of this method is as expected and satisfies the pre-determined goals.

2.2 Randomness Generation

Now that the secure protocols for the building blocks are defined, the next step is to construct protocols for noise generating mechanisms that are differential private. To explain the idea of differential privacy, we summarize the definitions from [19]. After that, three well-studied noise generating mechanisms are treated along with their secure protocols.

Suppose $\mathscr{D}^n$ is some database space. Two datasets $D_1, D_2 \in \mathscr{D}^n$ are called *neighboring* if $d(D_1, D_2) = 1$, i.e., if they differ on one individual's record. For a query q, the following characteristic is defined:

Definition 2.1 (Global Sensitivity) *For a real valued query function $q : \mathscr{D}^n \to \mathbb{R}$, the global sensitivity of q is defined as*

$$\Delta := \max_{D_1, D_2 \in \mathscr{D}^n} \|q(D_1) - q(D_2)\|_1,$$

for all neighboring D_1 and D_2.

Intuitively, the sensitivity stresses the maximum magnitude of information the query is able to leak about one individual in a database, and thus must be hidden. For example, the count function (a function that counts the existence of a certain event in the rows of the database) has global sensitivity 1, because changing one of the rows in the database causes the output to change by either zero or one. A histogram has also sensitivity 1.

Let the query be q, then the *true answer* to that query is $t = q(D)$ and the *response* of $\mathscr{K}$ is defined as the result of the mapping $t \mapsto t + X$. Here, X represents the noise value added to the true answer. Therefore, $\mathscr{K}$ is also called a noise generating mechanism. Now, privacy can be quantified by the following definition:

Definition 2.2 (ε-differential privacy) *A mechanism $\mathscr{K}$ gives ε-differential privacy if for all neighboring databases D_1 and D_2, and all $S \subseteq Range(\mathscr{K})$,*

$$\Pr[\mathscr{K}(D_1) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathscr{K}(D_2) \in S]$$

Here, ε is usually small so that $e^{\varepsilon} \approx 1 + \varepsilon$. Then the distributions of $\mathscr{K}$ on D_1 and D_2 are more or less the same, thus the privacy of the individual represented by the missing entry is preserved. In general, it holds that relative low values for ε result in relative high privacy, but low utility. Relative high values for ε result in relative high utility, but low privacy. Therefore, it is hard to determine what is the middle ground. It is mostly determined by the context, domain and by conducting experiments, looking into the worst case scenario.

2.2.1 Laplace Mechanism

The Laplace mechanism is coined by Dwork et al. [19] and it is proven to preserve ε-differential privacy. The output represents perturbed answers in $\mathbb{R}$. The definition is given as follows:

Definition 2.3 (Laplace mechanism [19]) *For a query function $q : \mathscr{D}^n \to \mathbb{R}$ with sensitivity Δ and privacy parameter ε, Laplace mechanism will output $\mathscr{K}(D) := q(D) + X_{\Delta,\varepsilon}$ where $X_{\Delta,\varepsilon} \sim Lap(\frac{\Delta}{\varepsilon})$ and $Lap(\lambda)$ is the Laplace distribution centered around 0. Its probability density function is given by*

$$f(x) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}}, \forall x \in \mathbb{R}$$

Inverse sampling. In case of the Laplace mechanism and the corresponding Laplace distribution, the cumulative distribution function with parameter $\lambda > 0$ is given as follows:

$$F(x) = \begin{cases} \frac{1}{2} \cdot \exp(\frac{x}{\lambda}), & x < 0 \\ 1 - \frac{1}{2} \cdot \exp(-\frac{x}{\lambda}), & x \geq 0 \end{cases}$$

To determine $F^{-1}(x)$, the equation $F(F^{-1}(x)) = x$ must hold for both cases:

$$\begin{aligned} \frac{1}{2} \exp\left(\frac{F^{-1}(x)}{\lambda}\right) = x \Leftrightarrow \\ \frac{F^{-1}(x)}{\lambda} = \ln(2x) \Leftrightarrow \\ F^{-1}(x) = \lambda \cdot \ln(2x) \end{aligned}$$

and

$$\begin{aligned} 1 - \frac{1}{2} \exp\left(-\frac{F^{-1}(x)}{\lambda}\right) = x \Leftrightarrow \\ -\frac{F^{-1}(x)}{\lambda} = \ln(-2(x-1)) \Leftrightarrow \\ F^{-1}(x) = -\lambda \cdot \ln(2x-2) \end{aligned}$$

Now, $F^{-1}(x)$ can be derived resulting in:

$$F^{-1}(x) = \begin{cases} \lambda \cdot \ln(2x), & x < \frac{1}{2} \\ -\lambda \cdot \ln(2-2x), & x \geq \frac{1}{2} \end{cases}$$

The domain of this function is (0,1) which coincides with the possible values a uniformly random variable can take.

Protocol for Laplace generation. In order to prevent information leakage about the noise value, $F^{-1}(x)$ is rewritten in the following way:

$$\begin{aligned} F^{-1}(x) = \begin{cases} \lambda \cdot \ln(2x), & x < \frac{1}{2} \\ -\lambda \cdot \ln(2-2x), & x \geq \frac{1}{2} \end{cases}, x \in (0,1) \Leftrightarrow \\ F^{-1}(y) = \begin{cases} \lambda \cdot \ln(y), & y < 1 \\ -\lambda \cdot \ln(2-y), & y \geq 1 \end{cases}, y \in (0,2) \Leftrightarrow \\ F^{-1}(y) = \begin{cases} \lambda \cdot \ln(y), & y < 1 \\ -\lambda \cdot \ln(1-y), & y \geq 0 \end{cases}, y \in (0,1) \end{aligned}$$

This is equivalent to generating one element $u \in (0,1)$ and a random bit $b \in \{0,1\}$ and letting b decide which case it will be.

Algorithm 8 Secure protocol for generating Laplace noise

Input: parameter $\lambda = \Delta/\varepsilon$
Output: R.v. $X \sim$ Laplace distribution

$u \leftarrow$ `random(secfxp)`
$b \leftarrow$ `random_bit(secfxp, signed=True)`
$X \leftarrow b \cdot \ln(u)$
return X

2.2.2 Geometric Mechanism

The geometrical mechanism [21] is a discrete variant of the Laplace mechanism, and it is proved to be preserving ε-differential privacy. It is especially useful when applied to statistics that are integers, e.g., counts, ages and amounts. The output represents perturbed answers in $\mathbb{N}$. The definition for specifically a count query is given as follows:

Definition 2.4 (Geometric mechanism [21]) *For a count query $q : \mathscr{D}^n \to \mathbb{N}$ and $\varepsilon \in (0,1)$, geometric mechanism will output $\mathscr{K}(D) := q(D) + X_\varepsilon$ where $X_\varepsilon \sim G(\varepsilon)$ and $G(\varepsilon)$ is a two-sided geometric distribution centered around 0. Its probability distribution function is given by*

$$f(x) = \frac{1-\alpha}{1+\alpha}\alpha^{|x|}, \forall x \in \mathbb{Z} \text{ and } 0 \leq \alpha = e^{-\varepsilon} \leq 1$$

Inverse sampling. An inverse sampling function is derived with input uniformly distributed random variable U and output (two-sided) geometrically distributed random variable $G = i$ if the following inequalities hold for the probability density function $f(x) = \frac{1-\alpha}{1+\alpha}\alpha^{|x|}, \alpha = e^{-\varepsilon}$:

$$\begin{aligned}
\sum_{|x|<i} f(x) \leq U < \sum_{|x|<i+1} f(x) &\Leftrightarrow \\
\sum_{|x|<i} \frac{1-\alpha}{1+\alpha}\alpha^{|x|} \leq U < \sum_{|x|<i+1} \frac{1-\alpha}{1+\alpha}\alpha^{|x|} &\Leftrightarrow \\
1 - \frac{2\alpha^i}{1+\alpha} \leq U < 1 - \frac{2\alpha^{i+1}}{1+\alpha} &\Leftrightarrow \\
i = \pm \left\lfloor \left(\frac{\ln\left(\frac{(1-U)(1+\alpha)}{2}\right)}{-\varepsilon} \right) \right\rfloor &
\end{aligned}$$

The third line follows from:

$$\begin{aligned}
\sum_{x \geq i} \frac{1-\alpha}{1+\alpha}\alpha^{|x|} &= \frac{1-\alpha}{1+\alpha}\alpha^i \sum_{x \geq 0} \alpha^x \\
&= \frac{1-\alpha}{1+\alpha}\alpha^i \frac{1}{1-\alpha} = \frac{\alpha^i}{1+\alpha}
\end{aligned}$$

So $\sum_{|x|<i} \frac{1-\alpha}{1+\alpha}\alpha^{|x|}$ becomes $1 - 2\sum_{x \geq i} \frac{1-\alpha}{1+\alpha}\alpha^{|x|}$ due to symmetry. Also, i could be positive or negative, so we use a random sign bit to decide this case. Implementation of this sampler is given in Algorithm 9.

Algorithm 9 Secure protocol for generating (two-sided) geometric noise

Input: privacy parameters ε

Output: R.v. $X \sim$ (two-sided) geometric distribution

$\alpha \leftarrow e^{-\varepsilon}$

$s \leftarrow$ `random_bit(secfxp, signed=True)`

$U \leftarrow$ `random(secfxp)`

$X \leftarrow s \cdot \texttt{trunc}\left(\frac{\ln\left(\frac{(1-U)(1+\alpha)}{2}\right)}{-\varepsilon}\right)$

return X

2.2.3 Gaussian Mechanism

A relaxed version of ε-differential privacy is (ε, δ)-differential privacy. Intuitively, this form leaves room for straying out of bound. Where ε-differential privacy ensures that every response on some database D is (almost) the same as the response given by the same mechanism on a neighboring database D', (ε, δ)-differential privacy only states that it will be extremely unlikely that $\mathscr{K}(D)$ will result in something else than $\mathscr{K}(D')$. The formal definition is as follows:

Definition 2.5 ((ε, δ)-differential privacy) *A mechanism $\mathscr{K}$ gives (ε, δ)-differential privacy if for all neighboring databases D_1 and D_2, and all $S \subseteq Range(\mathscr{K})$,*

$$\Pr[\mathscr{K}(D_1) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathscr{K}(D_2) \in S] + \delta$$

The Gaussian mechanism. The Gaussian mechanism [18] satisfies (ε, δ)-differential privacy according to Definition 2.5. The protocol takes as input δ, ε and the sensitivity Δ_2 of a query f defined as

$$\Delta_2 = \max_{D_1, D_2} \|f(D_1) - f(D_2)\|_2$$

with $D_1, D_2 \in \mathscr{D}^n$ neighboring datasets. The output represents perturbed answers in $\mathbb{R}$. The definition is given as follows:

Definition 2.6 (Gaussian mechanism [18]) *For $\sigma = \Delta_2 \cdot \frac{\sqrt{2\ln(\frac{1.25}{\delta})}}{\varepsilon}$, Gaussian mechanism will output $\mathscr{K}(D) := q(D) + X_{\Delta_2, \delta, \varepsilon}$ where $X_{\Delta_2, \delta, \varepsilon} \sim \mathscr{N}(0, \sigma^2)$ and $\mathscr{N}(0, \sigma^2)$ is the normal distribtuion centered around 0. Its probability distribution function is given by*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2 / 2\sigma^2}, \forall x \in \mathbb{R}$$

Inverse sampling. For the implementation, one needs to be able to sample from a standard Normal distribution in a fast and secure way. Several algorithms were implemented from the literature. An overview of these implementations are to be found in [43]. The winning sampler is based on the Irwin-Hall distribution: a random variable following the Irwin-Hall distribution is given as $Y = \sum_{i=1}^{k} U_i$. Here, $U_i \sim Unif(0,1)$ has mean $k/2$ and variance $k/12$. For $k = 12$, $Y - 6$ behaves standard normally distributed. This idea is based on the Central Limit Theorem (CLT) where it states that $\sqrt{n}(S_n - \mu) = 1 \cdot (S_1 - 6) = Y - 6 \sim \mathscr{N}(0,1)$. Note that this is a crude approximation as the CLT normally holds for a sequence of random variables. In the following protocol, `normalsampler4()` samples Normal distributed noise as described.

Algorithm 10 Secure protocol for generating Gaussian noise

Input: Sensitivity Δ_2, privacy parameters ε and δ
Output: R.v. $X \sim$ Gaussian distribution

$y \leftarrow$ `normalsampler4()`
$s \leftarrow \Delta_2 \cdot \sqrt{\frac{2\ln(1.25/\delta)}{\varepsilon}}$
$X \leftarrow s \cdot y$
return X

2.3 Combining Differential Privacy with MPC

Two great flavors that go great together: secure multi-party computation solves the problem of computing on private databases without learning their content, and differential privacy solves to problem of disclosing information about the individuals that are contained in databases. A union of secure multi-party computation protocols and differential private techniques would cover the complete process of working with people's data and releasing results to the public in a secure and private way. In case of a database with patient records, the data is then processed in a secure way and the published data will not reveal any information about the patients. An abstract visualization of the setting for three hospitals H1, H2 and H3, each containing datasets of patients, is displayed in Figure 1.

In [44], an abstract definition is given to combine the two concepts. Assume a database, consisting of n rows is shared among m parties $\mathscr{P}_1, ..., \mathscr{P}_m$, $\frac{n}{m} \in \mathbb{N}$. In this setting, an adversary A is passive (i.e., a malicious entity follows the protocol, but tries to retrieve information based on the observed communication), computationally unbounded and controls at most $t \leq m-1$ parties. Furthermore, party $\mathscr{P}_k$ has exactly n/m rows of the dataset, denoted by $D_k = (D_{k,1}, D_{k,2}, ..., D_{k,n/m})$. The goal is for $\mathscr{P}_k$ to ensure the privacy of the rows in D_k against A. Depending on the strategy of A, every piece of communication that A is able to observe while running the protocol $Q = (\mathscr{P}_1, ..., \mathscr{P}_m)$ on input x is denoted by $\text{View}_A(A \leftrightarrow Q(x))$. Assume the worst case and that A passively controls $P_{-k} = (\mathscr{P}_1, ..., \mathscr{P}_{k-1}, \mathscr{P}_{k+1}, ..., \mathscr{P}_m)$, $\text{View}_A(A \leftrightarrow Q(x))$ is determined by all inputs and randomness of all parties other than $\mathscr{P}_k$ and the messages sent by $\mathscr{P}_k$.

Definition 2.7 (Multiparty Differential Privacy) *Let $\mathscr{P}_1, ..., \mathscr{P}_m \in (\mathscr{D}^{n/m})^m$ be parties in a protocol $Q = (\mathscr{P}_1, ..., \mathscr{P}_m)$ with input datasets $(D_1, ..., D_m)$. Q is (ε, δ)-differentially private (in an honest-but-curious model), if for every $k \in \{1, ..., m\}$ and for every two neighboring dataset $D, D' \in (\mathscr{D}^{n/m})^m$, the following holds for every feasible set T:*

$$P[View_{P_{-k}}(P_{-k} \leftrightarrow Q(D))] \in T] \leq e^{\varepsilon} \cdot P[View_{P_{-k}}(P_{-k} \leftrightarrow Q(D')) \in T] + \delta$$

Take the hospitals in Figure 1. In this example, a database containing records of patients on treatment X is distributed over a set of three hospitals $H = \{H_1, H_2, H_3\}$. Suppose a malicious entity E has control over two of them, say $H' = \{H_1, H_2\}$. All hospitals engage in a (ε, δ)-differentially private protocol Q. Then $V = \text{View}_{H'}(H' \leftrightarrow Q(D))$ is the communication E is able to observe among H' and to and from H_3 with database $D \in (\mathscr{D}^{n/m})^m$. The goal of E is to recover (a piece of) the database of H_3 by observing V. The goal of H_3 is then to ensure that for any two neighboring databases $D, D' \in (\mathscr{D}^{n/m})^m$ possessed by H_3, the adversary will not be able to tell which of these two is used as input in the communication of the hospitals.

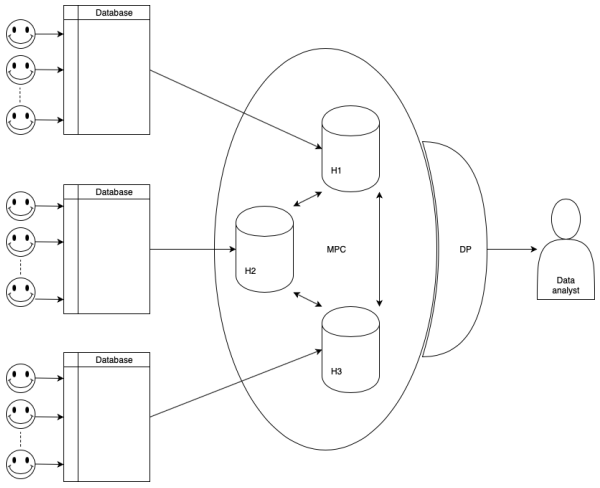


Figure 1: Abstract communication of the hospitals

2.3.1 Security Requirements and Challenges

The idea of using both MPC and differential privacy in one program takes care of a couple of problems: there is no reason for data to be stored at one specific site, noise only has to be added one time (to the MPC result) instead of adding noise locally at each party, and it is possible to publish statistics in a privacy-preserving manner. This results in the following security guarantees given in [44]:

- *Correctness*: Every party receives the correct output of the model.
- *Data privacy*: No information about the data in the databases held by the parties expect for the output of the protocol itself is revealed.
- *Individual privacy*: No information about the individuals contained in the private databases held by the parties is revealed.

Still, there are some challenges, even with the use of MPC and differential privacy combined. In case an honest-but-curious model is assumed, it does not matter if a party is corrupted, all parties follow the protocol honestly. Only the communication is observed. What if this is not the case? Corrupted parties may deviate from the protocol, generating values out of proportion. This could result in either bad privacy or bad utility. This phenomenon is also called *answer pollution*.

Also, a problem arises when multiple parties are colluding: sensitive data could be endangered if multiple parties work together to get information about the secret data. Moreover, the noise itself may be disclosed as well.

Furthermore, it is even harder to quantify the tradeoff between privacy and utility: in general the more corrupted parties, the more noise has to be added and therefore the less accurate is the result. In order to guarantee the privacy for all scenarios, one should assume all other parties to be corrupted, but this may lead to a big perturbation of the data. Lowering the number of corrupted parties not only affects the privacy guarantees, but also enforces to agree on how many of the parties are corrupted (which is not a realistic situation in practice).

Another challenge is to scale the hybrid model to more parties, provide the ability to use other mechanisms and improve the overall efficiency.

2.4 Conclusions

The goal is to generate randomness in a secure way, such that the result can be used to ensure the privacy of people's data. This goal translates to exploring the possibilities of combining techniques in MPC and differential privacy.

In order to construct differential privacy algorithms, building blocks are needed for the randomness generation. They have been designed and implemented in Section 2.1. The implementations are analyzed to work optimally for the MPC setting. All work is done in the MPyC framework which may include further specific optimizations using the capabilities of the particular framework. Most implementations rely on a numerical approximation of a specific function evaluation in a certain domain. Scaling the input of such an implementation to this domain is then inevitable. Using the information about the scaling leads to straight forward protocols for secure function evaluation for any input from the feasible domain. Using approximations in the implementations results in applying MPC primitives optimally. This approach sets an example for further development of secure protocols for other complex functions.

Section 2.2 presents the construction of the protocols for the differential privacy algorithms. First, the notion of sampling from a distribution is explained. Then the implementation of these samplers are combined with the theory from the literature to design secure noise generating mechanisms that satisfy the differential privacy guarantees.

After that, Section 2.3 shows the two concepts can work together and the theory behind the combination is described extensively. Combining MPC and differential privacy does have its benefits. However, adding differential privacy may not be necessary at all times. An individual's record is easily hidden in a relatively big database. Querying the database will not be likely to violate privacy in this case, so adding noise here would only increase running time of the protocols and ruining the utility of the results. This is also the case if only a small amount of queries is asked. Furthermore, when databases do not contain sensitive data, adding differential privacy would be completely redundant.

It is highly recommended to use the combination of MPC and differential privacy for sensitive data that is distributed amongst multiple entities for which privacy is important as well. Then for suitable parameter and protocol choices, the combination of both techniques could be a solution for preserving privacy while maintaining utility.

3 Differentially Private Logistic Regression

In this chapter, we present a technique to build a logistic regression classifier based on multiple sensitive datasets from mutually distrusting data providers. Existing work on this problem is either inefficient, leaks sensitive information, or is not accurate enough [36, 45, 41, 28]. By combining multi-party computation and differential privacy, we overcome these problems.

3.1 Background

3.1.1 Privacy-Preserving Data Mining

MPC and differential privacy are two well-known techniques to enable privacy-preserving data mining, i.e., performing analytics based on sensitive data. Whereas MPC focusses on processing and protecting sensitive information from workers, differential privacy focusses on the processing result and protecting sensitive information from the recipient of the analytics outcome.

In MPC, processing takes place in a distributed way between multiple workers, with cryptographic techniques guaranteeing that no worker learns any information about the underlying data except the computation output. Many constructions are known that differ in the number of workers supported and the conditions under which privacy and correctness are guaranteed. To improve the performance of MPC, it can help to open intermediate values. One classical example is in the case of an iterative algorithm, where after each iteration it is revealed whether the stopping criterion has been reached. Otherwise, in order not to reveal the number of iterations performed, the algorithm would always need to run for the maximum number of iterations that may be needed.

In differential privacy, noise is added to an analytics outcome in order to limit the amount of information that the outcome can give about any single record in the dataset. The amount of leakage permitted is captured by a privacy budget, typically denoted ε. Every time when disclosing a value that depends on sensitive information, part of this budget is spent. The more budget spent on disclosing a particular value, the less noise needs to be added.

Since MPC protects processing and differential privacy protects its output, it is natural to combine them into one overall approach that protects both. The idea is that, instead of using MPC to compute the regular analytics outcome, we use MPC to compute the analytics outcome with differential privacy noise added. This is discussed, e.g., in [18, 6] and the previous chapter.

3.1.2 Logistic Regression and Privacy

Logistic regression aims to construct a binary classifier for records with numerical attributes. Given a dataset with k-valued records, the classifier is a vector $\vec{\beta} = (\beta_1, \ldots, \beta_k)$ and a record $(x_1, \ldots, x_k)$ is classified according to whether

$$\tilde{y}(\vec{\beta}; \vec{x}) := 1/(1 + \exp(-\beta_1 x_1 - \ldots - \beta_k x_k)) > T,$$

for threshold T[8]. The task of logisitic regression is to find the best such classifier for a given dataset. This task is typically phrased as an optimization problem that includes a regularization parameter $\lambda \geq 0$ used to reduce overfitting. This optimization problem can be solved with Newton iteration, that is, by iteratively improving an approximation to $\vec{\beta}$ until convergence is reached.

[8] Typically, the dataset contains one attribute with constant value, whose corresponding β_i coefficient is called the intercept.

Algorithm 11 Logistic regression by Newton iteration

Require: $\mathbf{X}_i, \vec{y}_i$: datasets/outputs for party i; λ: regularization parameter
Ensure: $\vec{\beta}$: logistic regression classifier for given dataset

1: **function** Logreg($\mathbf{X}_i, \vec{y}_i, \lambda$)
2: **parties** i **do** $\mathbf{H}_i \leftarrow \frac{1}{4}(\mathbf{X}_i)^T \mathbf{X}_i$ ▷ Hessian
3: $\mathbf{H} \leftarrow \sum_i \mathbf{H}_i - \lambda \mathbf{I}$
4: $\vec{\beta} \leftarrow (0, \ldots, 0)$
5: **while** not converged **do**
6: **parties** i **do for** $k = 1, \ldots$ **do** $(\vec{l}_i)_k \leftarrow \sum_j ((\vec{y}_i)_j - \tilde{y}\left(\vec{\beta}; (\mathbf{X}_i)_j\right))(\mathbf{X}_i)_{j,k}$
7: $\vec{l} \leftarrow \sum_i \vec{l}_i - \lambda \vec{\beta}$ ▷ gradient
8: $\vec{\beta} \leftarrow \vec{\beta} - \mathbf{H}^{-1}\vec{l}$ ▷ new approximation of $\vec{\beta}$
9: **return** $\vec{\beta}$

MPC allows to train a logistic regression classifier based on sensitive data from multiple mutually distrusting data owners. Several earlier works have described MPC-based implementations [28, 45, 41] of logistic regression that all implement the above iterative approach, while apparently leaking the intermediate approximations of $\vec{\beta}$ (although these works are not very explicit about this). [33] presents the first fully privacy-preserving MPC-based implementation of logistic regression.

Differential privacy can be used to ensure that a logistic regression classifier $\vec{\beta}$ does not leak information about individual records. This can be achieved either by computing $\vec{\beta}$ normally and then adding noise, or by adding noise to the optimization problem and then solving this modified optimization problem [10, 11]. (Note that these approaches assume a nonzero regularization parameter.) One work also suggests to use differential privacy instead of MPC to train a logistic regression classifier on sensitive data from multiple sources [25]. Deferring details to the next section, this work is based on Newton iteration; in each iteration the different parties provide contributions based on their sensitive data with differential privacy noise added.

3.2 Privacy-Preserving Logistic Regression

The state-of-the-art techniques for privacy-preserving logistic regression discussed above are all based on approximating $\vec{\beta}$ with Newton iteration. A basic implementation of this approach is shown in Algorithm 11.

As discussed above, it is possible to perform logistic regression in a fully private way [34]. However, in this case Algorithm 11 needs to be performed fully under MPC, and in particular, $\tilde{y}$ in line 6 needs to be evaluated under MPC for every item in the dataset. This is computationally expensive.

Several existing works [28, 45, 41] that perform logistic regression with MPC are based on the idea of keeping successive approximations of the classifier $\vec{\beta}$ open, as shown in Algorithm 12[9]. The central observation is that, if $\vec{\beta}$ is public, then in line 6 of the algorithm, each party can compute its contribution $[\vec{l}_i]$ to the gradient. As a consequence, the computation in the encrypted domain does not depend on the number of items in the respective datasets, and hence is much faster. However, the disadvantage of this approach is that these intermediate versions of the classifier leak information about the underlying sensitive data; in particular, they are not differentially private.

[9] Values in brackets are secret. By **parties** i **do** $[a] \leftarrow b$ we mean that each party can locally compute b and secret-shares it with the other parties as secret value $[a]$.

Algorithm 12 Logistic regression by Newton iteration

Require: $\mathbf{X}_i$, $\vec{y}_i$: datasets/outputs for party i; λ: regularization parameter
Ensure: $\vec{\beta}$: logistic regression classifier for given dataset

1: **function** Logreg($\mathbf{X}_i, \vec{y}_i, \lambda$)
2: **parties** i **do** $[\mathbf{H}_i] \leftarrow \frac{1}{4}(\mathbf{X}_i)^T\mathbf{X}_i$ ▷ Hessian
3: $[\mathbf{H}] \leftarrow \sum_i [\mathbf{H}_i] - \lambda\mathbf{I}$
4: $\vec{\beta} \leftarrow (0, \ldots, 0)$
5: **while** not converged **do**
6: **parties** i **do for** $k = 1, \ldots$ **do** $[(\vec{l}_i)_k] \leftarrow \sum_j ((\vec{y}_i)_j - \tilde{y}\left(\vec{\beta}; (\mathbf{X}_i)_j\right))(\mathbf{X}_i)_{j,k}$
7: $[\vec{l}] \leftarrow \sum_i [\vec{l}_i] - \lambda\vec{\beta}$ ▷ gradient
8: $\vec{\beta} \leftarrow \vec{\beta} - [\mathbf{H}^{-1}][\vec{l}]$ ▷ new approximation of $\vec{\beta}$
9: **return** $\vec{\beta}$

As discussed, differentially private logistic regression on distributed datasets is possible without the use of MPC [25]. In this approach, instead of using a Hessian $\mathbf{H}$ based on the sensitive datasets, it is assumed that a public dataset is available from which $\mathbf{H}$ is computed. Similarly to the MPC-based solution above, parties locally compute $\vec{l}_i$ based on the current approximation of β, but now they make this value differentially private by adding noise instead of inputting it to the multi-party computation. This approach has two disadvantages. First, using public $\mathbf{H}$ requires a public dataset to be available, and reduces accuracy compared to computing $\mathbf{H}$ from the full dataset. Second, adding differentially private noise to the values $\vec{l}_i$ means that the privacy budget for the algorithm needs to be distributed over many disclosures; moreover, because each individual dataset is relatively small compared to the full dataset, the noise that needs to be added to achieve differential privacy is relatively large.

3.3 Approach

The core idea is to combine the private and public approaches for logistic regression by opening up intermediate values of $\vec{\beta}$ while adding differentially private noise to make sure that these intermediate values do not leak information about the underlying dataset. Compared to the third approach, less values need to be opened, and moreover, these values are aggregated over the full dataset rather than subdatasets. Both have as a consequence that less noise can be added per disclosure, making the final result more accurate.

This approach overcomes the disadvantages of the above three approaches. Compared to the first approach, we avoid the need to perform MPC-based computations on the full dataset, as in the second approach. However, unlike the second approach, we do guarantee differential privacy of disclosed values. Unlike the third approach, however, we improve accuracy by allowing $\mathbf{H}$ from the real dataset to be used, and reducing the amount of noise that needs to be added.

3.4 Logistic Regression with MPC and Differential Privacy

Algorithm 13 implements above approach. Secret-shared real numbers can be represented by fixed-point numbers. As in earlier works, in practice the value $[\mathbf{H}^{-1}]$ in line 9 is not actually computed, but instead, the Cholesky decomposition of $[-\mathbf{H}]$ is computed once and $[\mathbf{H}^{-1}][\vec{l}]$ is repeatedly computed by back substitution as described in [36].

Unlike above, we fix the number of iterations to be performed as a parameter N. There are three

Algorithm 13 Logistic regression with MPC and differential privacy

Require: $\mathbf{X}_i$, $\vec{y}_i$: normalized datasets/outputs for party i; λ: regularization parameter; ε: differential privacy parameter (aka privacy budget); N: number of iterations

Ensure: $\vec{\beta}$: logistic regression classifier for given dataset

1: **function** Logreg($\mathbf{X}_i, \vec{y}_i, \lambda, \varepsilon, N$)
2: **parties** i **do** $[\mathbf{H}_i] \leftarrow \frac{1}{4}(\mathbf{X}_i)^T\mathbf{X}_i$ ▷ Hessian
3: $[\mathbf{H}] \leftarrow \sum_i [\mathbf{H}_i] - \lambda\mathbf{I}$
4: $\vec{\beta} \leftarrow (0, \ldots, 0)$
5: **for** $it = 1, \ldots, N$ **do**
6: **parties** i **do for** $k = 1, \ldots$ **do** $[(\vec{l}_i)_k] \leftarrow \sum_j ((\vec{y}_i)_j - \tilde{y}\left(\vec{\beta}; (\mathbf{X}_i)_j\right))(\mathbf{X}_i)_{j,k}$
7: $[\vec{l}] \leftarrow \sum_i [\vec{l}_i] - \lambda\vec{\beta}$ ▷ gradient
8: $[\vec{n}] \leftarrow \mathsf{GenNoise}(\sum_i |\mathbf{X}_i|, \varepsilon/N)$ ▷ noise
9: $\vec{\beta} \leftarrow \vec{\beta} - [\mathbf{H}^{-1}][\vec{l}] + [\vec{n}]$ ▷ new approximation of $\vec{\beta}$
10: **return** $\vec{\beta}$

reasons for this. First, the number if iterations needs to be known beforehand to decide how to divide the privacy budget ε between disclosures. Second, the algorithm will not actually converge since in every iteration different random noise is added. Finally, the amount of noise to be added increases with the number of iterations, so it is profitable to reduce the number of iterations compared to the original algorithm.

As shown in Line 8, a noise vector $[\vec{n}]$ is computed and then added to the new value for $\vec{\beta}$ before opening it in line 9. Here, $\mathsf{GenNoise}(M, \varepsilon)$ represents a function generating differentially private noise for $\vec{\beta}$ given dataset size M and privacy budget ε. This can be implemented, as suggested by [10, 11], by drawing $\vec{n}$ from a Laplacian distribution, i.e., by drawing a random norm from the $\Gamma(d, 2/(M\varepsilon\lambda))$ distribution and a uniformly random direction, and letting $\vec{n}$ be the vector with that norm and direction (here, d is the number of attributes in the dataset). The second parameter for the gamma distribution also reflects the upper bound for sensitivity for logistic regression: $2/M\lambda$. We remark that this approach assumes that the inputs are normalized, i.e., each record $\vec{x} \in \mathbb{R}^n$ has Euclidean norm ≤ 1. Drawing the random vector needs to be performed under MPC, which can be done along the lines of [18]. Particularly, to implement this random direction sampling may use a normal distribution with parameters 0 and 1 to generate a vector full of Normal distributed elements and subsequently multiply the normalized vector with the norm in order to obtain the desired norm. Note that GenNoise is called with privacy budget ε/N, which is the overall privacy budget ε divided by the number of openings.

3.5 Discussion

The approach presented here for performing logistic regression combines various ideas from literature. Below we discuss in more detail how the result relates to and improves on these works.

"Privacy-Preserving Ridge Regression on Hundreds of Millions of Records" [36] is the best work in this direction that we used. In this work, intermediate versions of the model are kept closed so this is fully secure (but as a consequence, the size of the MPC scales in the size of the dataset). Our implementation of back-substitution (line 8 of Algorithm 2) is as described in their section "C. Solving a Linear System in a Circuit".

In "PrivLogit: Efficient Privacy-preserving Logistic Regression by Tailoring Numerical Optimizers" [45], a similar approach is proposed. Our exact formulas for computing the logistic regression

model (including a regularization parameter) were taken from here, and our idea of simply opening intermediate values of the model was inspired by their description of logistic regression in a kind of "aggregate-and-combine" way in their Algorithm 1. In this work, homomorphic encryption is used, which allows intermediate models to remain closed since the update formulas can be phrased in such a way inputters only need to perform linear operations on the intermediate model using their plain data. In theory an approach like this could work, but the paper does not exactly specify what encrypted data is sent to the inputters and why this results in only linear operations from their side (this is not clear immediately from the logistic regression formulas and indeed seems non-trivial). Also, from their protocol descriptions it seems in their practical implementation, they simply open intermediate results ("Our protocols do disclose the regression coefficients (...) to distributed Nodes"). They do not reveal exactly how this can be done.

In "Secure Multi-pArty Computation Grid LOgistic REgression (SMAC-GLORE)" [41], a similar approach is used. This work does not open intermediate results, but instead claims to achieve the effect of locally performing computations on the full dataset by somehow using an expansion of the exponential function to make sure the party only needs to perform linear operations ("The first derivative of the maximum likelihood function"). However, exactly why this allows all computations to be performed locally is not clear.

Finally, in "Supporting Regularized Logistic Regression Privately and Efficiently" [28], a similar approach to the previous one is given, which claims to achieve full privacy with a kind of aggregate-and-combine approach where parties apply an encrypted estimate of the model to their dataset in order to locally compute an updated estimate of the model. Also, here, crucial details are missing, and the implementation is not fully secure: the paper says that they "take a pragmatic approach to security" without saying which part they actually implemented securely and which part they did not. At least it seems they did not securely implement matrix inversion, which we did.

4 Controlling Leakage in MPC on Non-Integer Types

Secure multi-party computation (MPC) is an increasingly popular solution to performing joint computation on some data while keeping the data private. In short, MPC involves a number of participants each holding some private input who want to compute a function of the joint input, but without revealing individual inputs to any other party.

The common way of arguing security for an MPC protocol is to define an ideal functionality, and show that the real protocol does not leak any more than what is leaked in the ideal functionality. However, the ideal functionality itself may leak something due to the nature of an MPC protocol. This can be seen in practice if we instantiate an MPC protocol using non-integer types, such as floating-point numbers. For instance, a simple example is that $0.1 \times 10 \neq 0.5 \times 2$ if we use the floating-point approximation of each real number in the equation. This means that in this case, any MPC protocol used to realize a simple addition $f(x_1, \ldots, x_{10}) = \sum_{i=1}^{10} x_i$ will be vulnerable to an input distinguishability attack, since $f(0.1, 0.1, \ldots, 0.1) \neq f(0.5, 0.5, 0, \ldots, 0)$, so an attacker who only observes the output will be able to distinguish between these two possible sets of inputs.

This input distinguishability attack exists in essentially all practical implementations of MPC using non-integer types, including important applications like privacy-preserving machine learning (which naturally uses non-integer types), but is rarely acknowledged. This means that these implementations do not achieve the standard security notion of an MPC protocol, and using them in an application could lead to unintended information leakage.

Our Contribution. We propose a protocol secure against input distinguishability attacks and hence can achieve secure MPC on non-integer types. Our security definition builds upon the definition introduced by Feigenbaum et al. [20]. We then construct a protocol that, given a function f with domain $\mathbb{R}$, computes a function f' with domain $\mathscr{X} \subset \mathbb{R}$ such that f' is a good approximation of f, but the computation of f' does not leak any more to an adversary than if the adversary was given the output of f instead. We show how to instantiate our protocol using the SPDZ [16, 14, 26] MPC protocol. This chapter is based on ongoing, joint work by SODA researchers Prastudy Fauzi, Claudio Orlandi and Peter Scholl. We omit the full details and formal proofs from this high-level overview.

4.1 Preliminaries

Each value x involved in a computation has domain $\mathscr{X}$ such that $\mathscr{X} \subset \mathbb{R}$ and has a representation of fixed length. Our model considers secure computation based on secret sharing. The secret sharing corresponding to a variable x will be denoted by $[\![x]\!]$. We use the standard arithmetic black box model of MPC [15], which allows performing a basic set of arithmetic operations on secret shared values.

4.2 Machine Learning and Truncation Problems

In machine learning, many functions that need to be computed have output in the real numbers, and so computation is typically done using representations of real numbers. For example, the sigmoid activation function commonly used in neural networks takes an input $x \in \mathbb{R}$ and outputs $y \in (0,1)$. Hence, one important factor to consider in doing MPC for machine learning is how to represent real numbers. As any representation is merely an approximation using a finite amount of bits, there is a possibility of errors related to operations done on secret-shared data which must be accounted for. These operations commonly involving truncation, where a number that can be accurately represented in k bits needs to be stored in only $k' < k$ bits.

4.2.1 Representations of Real Numbers

Golden ratio In the golden ratio representation (used by, e.g., Dimitrov et al. [17]), numbers are represented as $(a,b) := a - \phi b$, where a,b are integers and $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio number. Note that since $\phi^2 = \phi + 1$,

$$\begin{aligned}(a,b)+(c,d) &= (a+c,b+d)\\ (a,b)*(c,d) &= (ac+bd,bc+ad-bd)\ .\end{aligned}$$

For a real number x, (a,b) is a (k,ε) representation of x if $|a| \leq 2^{-k}$, $|b| \leq 2^{-k}$ and $|a-\phi b-x| \leq \varepsilon$.

After multiplication, the values (a,b) used to represent the result grows exponentially fast. In [17], this is solved by using a normalization process that uses (a',b') with $|a'| << a$ and $|b'| << b$ but $|(a,b)-(a',b')|$ is small. To help in this step, they used several representations $(2^i,[\frac{2^i}{\phi}])$ of 0 with different magnitudes.

Unfortunately, this step introduces an input distinguishability attack. One simple one is to consider two different sets of input on the same functionality, but both result in 0 represented in different magnitudes (i.e., $(2^i,[\frac{2^i}{\phi}])$ and $(2^j,[\frac{2^j}{\phi}])$ with $i \neq j$).

To visualize this, consider $f(x,y,z) := xy - z^2$. Let a,b,c be real numbers such that $ab - c^2 = 0$. Then while $f(a,b,c) = f(2a,2b,2c)$, during the normalization step the representation of one is close to $(2^i,[\frac{2^i}{\phi}])$, while the other is close to $(2^{i+1},[\frac{2^{i+1}}{\phi}])$. So inputs are easily distinguishable based on the magnitude of the representation of 0.

Fixed point numbers [8, 9] In the fixed point number representation (used in [8, 9]), numbers are represented as $m \cdot 2^{-f}$, where m is an integer and f is a positive integer that determines the *resolution*. Basic operations add, multiply and inner product have absolute error at most 2^{-f}.

Computing the reciprocal uses the Newton-Rhapson method and has absolute error at most 2^{-p} to get $(p+1)$ bits. We assume fixed-point division has absolute error 2^{-f}.

4.2.2 Probabilistic Rounding / Truncation

Operations that use truncation (scaling down) include multiplication with scaling and inner product. This is related to using rounding to obtain the value in the representation with resolution f which is closest to the actual value of a computation.

The truncation protocol of Catrina and de Hoogh [8] works as follows. Given $m, [\![a]\!]$, with $m, a \in \mathbb{Z}$, $TruncPr([\![a]\!],m)$ outputs $[\![d]\!]$ where

$$d = [\frac{a}{2^m}] + b$$

for some random bit b that is 1 with probability $\frac{a \mod 2^m}{2^m}$. Hence the protocol rounds $\frac{a}{2^m}$ to the nearest integer with probability $1-\alpha$, where α is the distance of $\frac{a}{2^m}$ to the nearest integer.

Unfortunately, truncation using $Trunc$ can introduce input distinguishability attacks. For instance, for multiplication one set of inputs will not need to be truncated, but another set does. Set $a = 3, b = 5 \cdot 2^{-2}, c = 2^{2-f}$ and $a' = 3 \cdot 2^{2-f}, b' = 5 \cdot 2^{-3}, c' = 2$. Then

$$(a*b)*c = (a'*b')*c'$$

but $Mult(Mult([\![a]\!],[\![b]\!]),[\![c]\!]) = [\![15 \cdot 2^{-f}]\!]$ while $Mult(Mult([\![a']\!],[\![b']\!]),[\![c']\!]) = [\![16 \cdot 2^{-f}]\!]$.

Probablistic truncation by $TruncPr$ will have more pronounced differences, as there is possibly an error in the last bit, not dependent on rounding. We note that this type of attack is not exclusive to

the truncation protocol of Catrina and de Hoogh; it can also be done on the probabilistic truncation protocols in SecureML [35] and ABY3 [32].

4.2.3 Linear and Logistic Regression

A natural extension of the previous attack can be seen in the problem of regression, a task which is often run inside MPC.

Linear regression. Given a set of data points $D = \{(x_i, y_i)\}_i$, the goal of linear regression is to find a line $y(x) = ax + b$ that approximates the relationship between the two variables x and y. A possible requirement for privacy-preserving linear regression is as follows: suppose two data sets D_1 and D_2 result in linear regressions y_1 and y_2 that intersect at x. Given $(x, y_1(x)$ (= $(x, y_2(x))$, it should not be possible to determine if it came from D_1 or D_2.

We show that this is not satisfied against active adversaries in SecureML [35] and ABY3 [32]. Suppose that shares are in $\mathbb{Z}_{2^k}$, and consider the two lines $y_1(x) = -2^{k-2}x + 2^{k-1}$, $y_2(x) = 0$ that intersect at $x = 1/2$. While $[\![y_2(1/2)]\!] = [\![0]\!]$, $[\![y_1(1/2)]\!] = [\![-2^{k-2} \cdot 1/2 + 2^{k-1}]\!] \neq [\![0]\!]$ due to 2^{k-1} being an edge case for truncation in SecureML.

We can avoid this type of attack by either using integer types or by doubling the precision of non-integer types, in both cases sacrificing efficiency.

Logistic regression. Logistic regression can be seen as an extension to linear regression, where the line $y(x)$ is then mapped to $z(y) \in [0,1]$, and from there a classification $k \in \{1, \ldots, n\}$. For $n = 2$, the classifications can be seen as $k = 1$ if, for a fixed ε, $y(x) + \varepsilon > 0$, and $k = 2$ if $y(x) + \varepsilon \leq 0$.

Consider the same scenario as in linear regression, where y_1 and y_2 intersect at x. Due to the previous attack, $z_1(y_1(x)) \neq z_2(y_2(x))$. Hence if the cutoff point for classification is in the interval $[z_1(y_1(x)), z_2(y_2(x))]$ one can distinguish y_1 and y_2.

We note that in both linear and logistic regression, the error is so small that it is not sufficient to launch a practical attack. However, we cannot rule out the possibility that these theoretical attacks can be amplified into something more serious.

4.3 Our Construction

In the next two sections we explore some possibilities to fix the existing MPC protocols to be secure against input distinguishability attacks.

1. A naive solution would be to just add more precision, e.g., a fixed number of additional bits for the fractional part of the data type. While the probability of reaching the edge cases of (say) truncation would be smaller, in general it could still be possible for a malicious party to craft inputs that make use of these edge cases to launch an input distinguishability attack.

   One could take this to the extreme, so that the output of every multiplication gate doubles the precision of the inputs to the gate. One could either adjust the domain of computation after every gate, or, assuming the multiplication depth is known beforehand, set the domain of inputs and all intermediate values to match the precision required for the final output. While this can give a provably secure scheme, since there is no need for truncation and we are simply performing the exact real-valued computation throughout the entire circuit, this is highly inefficient in practice.

2. Inspired by the work of Feigenbaum et al. [20], one can use the naive solution but approximate an error bound (due to probabilistic rounding, truncation, etc) of the output. Before opening the output, the parties add noise proportional to the error bound (and dependent on the statistical security parameter). This idea will be the focus of the current section.

3. The above idea can be improved by approximating error bounds of each intermediate result, using annotated secret sharing. Before opening any intermediate values, the parties add noise proportional to the error bound associated with the intermediate value's annotation. As a consequence, the error bound for the output will be more accurate. We disucss this approach in Section 4.4.

4.3.1 Feigenbaum et al. Definition

We start with the following theorem of Feigenbaum et al. [20].

Theorem 4.1 *Let f be a multivariate function from integers to the reals with short symbolic description. Suppose, for any integer k and any x, one can compute a value $\tilde{f}(x) = f(x) \pm 2^{-k}$ in time $(|x|+k)^{O(1)}$. Then there exists a function g, from integers to finite-precision reals (i.e., integer multiples of fixed small unit), such that the following properties hold.*

1. *(Good approximation.) For all x, $g(x) = f(x) \pm 2^{-k}$.*
2. *(Efficiency.) $g(x)$ is computable in time $(|x|+k)^{O(1)}$.*
3. *(Functional privacy, in a modified sense.) There is a simulator, S, such that, for any family $\{\rho_j\}$ rounding functions that take real values to finite-precision real values satisfying $|\rho_j(x) - x| \leq 2^{-j}$, we have $S(\rho_j(f(x))) \equiv_s g(x)$.*

We claim that the above theorem also holds for real functions with short symbolic description, and in particular for fixed point arithmetic.

We can thus add input privacy to the constructions in the previous section with the following general idea:

1. Compute the upper bound of error ε in a secure approximation protocol.
2. Let the function f be approximated by $\tilde{f}$ where $|f - \tilde{f}| < \varepsilon$.
3. Then we can get a secure computation $\hat{f} = \tilde{f} + D_\varepsilon$.

The problem with this approach is that it may not be efficient to compute $\tilde{f}$. In our model, we will consider classes of functions f that can be represented as an arithmetic circuit consisting of fixed-point additions and multiplications, with multiplicative depth d. In this case, we observe that the approximation $\tilde{f}$ can be computed with reasonable efficiency.

4.3.2 Instantiating the Feigenbaum et al. Secure Approximation in SPDZ

We will implement the above idea and combine it with the SPDZ [16, 14] MPC protocol. We assume that the function can be evaluated as a fixed-point arithmetic circuit C. We assume pairwise authenticated channels between the parties involved in the MPC.

Protocol 1.

Setup. Let the arithmetic circuit C have depth $d = poly(\kappa)$, where κ is the security parameter. Let the inputs be floating point numbers represented as t bit integers with precision f. Choose a prime field F_p where p is a $t + g(d)f + \kappa$ bit prime and $g(\cdot)$ is a function of the circuit depth. For general circuits we take $g(d) = d^2$.[10] Let the inputs be fixed point numbers with precision f, but be parsed as $m \cdot 2^{-(g(d)f+\kappa)}$ (i.e., we add $(g(d)-1)f + \kappa$ bits of precision).

Preprocessing phase. This follows the standard SPDZ preprocessing, to produce the desired number of random bits and multiplication triples.

Online phase. Each share $[\![a]\!]$ will have an associated depth d_a, such that the domain of $[\![a]\!]$ will have precision $d_a f + \kappa$.

- On $add([\![a]\!], [\![b]\!], d_a, d_b)$: parties perform local addition of their shares, as in normal additive secret sharing. The output will have associated depth $\max\{d_a, d_b\}$.
- On $mult([\![a]\!], [\![b]\!], d_a, d_b)$: parties perform the SPDZ multiplication protocol, truncated to precision $(d_a + d_b)f + \kappa$. The output will have associated depth $d_a + d_b$

Output. For $output([\![y]\!])$, the output can be seen as an integer with precision $g(d)f + \kappa \leq d^2 f + \kappa$. The parties sample a κ-bit $\varepsilon \in [-2^{-f}, 2^{-f} - 1]$ uniformly at random and denote the shares as $[\![\varepsilon]\!]$. The parties then locally compute $[\![y + \varepsilon]\!] = add([\![y]\!], [\![\varepsilon]\!])$, and finally open $y + \varepsilon$.

4.3.3 Security

This approach can be seen as a specialized version of the general approach of Feigenbaum et al. [20] Security can be proven similarly to the results in [20], and we obtain the following theorem.

Theorem 4.2 *Let f be the function approximated by Protocol 1. For a given input x, let $g(x)$ be the output of Protocol 1. Then the function g has the following properties.*

1. *(Good approximation.) For all x, $g(x) = f(x) \pm 2^{-f}$.*
2. *(Efficiency.) $g(x)$ is computable in time $(|x| + k)^{O(1)}$.*
3. *(Functional privacy, in a modified sense.) There is a simulator, S, such that, for any family $\{\rho_j\}$ rounding functions that take real values to finite-precision real values satisfying $|\rho_j(x) - x| \leq 2^{-j}$, we have $S(\rho_j(f(x))) \equiv_s g(x)$.*

4.3.4 Efficiency

This approach performs reasonably well when the multiplicative depth d is small, however, performance degrades with d due to the high precision requirements.

4.4 Optimizing Using Annotated Secret Sharing

In this section we describe another method that allows for lower precision requirements for more complex classes of functions.

[10] Note that this can be optimized for certain circuits. For example if every multiplication gate always involves at least one input value, we only need $g(d) = d$.

4.4.1 Annotated Secret Sharing

In annotated secret sharing, a secret sharing corresponding to x is additionally annotated with values $\mathsf{lb}(x)$ and $\mathsf{ub}(x)$ that represent known lower and upper bounds on the range of x, and with an imprecision level $\mathsf{imp}(x)$.

4.4.2 Secure Function Evaluation Using Annotated Secret Sharing

Let π be an n-party protocol that implements an approximation $\tilde{f}$ of an m-variate function f. The protocol π takes as input annotated secret sharing $([\![x_i]\!], \mathsf{lb}(x_i), \mathsf{ub}(x_i), \mathsf{imp}(x_i))$ for $i \in [m]$ and outputs an annotated secret sharing $([\![y]\!], \mathsf{lb}(y), \mathsf{ub}(y), \mathsf{imp}(y))$. The correctness requirements for π are:

- The output is such that $y = \tilde{f}(x_1, \ldots, x_m)$.
- If for all i it holds that $\mathsf{lb}(x_i) \leq x'_i \leq \mathsf{ub}(x_i)$, then $\mathsf{lb}(y) \leq f(x'_1, \ldots, x'_m), \tilde{f}(x'_1, \ldots, x'_m) \leq \mathsf{ub}(y)$.
- If for all i it holds that $\mathsf{lb}(x_i) \leq \tilde{x}_i, x'_i \leq \mathsf{ub}(x_i)$ and $\|\tilde{x}_i - x'_i\| \leq \mathsf{imp}(x_i)$, then $\|\tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_m) - f(x'_1, \ldots, x'_m)\| \leq \mathsf{imp}(y)$.

Sub-protocols computing multiple functions can be arbitrarily composed, but the secret shared values should not be opened by them. The central idea for guaranteeing the security of the approximate computation is that noise is added to the secret shared values prior to any opening. Given an annotated secret sharing $([\![z]\!], \mathsf{lb}(z), \mathsf{ub}(z), \mathsf{imp}(z))$ as input, the noise generator algorithm π_{ng} produces an output $[\![w]\!]$ that can be opened by the parties. The security requirements on π_{ng} are:

- For all $\mathsf{lb}(z) \leq z_0, z_1 \leq \mathsf{ub}(z)$ such that $\|z_0 - z_1\| \leq \mathsf{imp}(z)$ and for $[\![w_0]\!] \xleftarrow{\$} \pi_{\mathsf{ng}}([\![z_0]\!], \mathsf{lb}(z), \mathsf{ub}(z), \mathsf{imp}(z))$ and $[\![w_1]\!] \xleftarrow{\$} \pi_{\mathsf{ng}}([\![z_1]\!], \mathsf{lb}(z), \mathsf{ub}(z), \mathsf{imp}(z))$, w_0 and w_1 should be statistically indistinguishable (as a function of a statistical security parameter λ).

Before we resume, we show the problems when using the constructions of [17] and [8]. Recall that numbers are represented as $m \cdot 2^{-f}$, where $m \in [-2^{k-1}+1, 2^{k-1}-1]$. Suppose we want to compute $f(x,y) = x \cdot y$. Then $\mathsf{lb}(x) = \mathsf{lb}(y) = (-2^{k-1}+1)2^{-f}$ and $\mathsf{ub}(x) = \mathsf{ub}(y) = (2^{k-1}-1)2^{-f}$ are the smallest and biggest numbers in the domain, respectively, and the imprecision level is $\mathsf{imp}(x) = \mathsf{imp}(y) = 2^{-f}$. Let (x,y) and (x',y') be such that $|x - x'| \leq 2^{-f}$ and $|y - y'| \leq 2^{-f}$. Then $f(x,y) - f(x',y') \leq (2^{k-1}-1)2^{-f} + 2^{-2f} = \mathsf{imp}(f)$. This means that to guarantee security, a large noise needs to be added, which may ruin the usefulness of the function to be computed. However, for some classes of functions we can get by with a much smaller amount of noise, compared with the previous approach.

Protocol 2. As in the previous section, we will combine the annotated secret sharing idea with SPDZ.

Setup. Let the arithmetic circuit C have depth $d = poly(\kappa)$, where κ is the security parameter. Let the inputs be floating point numbers represented as t bit integers with precision f. Choose a prime field F_p where p is a $t + df + \kappa$ bit prime.

Let the inputs be have initial annotation $([\![x]\!], \mathsf{lb}(x), \mathsf{ub}(x), \mathsf{imp}(x) = 0)$, where $L := -2^{t-f} \leq \mathsf{lb}(x) \leq \mathsf{ub}(x) \leq U := (2^t - 1)2^{-f}$. Denote by $[\![\vec{x}]\!]$ the tuple $([\![x]\!], \mathsf{lb}(x), \mathsf{ub}(x), \mathsf{imp}(x))$.

Preprocessing phase. As before, this follows the SPDZ preprocessing phase.

Online phase.

- On $add([\![\vec{a}]\!],[\![\vec{b}]\!])$: parties perform local addition of their shares, as in normal additive secret sharing. The output $[\![\vec{c}]\!]$ will have $\mathsf{lb}(c) = \max\{L, \mathsf{lb}(a)+\mathsf{lb}(b)\}, \mathsf{ub}(c) = \min\{U, \mathsf{ub}(a)+\mathsf{ub}(b)\}, \mathsf{imp}(c) = \mathsf{imp}(a)+\mathsf{imp}(b)$.
- On $mult([\![\vec{a}]\!],[\![\vec{b}]\!])$: parties perform the SPDZ multiplication protocol, truncated to precision $\log_2 \mathsf{imp}(a)\cdot\mathsf{imp}(b)$. The output $[\![\vec{c}]\!]$ will have $\mathsf{lb}(c) = \max\{L, \mathsf{lb}(a)\cdot\mathsf{lb}(b)\}, \mathsf{ub}(c) = \min\{U, \mathsf{ub}(a)\cdot\mathsf{ub}(b)\}, \mathsf{imp}(c) = \mathsf{imp}(a)\cdot\mathsf{imp}(b)$.
- On $open([\![\vec{a}]\!])$: parties sample $\varepsilon \in D_{\mathsf{imp}(a)}$ uniformly at random obtaining shares $[\![\varepsilon]\!]$. Then compute $[\![a+\varepsilon]\!] = [\![a]\!]+[\![\varepsilon]\!]$, and open $a+\varepsilon$.

Output. For $output([\![\vec{y}]\!])$, simply perform $open([\![\vec{y}]\!])$.

4.4.3 Efficiency

We first note that our proposed definition works well for linear functions, e.g., $f(x,y,z) = ax+by+cz$. This is because $\mathsf{imp}(f) \le a\cdot\mathsf{imp}(x)+b\cdot\mathsf{imp}(y)+\le c\cdot\mathsf{imp}(z) \le (|a|+|b|+|c|)\max\mathsf{imp}(x,y,z)$. For higher degree polynomials, we will again have the same problems as in 4.3.2, i.e., $\mathsf{lb}(x), \mathsf{ub}(x), \mathsf{imp}(x)$ will increase. However, not all functions will require an increase in precision. For instance, consider a circuit containing sub-circuits performing an equality test or comparison function on fixed-point inputs. When using the protocols based on SPDZ for these operations, the imprecision in the output of the sub-circuit no longer depends on the complexity of the sub-circuit itself, since these protocols essentially proceed by performing a bit decomposition of the input and emulating a Boolean computation. The sub-circuit itself therefore has no imprecision, and the resulting precision loss in the output only depends on the imprecision of the inputs, which can lead to much better estimates.

4.5 Future Directions: Entropy-Based Solution

The vulnerabilities of existing MPC protocols for non-integer types stem from the fact that the security model often does not even consider them valid attacks. Hence, one interesting discussion is on how to correctly model such leakage of information based on the floating-point representation used by the computing parties. A natural way of seeing this is in terms of entropy lost by opening output values in an MPC with non-integer types. Such a model could allow us to run computations with a much smaller precision, whilst still obtaining a good bound on the amount of information leakage that occurred, which should in practice be quite small. An additional benefit is that this approach seems more amenable to composition, and much easier to plug into existing protocols without a large degradation in security.

References

[1] *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017.

[2] SCALE MAMBA 1.2, 2018. https://homes.esat.kuleuven.be/~nsmart/SCALE/.

[3] MPyC 0.5, 2019. https://github.com/lschoe/mpyc.

[4] Robert J Aumann. Game theory. In *Game Theory*, pages 1–53. Springer, 1989.

[5] P. D. Azar, S. Goldwasser, and S. Park. How to incentivize data-driven collaboration among competing parties. In *ITCS*, pages 213–225, 2016.

[6] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 451–468, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[7] Simina Brânzei, Claudio Orlandi, and Guang Yang. Sharing information with competitors. *CoRR*, abs/1809.10637, 2018.

[8] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 134–150. Springer, 2010.

[9] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2010.

[10] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 289–296. Curran Associates, Inc., 2008.

[11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

[12] Y. Chen, K. Nissim, and B. Waggoner. Fair information sharing for treasure hunting. In *AAAI*, pages 851–857, 2015.

[13] Y. Chen and B. Waggoner. Informational substitutes. In *FOCS*, pages 239–247, 2016.

[14] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany.

[15] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[16] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[17] Vassil Dimitrov, Liisi Kerik, Toomas Krips, Jaak Randmets, and Jan Willemson. Alternative implementations of secure real numbers. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 553–564, Vienna, Austria, October 24–28, 2016. ACM Press.

[18] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

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

[20] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006.

[21] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM J. Comput.*, 41(6):1673–1693, 2012.

[22] J. Halpern and V. Teague. Rational secret sharing and multiparty computation. In *STOC*, pages 623–632. ACM, 2004.

[23] John F. Hart. *Computer Approximations*. Krieger Publishing Co., Inc., Melbourne, FL, USA, 1978.

[24] S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. In *FOCS*, pages 585–594, 2005.

[25] Zhanglong Ji, Xiaoqian Jiang, Shuang Wang, Li Xiong, and Lucila Ohno-Machado. Differentially private distributed logistic regression using private and public data. *BMC Medical Genomics*, 7(1):S14, May 2014.

[26] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 549–560, Berlin, Germany, November 4–8, 2013. ACM Press.

[27] G. Kol and M. Naor. Cryptography and game theory: Designing protocols for exchanging information. In *TCC*, pages 320–339, 2008.

[28] Wenfa Li, Hongzhe Liu, Peng Yang, and Wei Xie. Supporting regularized logistic regression privately and efficiently. *CoRR*, abs/1510.00095, 2015.

[29] R. Mantena, R. Sankaranarayanan, and S. Viswanathan. Platform-based information goods: The economics of exclusivity. *Decis. Support Syst.*, 50(1):79–92, December 2010.

[30] R. McGrew, R. Porter, and Y. Shoham. Towards a general theory of non-cooperative computation. In *TARK*, pages 59–71. ACM, 2003.

[31] P. B. Miltersen, J. B. Nielsen, and N. Triandopoulos. Privacy-enhancing auctions using rational cryptography. In *CRYPTO*, pages 541–558, Berlin, Heidelberg, 2009. Springer-Verlag.

[32] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[33] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017* [1], pages 19–38.

[34] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[35] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017* [1], pages 19–38.

[36] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, pages 334–348, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[37] K. Nissim, C. Orlandi, and R. Smorodinsky. Privacy-aware mechanism design. In *ACM EC*, pages 774–789. ACM, 2012.

[38] A. Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[39] I. Segal and M. Whinston. Exclusive contracts and protection of investments. *RAND Journal of Economics*, 31(4):603–633, 2000.

[40] Lloyd S Shapley. A value for n-person games. *The Shapley value*, pages 31–40, 1988.

[41] Haoyi Shi, Chao Jiang, Wenrui Dai, Xiaoqian Jiang, Yuzhe Tang, Lucila Ohno-Machado, and Shuang Wang. Secure multi-party computation grid logistic regression (SMAC-GLORE). *BMC Med. Inf. & Decision Making*, 16(S-3):89, 2016.

[42] Y. Shoham and M. Tennenholtz. Non-cooperative computation: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 343(1):97–113, 2005.

[43] Kimberley Thissen. Achieving Differential Privacy in Secure Multiparty Computation. Master's thesis, Eindhoven University of Technology, the Netherlands, 2019.

[44] Salil P. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography.*, pages 347–450. 2017.

[45] Wei Xie, Yang Wang, Steven M. Boker, and Donald E. Brown. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. *CoRR*, abs/1611.01170, 2016.