MASTER

Achieving differential privacy in secure multiparty computation

Thissen, K.K.A.

Award date:
2019

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain.
Achieving Differential Privacy in Secure Multiparty Computation

Author: K.K.A. Thissen

Supervisors:
Dr. Ir. L.A.M. Schoenmakers
Ir. R.P. Koster
Ir. P.P. van Liesdonk

Eindhoven University of Technology
Department of Mathematics and Computer Science

Philips Research

May 15, 2019
Abstract

Collecting data about people’s health contributes to useful medical research, adapting health care programs or improving overall knowledge about us, human beings. Data is stored in databases, which could be queried by various organizations and institutions. As the databases are likely to contain sensitive data, it would be a violation of privacy if this information is disclosed somehow. In particular, statistics gathered from these databases are subject to disclosing private information. Differential privacy is a technique to preserve the privacy of the individuals in these situations, by generating randomness and perturbing statistics by adding noise. Secure multiparty computation techniques distributes the computation of the statistics and the noise for a more robust privacy guarantee. This thesis explores possibilities for generating randomness in a secure multiparty computation setting in order to generate the noise accordingly.
## Contents

1 Introduction ......................................................... 1  
   1.1 Motivation ................................................. 1  
   1.2 Contributions ............................................. 2  
   1.3 Roadmap .................................................... 2  

2 Differential Privacy .................................................. 3  
   2.1 Why do we need differential privacy? ......................... 3  
      2.1.1 Statistical disclosure control ............................... 3  
      2.1.2 Methods for preserving privacy ......................... 4  
   2.2 Private data analysis ....................................... 6  
      2.2.1 Definitions ............................................ 6  
      2.2.2 The interpretation of $\epsilon$ and $\delta$ ............... 9  
      2.2.3 Properties of differential privacy ..................... 9  
      2.2.4 Noise generation mechanisms ............................ 10  
      2.2.5 Utility metrics ........................................ 12  
      2.2.6 Challenges in differential privacy ...................... 12  

3 Secure Multiparty Computation ..................................... 14  
   3.1 Why do we need secure multiparty computation? ............... 14  
   3.2 Secure distributed data analysis ............................ 14  
      3.2.1 Threshold cryptography .................................. 16  
      3.2.2 Secret sharing and resharing ............................. 16  
      3.2.3 Protocols ............................................. 17  
      3.2.4 Security requirements .................................. 17  
      3.2.5 Involved parties ...................................... 17  
      3.2.6 Examples ............................................. 18  
   3.3 MPyC .......................................................... 19  
      3.3.1 Datatypes .............................................. 19  
      3.3.2 Computations in MPyC ................................... 19  

4 Achieving differential privacy using secure multiparty computation 21  
   4.1 How can we join forces between differential privacy and secure multiparty computation? 21  
   4.2 Private distributed data analysis ............................ 21  
   4.3 Security requirements and challenges ....................... 23  
   4.4 Existing frameworks and applications ...................... 24
Combining Differential Privacy with MPC

C  Supporting methods  70
   C.1  Truncation  70
   C.2  Evaluating $2^n$ for positive integers  70
   C.3  Modified scaling method  71
   C.4  Exponential distribution sampler  71
   C.5  Bernoulli distribution sampler  72
   C.6  Geometric distribution sampler  72

D  Inverse standard Normal sampler  73
   D.1  Acceptance-rejection method  73
   D.2  Box-Muller  73
   D.3  Marsaglia  74
   D.4  Irwin-Hall  74
   D.5  Approximation  75

E  UCI adult data description  76
Chapter 1

Introduction

Nowadays, privacy matters make headlines more often than before. The world is adapting fast to this new realization that privacy is not trivial and that violations of privacy come in many shapes and forms. In particular information leakage from processed data, e.g., summary statistics, histograms and other results, is not directly visible. This type of data should be released with caution.

Many methods have been tried before to deal with the privacy of people in databases. An intuitive solution for this is to strip names and other personal information from the database. Netflix did this for a contest [21]: by releasing a set of existing user data without their actual identity, people were challenged to come up with an algorithm to predict user ratings for movies. However, researchers showed that it was possible to link the data back to the users with the help of the IMDB database [20]. This is an example of linkage attacks: connecting anonymized databases with non-anonymized databases so that records could be de-identified. In the end, databases are used as a means for other purposes, such as research. Results based on this data should not change whether a person is included in the database or not. The idea behind this is differential privacy, coined by Dwork et. al. [8]. 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 causing noise on the results using so-called noise generating mechanisms.

Suppose some results that are based on multiple databases, distributed over multiple entities. If noise is added to each of the individual database results, then the utility of the end results is likely to be destroyed. Here, secure multiparty computation (MPC) is proposed as a solution. Multiple parties jointly compute on their databases in order to get the relevant results. Then, noise is added only one time. This results in the interesting matter how to implement noise generating mechanisms in an MPC setting, with the end goal in mind that the technology will get closer to real life implementations and applications for arbitrary databases in order to ensure overall privacy.

1.1 Motivation

A project of interest covers both concepts with the main focus on secure multiparty computation for data analytics and, to a limited extend, differential privacy. The name of the project is SODA [29]: Scalable Oblivious Data Analytics. This project recognizes that analytics on databases with sensitive information could violate privacy. Since this is especially true for
health care data, the project chose health care as their use case. Three important goals are set for achieving a working solution. At first, techniques for MPC-based data processing has to be improved. This would motivate hospitals to share their knowledge for further improvement of services without actually revealing their own data. But while MPC might help the hospitals with, e.g., computing statistics on the data securely, it would not help the patients the moment those statistics are released publicly. This formulates the second goal: develop techniques that provide the same sense of privacy for the patients as for the hospitals. One solution that is proposed is applying differential privacy to MPC results, which ensures the privacy of the patients. It will lead to an improved feeling of privacy and an increased willingness to allow processing their data. The third goal is eventually delivering functional and secure frameworks. This thesis contributes to this goal by working on one of the available frameworks, MPyC, by designing and implementing the necessary mathematical functions.

1.2 Contributions

After extensive research, an inventory is made for the necessary secure protocols to deploy differential privacy, including exp(·), ln(·), √·, cos(·) and sin(·). All protocols are designed to perform optimally in an MPC setting.

The literature proposes many candidates for noise generating mechanisms that are proven differential private. This thesis covers a couple of mechanisms to investigate their potential in the MPC setting. The underlying algorithms have been optimized with respect to performance speed. The output is validated and an analysis shows the applicability of the mechanisms.

Several use cases illustrate the application of differential privacy and its challenges. Using specific examples and available databases, the impact of choices for sensitivity, mechanisms and privacy budget is made visible. Also, a worked out example combines differential privacy with secure multiparty computation.

1.3 Roadmap

This thesis is organized as follows:

Chapter 2, 3 and 4 give preliminaries on the concepts differential privacy, secure multi party computation and how their combination is defined. These chapters present the motivation, along with formal definitions, clear examples and challenges.

Chapter 5 provides the necessary tools for randomness generation. This includes secure protocols for evaluating functions. The behavior of these protocols is analyzed on their validity, accuracy, speed and complexity. These protocols are necessary for what comes next.

Chapter 6 investigates several ways for randomness generation. The theory is used to efficiently implement noise generating mechanisms. They have a special property, as they are proven to be $\epsilon$-differential private. All corresponding protocols are based on literature, but some optimizations are applied for the MPC setting.

Chapter 7 puts differential privacy into practice by means of use cases.

Chapter 8 concludes this project by giving a summary of results and some suggestions for further research.
Chapter 2

Differential Privacy

This chapter describes the motivation behind differential privacy. Furthermore, the concept of statistical disclosure control is explained and some other existing measures for that as well. Then all relevant definitions are stated and in the end, the challenges of differential privacy are given.

2.1 Why do we need differential privacy?

Differential privacy cares about the protection of the privacy of individuals by preventing the possibility of leaking the content of their data. Collecting data in various forms has been a phenomenon for ages. It is used for improvement of processes, products and profits, and overall areas, such as healthcare, crime-fighting and user experiences. Because data is invaluable, many companies and organizations are collecting it as much as possible without considering potential consequences. While there are many good causes, it is of utmost importance that the data cannot harm people by invading their privacy. At risk are the sensitive data like age, income, likes, sexual orientation and medical history. Disclosing this type of data could lead to gossip, discrimination or financial and even physical harm.

Leaking this information is not always directly visible. History shows that anonymized data, i.e., data without personal information such as names and addresses, are subject to disclosing sensitive information as well. The Netflix Prize Contest is a famous example of this: given anonymized user data by Netflix, researchers were able to assign the identity of users with the help of another public database. It is shown that this could also happen with aggregated data: e.g., statistics, numbers, figures, etc. Institutions that possess this kind of data should be aware of the potential consequences when intentionally or unintentionally releasing it, no matter what form. Differential privacy could be of help with releasing data in a controlled manner.

2.1.1 Statistical disclosure control

Compromised data is a recurrent issue nowadays and in particular issues like failure of access control, data breaches, leaks or stolen data. But even if all these issues would be resolved, ensuring privacy remains a challenge with respect to performing statistical analysis on data. Revealing accurate statistics about a set of respondents while preserving privacy of the individuals, is called statistical disclosure control. Three aspects of disclosure are involved:
1. Identity disclosure: the possibility of linking a data record with a data subject.

2. Attribute disclosure: the possibility of determining a new attribute of a data subject.

3. Inferential disclosure: the possibility of a more accurate idea of an attribute of a data subject.

Two recent examples show that statistical disclosure control is not a trivial concept:

- In 2010, Aleksandra Korolova [17] found out that Facebook enabled tailored targeted advertisement in such detail that it could come down to one single person. This way an advertiser could learn a user’s private information even though the user chose not to display it publicly (e.g., age and sexual orientation).

- In 2014, it was possible to de-anonymize taxi databases about fares, tips, time, date, distance, GPS coordinates etc. This was done by linking information based on paparazzi pictures to the database and that way, people learned how celebrities tipped their driver [32].

### 2.1.2 Methods for preserving privacy

Although this thesis covers differential privacy as leading method to enforce statistical disclosure control, other approaches have been mentioned by the literature.

#### Restricted query answering

Disallowing specific queries like asking for an individual’s information is another frequent suggestion to protect the privacy of the respondents. This will lead to failure once the adversary has knowledge about the existence of someone’s record in a certain database. In that case, the adversary could leak this record in only two queries: the first query is evaluated on all records in the database, and the second query is evaluated on all records except for the victim’s. If this leakage contains sensitive information, like having a criminal record, this breach could have serious consequences for the victim as other institutions may adapt their decisions about the victim based on this information.

A countermeasure is the notion of *query auditing*: each query is evaluated in context of the query history to determine if a response could violate privacy and if so, the query is refused. Then the problem arises that refusing itself already reveals too much about the sensitive information and furthermore, query monitoring is computationally infeasible [16].

#### Data modification

Data modification is applied such that it is safe to release a database without violating privacy. An easy way to do this is by removing attributes that could personally identify individuals in the database, such as names, social security numbers and addresses. This way, the data could be released while the individuals stay anonymous. This works only when no additional information is available that could result in linking databases (*linkage attack*). But since there are many big databases publicly available these days, linking attacks are realistic scenarios. Hence, releasing raw data this way is not a robust method for statistical disclosure.

Another form of data modification is called *k-anonymity* [30]. It simply states that an individual in a set of *k* individuals cannot be distinguished from at least *k* − 1 others in a
set of data, and it is established using quasi-identifiers. The best way to understand it is
by means of an example, shown in Table 2.2. Here, the original entries of Table 2.1 are
replaced by the quasi-identifiers. It is applied to the non-sensitive attributes in such a way
that the sensitive attributes are not to be traced back to the individual. Unfortunately, this
method has a drawback: in case of records having the same quasi-identifiers and a common
sensitive attribute, re-identification is possible. Considering Table 2.2, if the adversary has
the additional information that his 24-years-old friend Bob living in ZIP area 6412 is in the
database, it is easy to deduce that Bob has a heart disease.

<table>
<thead>
<tr>
<th>Non-sensitive</th>
<th>Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip Code</td>
<td>Age</td>
</tr>
<tr>
<td>5611</td>
<td>22</td>
</tr>
<tr>
<td>5612</td>
<td>22</td>
</tr>
<tr>
<td>5613</td>
<td>26</td>
</tr>
<tr>
<td>5611</td>
<td>41</td>
</tr>
<tr>
<td>5614</td>
<td>47</td>
</tr>
<tr>
<td>5614</td>
<td>44</td>
</tr>
<tr>
<td>6404</td>
<td>21</td>
</tr>
<tr>
<td>6412</td>
<td>22</td>
</tr>
<tr>
<td>6412</td>
<td>28</td>
</tr>
</tbody>
</table>

Table 2.1: Original database

<table>
<thead>
<tr>
<th>Non-sensitive</th>
<th>Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip Code</td>
<td>Age</td>
</tr>
<tr>
<td>56**</td>
<td>20-30</td>
</tr>
<tr>
<td>56**</td>
<td>20-30</td>
</tr>
<tr>
<td>56**</td>
<td>20-30</td>
</tr>
<tr>
<td>56**</td>
<td>40-50</td>
</tr>
<tr>
<td>56**</td>
<td>40-50</td>
</tr>
<tr>
<td>56**</td>
<td>40-50</td>
</tr>
<tr>
<td>64**</td>
<td>20-30</td>
</tr>
<tr>
<td>64**</td>
<td>20-30</td>
</tr>
<tr>
<td>64**</td>
<td>20-30</td>
</tr>
</tbody>
</table>

Table 2.2: 3-anonymized database

**Synthetic database**

A *synthetic database* consist of fake data based on the real data. The entries of a synthetic
database are generated with a statistical model: the data follow a certain distribution. On a
large scale, results of analyses are comparable. The benefits are that the data is fake, i.e. the
records resemble no specific individual, the set can be analyzed countless times without any
implications for the privacy of the individuals of the original set and thus, it can be shared
or made public. There exist differential private methods to generate these kind of databases,
but they have not proven to be optimal yet [5].
Perturbation

Further modifications to a database are *rounding*: original numerical values are replaced by their nearest integer, and *swapping*: swap individual values in a database at random or first a certain set is sorted on value and then swapped within a restricted range. Adding noise to values in datasets or to the query output also falls under this method, and will be explained in detail next.

2.2 Private data analysis

Differential privacy is a solution to the paradox of learning nothing about an individual while learning useful information about a population. Differential privacy ensures that the same conclusions will be reached independent of whether any individual opts in or out of the dataset. In this section, an in-depth description for differential privacy is given.

Suppose some data collector gathers (sensitive) information from a certain group of people, the *sample*. From this sample one can learn *statistics* about the underlying population and publish them publicly. In doing so, information might leak about specific individuals in the sample. So the challenge is to maximize the utility of the statistics while preserving privacy of the individual. A way to preserve privacy is to generate *noise*, a value following a certain distribution that can be added to a statistic. This noise could potentially perturb the statistics in a way that the result cannot be used to support some hypothesis. Therefore it is important to understand the fundamental trade-off between privacy and utility.

2.2.1 Definitions

*Differential privacy* is a formal framework to quantify the trade-off between privacy and the accuracy of query results. The idea is that the absence of a single row in a database should not affect the statistical results significantly. This implies worst case privacy guarantees: if an adversary is in possession of all knowledge about all individuals (contained in the database) except for person $X$ (also contained in the database), the adversary is not able to learn new information about $X$. In the following sections, all definitions are from [8].

The situation is described as follows: the data collector, also known as the *curator*, pledges to the data subject, any *individual* in a sample, that their contribution to the data set will never affect them in any way. With or without the individual’s contribution, the same conclusions will be drawn from the data by the *data analyst*. See Figure 2.1 for a schematic overview of their communication using differential privacy.

A typical database $D$ consists of a number of rows (*records*), the elements of $D$. Intuitively, each row represents an individual from the sample. Each entry of an arbitrary record corresponds to an *attribute*: descriptive information about the individual. The data analyst requests a statistic; a function acting on the database. This function is called a *query*. The queries may or may not be known to the curator beforehand, depending on the setting (*interactive* versus *non-interactive*). When the queries are fixed in advance, the non-interactive setting will result in the best accuracy of the results, since the noise is based on the structure of the queries. The noise is generated by some query-answering algorithm $K : D^n \rightarrow R$. Here, $D^n$ is the feasible set of databases with $n$ individuals and $R$ some output space of $K$. For databases $D_1, D_2 \in D^n$, the following characteristic is defined:
Definition 2.2.1 (The Hamming distance). The Hamming distance \( d(D_1, D_2) \) between two datasets is the number of entries in which they differ:

\[
d(D_1, D_2) = |\{i : D_1(i) \neq D_2(i)\}|
\]

Two datasets are called neighboring if \( d(D_1, D_2) = 1 \), i.e., if the difference is the presence of one individual’s record. For a query \( q \), the following characteristics are defined:

Definition 2.2.2 (Global Sensitivity). For a real valued query function \( q : D^n \rightarrow \mathbb{R} \), the global sensitivity of \( q \) is defined as

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

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. A histogram has also sensitivity 1.

Some functions have relatively large sensitivities, adding too much noise to the true answers and making the responses useless for further conclusions, see Section 2.2.6. Therefore, [23] proposed the concept of local sensitivity:

Definition 2.2.3 (Local Sensitivity). For a real valued query function \( q : D^n \rightarrow \mathbb{R} \) and some \( D_1 \in D^n \), the \( \ell_1 \)-local sensitivity of \( q \), i.e. the local sensitivity with respect to the \( \ell_1 \)-norm, is defined as

\[
LS_q(D_1) := \max_{D_2 \in D^n} \|q(D_1) - q(D_2)\|_1,
\]

for all neighboring databases \( D_1 \) and \( D_2 \).

Note that \( \Delta = \max_{D_1} LS_q(D_1) \). Local sensitivity adds noise not only dependent on the function but also proportional to the entries of the database. The downside is that this could potentially reveal too much about the database. In general it is hard to analyze the sensitivity of a function, so instead [25] proposes a sensitivity sampler that computes the sensitivity based on the sensitivity results for some 'random' related databases.
A query-answering algorithm takes as input a value along with some parameters and outputs the noisy version of the value, with the noise following some distribution. The underlying algorithm is therefore not deterministic, but randomized:

**Theorem 2.2.1** (Randomized algorithm). A randomized algorithm $\mathcal{K} : A \rightarrow B$ is associated with mapping $M : A \rightarrow \Delta(B)$ and $\Delta(B)$ is called the probability simplex over $B$, i.e.,

$$ \Delta(B) = \left\{ x \in \mathbb{R}^{|B|} : (\forall i, x_i \geq 0) \land \sum_{i=1}^{|B|} x_i = 1 \right\} $$

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

**Definition 2.2.4** ($\epsilon$-differential privacy). A mechanism $\mathcal{K}$ gives $\epsilon$-differential privacy if for all neighboring databases $D_1$ and $D_2$, and all $S \subset \text{Range}(\mathcal{K})$,

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

This is classical differential privacy, stating that there is an upper bound $e^\epsilon$ to the multiplicative distance of the probability distributions of the randomized query outputs for any two neighboring databases. To illustrate this concept more clearly, suppose the true answer is $t$. Then mechanism $\mathcal{K}$ is applied to $t$ to output noisy response $m$. The actual noise added to $t$ is $m - t$. Now, $D$ and $D'$ are neighboring datasets differing one record, so if $t - 1$ would have been the true answer and $m$ the noisy response, then the noise added is $m - t + 1$. Since it should not matter whether a record is included or not in the databases, the probability of these events should be approximately the same. Hence, to preserve differential privacy,

$$ \Pr[\text{noise} = m - t] \approx \Pr[\text{noise} = m - t + 1] \iff \frac{\Pr[\text{noise} = m - t]}{\Pr[\text{noise} = m - t + 1]} \approx 1 $$

To quantify the approximation, the bounds on the quotient are expressed as:

$$ e^{-\epsilon} \leq \frac{\Pr[\text{noise} = m - t]}{\Pr[\text{noise} = m - t + 1]} \leq e^\epsilon \iff 1 - \epsilon \lesssim \frac{\Pr[\text{noise} = m - t]}{\Pr[\text{noise} = m - t + 1]} \lesssim 1 + \epsilon $$

Note that this only holds for small epsilon. This makes sense as the bigger $\epsilon$ gets, the more likely the probabilities will deviate for each other and thus information could be deduced.

A relaxed version of $\epsilon$-differential privacy is $(\epsilon, \delta)$-differential privacy. Intuitively, this form leaves room for straying out of bound. Where $\epsilon$-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 $D'$, $(\epsilon, \delta)$-differential privacy only states that it will be extremely unlikely that $\mathcal{K}(D)$ will result in something else than $\mathcal{K}(D')$. The formal definition is as follows:

**Definition 2.2.5** ($(\epsilon, \delta)$-differential privacy). A mechanism $\mathcal{K}$ gives $(\epsilon, \delta)$-differential privacy if for all neighboring databases $D_1$ and $D_2$, and all $S \subset \text{Range}(\mathcal{K})$,

$$ \Pr[\mathcal{K}(D_1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{K}(D_2) \in S] + \delta $$
2.2.2 The interpretation of $\epsilon$ and $\delta$

In general, it is hard to quantify the trade-off between the utility and the privacy loss of the result provided by a noise generating mechanism. The parameter $\epsilon$ is able to give a rather relative indication: when $\epsilon$ is small, then $\epsilon$-differential privacy asserts that given neighboring databases $D_1, D_2$ and output $O$ of the mechanism, the adversary cannot tell which database resulted in $O$. When $\epsilon$ is large, an adversary could be able to identify certain outputs as very unlikely or could exclude certain (neighboring) databases to occur in practice. Nevertheless, the probability of leaking information remains present. So when $\epsilon \in \mathbb{R}$ is fixed, failure in preserving $\epsilon$-differential privacy might result in, the best case, effectively meaningless privacy loss or, in the worst case, complete revelation of the entire database.

Furthermore, $\epsilon$ could be seen as the representation of the concept of privacy budget. The privacy budget applies when one performs sequential querying on a certain database via the same mechanism. In general, sequential querying of differential private mechanisms degrades the overall privacy level. A mechanism $K(D)$ having the $\epsilon$-differential privacy property, inherits some privacy loss. If $m$ questions are asked by a data analyst but the data curator wants a total privacy loss of $\epsilon$, then each question must be answered by a mechanism that has the property of $\frac{\epsilon}{m}$-differential privacy. This is one of the most powerful features of differential privacy: it is closed under composition. Two important composition theorems:

**Theorem 2.2.2** (Sequential composition). Let $K_i$ be an $(\epsilon_i, 0)$-differentially private mechanism for $i \in M = \{1, ..., m\}$ and $D$ some database from $\mathcal{D}^n$. Let $K_M(D) = (K_1(D), ..., K_m(D))$. Then $K_M$ is $(\sum_{i=1}^{m} \epsilon_i, 0)$-differentially private.

**Theorem 2.2.3** (Parallel composition [22]). Let $K_i$ be an $(\epsilon_i, 0)$-differentially private mechanism and let $D_i$ be arbitrary disjoint subsets of the input domain $\mathcal{D}^n$ for $i \in M = \{1, ..., m\}$. Let $K_M(D) = (K_1(D_1), ..., K_m(D_m))$. Then $K_M$ is $(\max_{i \in M} \{\epsilon_i\}, 0)$-differentially private.

The composition theorem assumes the worst-case scenario: the same amount of leakage happens with each new response to a query. The data curator can enforce a maximum privacy loss. If the number of queries exceeds the threshold, then the privacy guarantee becomes too weak and the curator stops answering queries.

When a mechanism is $(\epsilon, \delta)$-differentially private with $\delta > 0$, a common interpretation is that the mechanism is $\epsilon$-differentially private "except with probability $\delta$". From the definition, for every neighboring $D_1, D_2$ and output $O$, an adversary could deduce that output $O$ is more or less likely to be generated when the database is $D_1$ than it was with database $D_2$. In practice, an increase of the value of $\delta$ results in having a higher probability for relative small noise added to the result (where an increase of the value of $\epsilon$ results in having a bigger range of possible noise values). This is clearly seen in the results of the Gaussian mechanism in Section 6.5.

2.2.3 Properties of differential privacy

Next to the fact that differential privacy is closed under composition, it has two other desirable properties:

**Privacy loss**

Differential privacy has a measure of privacy loss and this permits comparing different methods in a meaningful way. Fixing the bound on privacy loss, then comparing accuracy or vice versa,
enables to quantify the trade-off between privacy and utility. Formally, privacy loss is defined as follows:

**Definition 2.2.6** (Privacy loss [8]). Let $D_1, D_2 \in \mathcal{D}^n$ be neighboring databases. The privacy loss incurred by observation $\xi$, generated by mechanism $\mathcal{K}$, is defined as

$$L^{(\xi)}_{\mathcal{K}(D_1)\|\mathcal{K}(D_2)} := \ln \left( \frac{\Pr[\mathcal{K}(D_1) = \xi]}{\Pr[\mathcal{K}(D_2) = \xi]} \right)$$

For $\mathcal{K}$ $(\epsilon, \delta)$-differentially private it holds that

$$\Pr \left[ L^{(\xi)}_{\mathcal{K}(D_1)\|\mathcal{K}(D_2)} > \epsilon \right] \leq \delta$$

In words, this guarantees $\epsilon$-differential privacy on some $1 - \delta$ quantile of the distribution of $\mathcal{K}$. The proof of the existence of this quantile is given in [8], but the proof does not give the exact location of the quantile.

**Post-processing**

It is not possible to make an $\epsilon$-differential private mechanism less differential private or make it increase privacy loss by evaluating any function of the response of the mechanism given that there is no additional information about the database.

**Theorem 2.2.4** (Post-processing [8]). If $\mathcal{K} : \mathcal{D}^n \to R$ is $(\epsilon, \delta)$-differentially private, then also $Q \circ \mathcal{K}$ is $(\epsilon, \delta)$-differentially private for $Q$ any probabilistic or deterministic function, independent of database $D \in \mathcal{D}^n$.

### 2.2.4 Noise generation mechanisms

**Randomized response**

In order to visualize perturbing an answer in a differential private way, the following example introduces a research method [15] named *Randomized response*, which allows respondents to answer sensitive questions while maintaining confidentiality. It works as follows:

1. Flip a coin
2. If tails, then respond honestly
3. If heads, flip a second coin and respond "Yes" if heads and "No" if tails.

An answer is truthfully given by a respondent with a certain probability. In this case, plausible deniability is in place: the respondent legitimately can claim that he or she did not give a certain answer. This method could also be seen as a mechanism that perturbs any specific answer to a question.

**Lemma 2.2.1** (Randomized mechanism). *Randomized response is $(\ln 3, 0)$-differentially private.*
Proof. Suppose in the worst case, that there is only one respondent, which can be either "Yes" or "No". For both outcomes, the probability that the observed answer is the same as the actual answer is the same, namely \( \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \left( \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 0 \right) = \frac{3}{4} \). The probability that the observed answer and the actual answer do not match is then \( 1 - \frac{3}{4} = \frac{1}{4} \). This leaves, using Definition 2.2.4,

\[
\exp(\epsilon) = \frac{\Pr[\text{Response} = \text{Yes} | \text{Truth} = \text{Yes}]}{\Pr[\text{Response} = \text{Yes} | \text{Truth} = \text{No}]} = \frac{\Pr[\text{Response} = \text{No} | \text{Truth} = \text{No}]}{\Pr[\text{Response} = \text{No} | \text{Truth} = \text{Yes}]} = \frac{3}{4} \cdot \frac{1}{4} = 3 \iff \epsilon = \ln 3
\]

The general form

In order to execute the ideas of differential privacy, methods are needed that provide the noise on the true answers. These methods are also called noise generating mechanisms. The general form of a certain type of mechanism \( K \) with given true answer \( q(D) \) for known query \( q \) and database \( D \in \mathcal{D}^n \) is given as follows:

**Definition 2.2.7** (General form of a noise generating mechanism). The mechanism \( K \) takes as input true answer \( t = q(D) \) and adds a random value to \( t \) by generating \( X_S \) according to some distribution and \( S \) is the set of parameters of the distribution of \( X_S \) could depend on, e.g., \( \epsilon, \delta \), the sensitivity of the query, etc. This gives the end result:

\[
K(D) = q(D) + X_S
\]

**Laplace Mechanism**

The Laplace mechanism is coined by Dwork et al. and it is proven to preserve \( \epsilon \)-differential privacy. In this case, \( X_S \) takes as input the sensitivity \( \Delta \) of the query \( q \) and \( \epsilon \). The output represents perturbed answers in \( \mathbb{R} \). The definition is given as follows:

**Definition 2.2.8** (Laplace mechanism). For a query function \( q : \mathcal{D}^n \to \mathbb{R} \) with sensitivity \( \Delta \) and privacy parameter \( \epsilon \), Laplace mechanism will output \( K(D) := q(D) + X_{\Delta,\epsilon} \) where \( X_{\Delta,\epsilon} \sim \text{Lap}(\frac{\Delta}{\epsilon}) \) and \( \text{Lap}(\lambda) \) is the Laplace distribution centered around 0. Its probability density function is given by

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

**Geometrical Mechanism**

The geometrical mechanism is a discrete variant of the Laplace mechanism, and it is proved to be preserving \( \epsilon \)-differential privacy. It is especially useful when applied to statistics that are integers, e.g., counts, ages and amounts. In this case \( X_S \) takes as input \( \epsilon \). The output represents perturbed answers in \( \mathbb{N} \). The definition for specifically a count query is given as follows:

**Definition 2.2.9** (Geometric mechanism). For a count query \( q : \mathcal{D}^n \to \mathbb{N} \) and \( \epsilon \in (0, 1) \), geometric mechanism will output \( K(D) := q(D) + X_{\epsilon} \) where \( X_{\epsilon} \sim G(\epsilon) \) and \( G(\epsilon) \) is a two-sided geometric distribution centered around 0. Its probability distribution function is given by

\[
f(x) = \frac{1 - \alpha}{1 + \alpha} \alpha^{|x|}, \forall x \in \mathbb{Z} \text{ and } 0 \leq \alpha = e^{-\epsilon} \leq 1
\]
Gaussian mechanism

The Gaussian mechanism \([9]\) satisfies \((\epsilon,\delta)\)-differential privacy according to Definition 2.2.5. In this case, \(X_S\) takes as input \(\delta,\epsilon\) and the sensitivity \(\Delta_2\) of a query \(f\) defined as

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

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

**Definition 2.2.10** (Gaussian mechanism \([9]\)). For \(\sigma = \Delta_2 \cdot \sqrt{2\ln(\frac{1+\epsilon}{\epsilon})}\), Gaussian mechanism will output \(K(D) := q(D) + X_{\Delta_2, \delta, \epsilon}\) where \(X_{\Delta_2, \delta, \epsilon} \sim \mathcal{N}(0,\sigma^2)\) and \(\mathcal{N}(0,\sigma^2)\) is the normal distribution centered around 0. Its probability distribution function is given by

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

### 2.2.5 Utility metrics

In the literature, there are several utility metrics used for analyzing the effect of a randomized mechanism \(K\) on a certain true answer \(q(D)\), perturbing it to \(q'(D)\). In this thesis, the following metrics are used:

**Definition 2.2.11** (Absolute error). Given true answer \(q(D)\) and its perturbed value \(q'(D)\), the absolute error is defined as

\[
|q(D) - q'(D)|
\]

**Definition 2.2.12** (Relative error). Given true answer \(q(D)\) and its perturbed value \(q'(D)\), the relative error is defined as

\[
\left| \frac{q(D) - q'(D)}{q(D)} \right|
\]

### 2.2.6 Challenges in differential privacy

Differential privacy provides a way to guarantee individual privacy. However, it still has practical challenges and limitations.

**Preserving privacy**

Assuming the definition of differential privacy, an individual has limited impact on the published statistics whenever one of the differentially private mechanisms is applied to them. The best way to test this is by playing the adversarial game: suppose an adversary \(A\) picks differentially private mechanism \(K\), two arbitrary neighboring databases \(D_1\) and \(D_2\) and a series of queries either submitted at once in a non-interactive setting or adaptively in an interactive setting. All are submitted to the challenger \(C\). Now, \(C\) flips a coin \(b \in \{0,1\}\) to select one of the neighboring databases and runs the protocol to generate noise on the query output. The output is returned to \(A\), and it is up to \(A\) to guess \(b\).
Preserving utility

Since most of the privacy preserving mechanisms all rely on a certain distribution, the probability exists that noise ends up destroying the utility of the query output. Especially when a relative small query answer is expected, results cannot be properly interpreted anymore and thus the research cannot produce useful results. This endangers further development of other processes. It is important to consider appropriate parameters for any choice of noise generating mechanism.

Determining trade-off privacy and utility in practice using $\epsilon$

Relative low values for $\epsilon$ result in relative high privacy, but low utility. Relative high values for $\epsilon$ 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.

Computation of $\Delta$

For simple queries, the sensitivity is not hard to determine, but for the more advanced applications, like linear regression, computing the sensitivity is more of a challenge. It requires the knowledge of the maximum difference of any two possible values in a database. If these values are not chosen to be realistic, then the mechanism could potentially destroy utility by generating excessive noise or, on the other hand, fail to provide privacy.

For example, if a data analyst queries the mean of $n$ numbers taking values in range $[a,b]$, then the sensitivity is $\frac{b-a}{n}$. Note that it is important for the range to be bounded, otherwise the sensitivity of the query is unbounded and that also leads to destructive, rather useless, noise. Also, a rough indication of $a$ and $b$ is needed: revealing its exact values would breach the privacy of the individuals belonging to the values. This estimation determines the sensitivity: after guessing $[a,b]$, all values smaller than $a$ are replaced by $a$ and all values bigger than $b$ are replaced by $b$. If $[a,b]$ is chosen too wide, the utility is potentially destroyed caused by big noise. If $[a,b]$ is chosen too narrow, the utility is potentially destroyed caused by the number of replacements.

Multiple queries in an interactive system

To answer queries in this setting, an interactive differential privacy interface is provided to enable interaction between the data analyst and the data curator. The data curator does not know upfront how many queries the data analyst has and the data analyst can decide to change tactics based on released query answers. In this setting it is hard how to spend a certain privacy budget without setting some boundaries. A couple of suggestions are given:

- Fix $\epsilon$ upfront, and once this privacy budget is exceeded, the data curator refuses to answer any more queries.
- Allow only queries from a certain collection of queries $Q$ and a certain number of rounds.

These measures restrict quite some possibilities for research, but on the other hand it can guarantee privacy for the individuals in the database.
Chapter 3

Secure Multiparty Computation

In this chapter, the motivation behind secure multiparty computation (MPC) is given. Then some existing MPC protocols are explained with their relevance. Furthermore, some definitions are highlighted and some practical examples of the application of MPC protocols are given. In the end, a detailed description is given of the MPyC framework.

3.1 Why do we need secure multiparty computation?

Various institutions deal with information that is created, transferred, stored, analyzed and processed at an increasing rate every day. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

For example, a certain treatment $X$ is provided by 6 hospitals and after a while, the hospitals want to know how well the treatment is doing in general. They are in possession of databases with varying degrees of confidentiality, and the information extracted from the databases is vital for them as it helps to improve services and to make better decisions. Comparing their data to equivalent parties may contribute to this process. Due to the sensitive nature of the data, providing the data out in the open is not an option. It could harm an organization’s reputation or give away company secrets. Secure multiparty computation is a technique for collaboratively computing on private databases without revealing any content.

3.2 Secure distributed data analysis

In this section, the mathematical background on the concept of secure multiparty computation is given along with examples and applications.
Figure 3.1: Abstract communication of the hospitals
3.2.1 Threshold cryptography

Secure multiparty computation is a field in cryptography in which multiple parties jointly compute a function by using a cryptographic protocol, keeping the input values private. Not only is it important to study the theoretical properties of a cryptographic protocol, but also the practical side: how should this be implemented? Is it vulnerable to specific attacks? Also, there is a distinction between the theory and real implementations. While in theory some protocol would absolutely work if all parties are compliant to the protocol, in practice some of the parties could be compromised (dishonest) and could possibly deviate from the protocol, endangering the privacy of the data. A solution to consider is Threshold Cryptography. This involves a secret to be distributed over multiple parties, then each party having a so-called share of that secret. The share is constructed in such a way that a certain subset of the $m$ parties, say $t \leq m$, is needed to recover that secret. This means having $t - 1$ shares or fewer implicates that it is impossible to gain any information about the secret, let alone recovering the secret. A way to describe such a protocol is by means of a secret sharing scheme. Formally, the definition is as follows [27]:

**Definition 3.2.1 (Secret Sharing Scheme).** A secret sharing scheme for a dealer $D$ and participants $P_1, ..., P_m$ comprises the following two protocols:

- **Distribution:** A protocol in which dealer $D$ shares a secret $s$ such that each participant $P_i$ obtains a share $s_i$, $1 \leq i \leq m$.
- **Reconstruction:** A protocol in which secret $s$ is recovered by pooling shares $s_i$, $i \in Q$, of any qualified set of participants $Q \subseteq \{P_1, ..., P_m\}$.

The set of all qualified sets is called the access structure. In threshold cryptography, only the $(t, m)$-threshold secret sharing schemes are relevant. These are secret sharing schemes where the access structure consists of sets with size in $[t, m]$, with $1 \leq t \leq m$.

3.2.2 Secret sharing and resharing

An example of a $(t, m)$-threshold secret sharing schemes is Shamir’s secret sharing scheme [28]. The idea behind this scheme relies on the fact that it takes $t$ points to construct a unique polynomial of degree $t - 1$. So the secret could be represented by $m$ shares and there are only $t$ shares needed to construct $s$ again. Let $s \in \mathbb{Z}_p$ be a number representing the secret with $p$ prime.

For the distribution, the dealer picks a random polynomial $f(X) \in \mathbb{Z}_p[X]$, i.e., a polynomial with coefficients in $\mathbb{Z}_p$, and $\deg(f) < t$ with $f(0) = s$. It sends share $s_i = f(i)$ to participant $P_i$, $i \in [1, m]$.

For the reconstruction, any set of the access structure is able to recover $s$ using their shares and Lagrange interpolation, resulting in

$$s = \sum_{i=1}^{t} s_i \lambda_i \quad \text{with} \quad \lambda_i = \prod_{j=1, j\neq i}^{t} \frac{j}{j-i}$$

In practice, it is often necessary to compute new shares with some fresh randomness. It may happen that over time, some shares are compromised, and then new shares have to be produced and the old ones have to be discarded. This process is called resharing.
3.2.3 Protocols

Addition

Using the above, one can define a general addition protocol [2]. Suppose two dealers $D_1$ and $D_2$ have respectively secrets $s_1$ and $s_2$ and they want to compute $s_1 + s_2$ securely.

For the distribution, $D_i$ uses the above scheme to share $s_i$ among $n$ parties, $i = 1, 2$. Then each party $p_j$ holds two shares: $f_j$ from $D_1$ and $g_j$ from $D_2$, $j = 1, ..., m$. Now, $p_j$ locally computes $s(j) = f_j + g_j = f(j) + g(j) = (f + g)(j)$.

For the reconstruction, any subset of $t$ parties now can jointly compute $s_1 + s_2$ by using Lagrange interpolation using the $s(j)$ available resulting in $h(x) = (f + g)(x)$ and then evaluating $h(0) = s_1 + s_2$.

Multiplication

Computing $s_1 \cdot s_2$ is somewhat more complex, and for this the protocol of Gennaro, Rabin and Rabin is used [18]. Suppose two dealers $D_1$ and $D_2$ have respectively secrets $s_1$ and $s_2$ and they want to compute $s_1 \cdot s_2$ securely.

For the distribution, $D_i$ uses the above scheme to share $s_i$ among $m$ parties, $i = 1, 2$. Then each party $p_j$ holds two shares: $f_j$ from $D_1$ and $g_j$ from $D_2$, $j = 1, ..., m$. Now, $p_j$ locally computes $s(j) = f_j \cdot g_j$. $P_i$ shares this value by choosing random polynomial $h_i(x)$ with $\text{deg}(h_i) < t$, such that $h_i(0) = s(j)$ and sending $p_k$ the value $h_i(k)$. Locally, the parties have to compute $h(k) = \sum_{i=1}^{2t+1} \lambda_i \cdot h_i(k)$.

For the reconstruction, any subset of $t$ parties now can jointly compute $s_1 \cdot s_2$ by using Lagrange interpolation using the $h(k)$ available resulting in $h(x)$ and then evaluating $h(0) = s_1 \cdot s_2$.

3.2.4 Security requirements

Formally, let parties $P_1, ..., P_m$ be in possession of the values $x_1, ..., x_m$ respectively. They want to evaluate some function $f(x_1, ..., x_m)$. Then the formal definition of the problem of secure multiparty computation is to find a protocol for $P_1, ..., P_m$ to jointly compute $y = f(x_1, ..., x_m)$ while keeping $x_1, ..., x_m$ a secret. This means the following two conditions have to be satisfied [27]:

1. **Correctness**: Every party receives the correct output of the model.
2. **Privacy**: $y$ is the only new information that is released.

3.2.5 Involved parties

Secure multiparty computation involves a couple of important stakeholders:

- **Input Party**: providing the data.
- **Result Party**: asking for result of given query.
- **Computing Party**: part of the set of computing parties that jointly compute the necessary functions.
For MPC protocols, the idea of "no single point of trust" is central. This means that no individual link of the system should be able to get full access to the data. This is because there could be dishonest parties present: as opposed to honest parties, dishonest parties may be corrupted and might deviate from the protocol or leak information.

### 3.2.6 Examples

A couple of examples in which MPC protocols is used:

- **Yao's millionaires problem** [35]: Alice and Bob want to know who is richer. In this case, let $x_1$ be the amount of money Alice possesses, and $x_2$ be the amount of money Bob possesses. Then the function to be evaluated is $f(x_1, x_2) = x_1 > x_2$, which results in 'True' when Alice is richer than Bob, and otherwise in 'False'.

- **Private dating**: Alice and Bob meet up in a bar and after their date, they want to know if they both want to continue dating in the future while the other one's intention is secret in case there is no interest. Both choose a number: either 1 if there is interest, else 0. Let $x_1$ be that number for Alice, and $x_2$ the number for Bob. Then the function to be evaluated is $f(x_1, x_2) = x_1 \land x_2$.

- **Auction**: Many parties wants to execute a private auction in which the highest bid wins, but only that bid is revealed. Let $x_1, ..., x_m$ be the bids of respectively party $p_1, ..., p_m$. Then the function to be evaluated is $f(x_1, ..., x_m) = \max(x_1, ..., x_m)$.

- **Voting**: A population of $n$ people is going to vote on election day and the choice is either A or B. In order to know who won at the end of the day, all votes must be counted. Let $x_i$ be a vote of person $i$, which is 1 if $i$ voted for A and is 0 if $i$ voted for B. Then one has to evaluate $f(x_1, ..., x_m) = x_1 + ... + x_m$.

#### Secure sum

An example of a solution to the secure sum problem is given as follows:

Let parties $\mathcal{P}_i, 1 \leq i \leq m$ be participants and they want to know the sum of their values $x_1, ..., x_m$ with $x_i \in \mathbb{Z}_p$ and $p$ prime (collectively agreed on). Then each $\mathcal{P}_i$ randomly produces $r_{i,1}, ..., r_{i,m-1} \in \mathbb{Z}_p$ and sets $r_{i,m} = x_i - \sum_{j=1}^{m-1} r_{i,j} \mod p$.

The distribution phase comprises two steps:

1. First, $\mathcal{P}_i$ sends $\{r_{i,k} | k \neq j\}$ to $\mathcal{P}_j$ for $1 \leq j \leq m$, including himself, in a private way.

2. Second, $\mathcal{P}_i$ adds the shares of all secrets by computing $s_n = \sum_{k=1}^m r_{k,n} \mod p$ for all $n \neq i$ and announces all $s_n$ to the others.

For the reconstruction phase, the parties compute $v = \sum_{k=1}^m s_k \mod p$.

This protocol satisfies the security requirements:

$$v \equiv \sum_j s_j \equiv \sum_j \sum_i r_{i,j} \equiv \sum_i \sum_j r_{i,j} \equiv \sum_i x_i \mod p$$

and no information other than the sum is revealed.
3.3 MPyC

All programs in this thesis are written in Python3 and specifically in the MPyC environment, developed by Berry Schoenmakers [19]. MPyC is a Python framework for multiparty computation, building on some of the fundamental ideas underlying VIFF [34]: Virtual Ideal Functionality Framework. The secure computation is done by means of special MPyC coroutines. Based on Python’s module asyncio, the coroutines provide faster computation. All operators (+, -, *, /, <=, <, >, >=, ==, !=) are overloaded such that these operations on values are done in private and no information will leak from performing these operations. Furthermore, MPyC is able to compute with $m$ parties, supporting Shamir secret sharing, and it provides security against passive adversary. Also fields of order prime $p$ are implemented and all of the above operations support modular arithmetics. The transparency of the communication between parties is mainly due to the use of sophisticated operator overloading combined with asynchronous evaluation of the associated protocols.

3.3.1 Datatypes

In MPyC, three datatypes are used to type the secret shared data:

- $\text{secint} := \text{mpc.SecInt}(l=\text{None}, p=\text{None}, n=2)$: Secure $l$-bit integer values
- $\text{secfxp} := \text{mpc.SecFxp}(l=\text{None}, f=\text{None}, p=\text{None}, n=2)$: Secure $l$-bit fixed-point numbers with $f$-bit fractional part
- $\text{secfld} := \text{mpc.SecFld(order=\text{None}, modulus=\text{None}, char2=\text{None}, l=\text{None})}$: Secure prime or binary field of $(l + 1)$-bit order

3.3.2 Computations in MPyC

All computations are done asynchronously in MPyC. These computations are implemented as MPyC coroutines, which are a special type of Python coroutines. A Python coroutine, defined by the keyword async at the start of a function definition, is turned into an MPyC coroutine by using the decorator mpc.coroutine. When called, an MPyC coroutine will return immediately, i.e., it does not block execution. The main difference with Python coroutines is that an MPyC coroutine will return a placeholder or, more generally, nested lists/tuples containing placeholders. The placeholders are typed (e.g., of the types mentioned above), and the type of the placeholders is defined by the first await expression in the function definition, using the mpc.returnType method. This way it is clear what work is ahead of the computation. In the case of local computations, these can already be done. The main purpose is therefore to have the CPU(s) do the useful work while waiting for the data of others.

The system assumes passive security with a majority of honest parties. This means that with $m$ parties, there are up to $t$ (passive) dishonest parties with $m \geq 1$ and $0 \leq t \leq (m - 1)/2$. The underlying protocols are based on threshold secret sharing over finite fields (using Shamir’s threshold scheme as well as pseudorandom secret sharing).

The aim is for algorithms to be oblivious: all steps within the algorithm are executed independent of the input so nothing can be told about the actual input values. This process is often described as a circuit: the circuit takes some input, goes through a fixed number of stages and takes as much return time as it would have needed with equivalent input.
Regarding the implementation of such an algorithm, it is in general not possible to manage conditional expressions in a private way. It is important to hide which branch of an if-else statement is followed. Both cases are compared and combined in a so-called *oblivious choice*: suppose \( p \) is secret Boolean representing either True or False, then expression

\[
\text{if } p \text{ then } a \text{ else } b
\]

is replaced by

\[
p \cdot a + (1 - p) \cdot b
\]

and to reduce the number of multiplications, it is often implemented as

\[
b + p \cdot (a - b)
\]
Chapter 4

Achieving differential privacy using secure multiparty computation

In this chapter, the reasoning for the combination of differential privacy and secure multiparty computation is given. Then the formal definition is given and the challenges are explained. In the end, some existing frameworks are described and the main contributions of this thesis are stated.

4.1 How can we join forces between differential privacy and secure multiparty computation?

Two great flavors that go great together: secure multiparty computation solves the problem of computing on private databases without learning their content and differential privacy solves the problem of disclosing information about the individuals that are contained in databases. A union of secure multiparty 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 visualisation of the setting for three hospitals H1, H2 and H3, each containing datasets of patients, is displayed in Figure 4.1.

4.2 Private distributed data analysis

In [33], an abstract definition is given to combine the two concepts. Assume a database, consisting of \( n \) rows is shared among \( m \) parties \( P_1, ..., P_m, \frac{n}{m} \in \mathbb{N} \). 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 \( P_k \) has exactly \( \frac{n}{m} \) rows of the dataset, denoted by \( D_k = (D_{k,1}, D_{k,2}, ..., D_{k,n/m}) \). The goals is for \( P_k \) 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 \( Q = (P_1, ..., P_m) \) on input \( x \) is denoted by \( \text{View}_A(A \leftrightarrow Q(x)) \). Assume the worst case and that \( A \) passively controls \( P_{-k} = (P_1, ..., P_{k-1}, P_{k+1}, ..., P_m) \), \( \text{View}_A(A \leftrightarrow Q(x)) \) is determined by all inputs and randomness of all parties other than \( P_k \) and the messages.
Combining Differential Privacy with MPC

Figure 4.1: Combining DP and MPC
sent by $\mathcal{P}_k$.

**Definition 4.2.1 (Multiparty Differential Privacy).** Let $\mathcal{P}_1, \ldots, \mathcal{P}_m \in (\mathcal{D}^{n/m})^m$ be parties in a protocol $Q = (\mathcal{P}_1, \ldots, \mathcal{P}_m)$ with input datasets $(D_1, \ldots, D_m)$, $Q$ is $(\epsilon, \delta)$-differentially private (in an honest-but-curious model), if for every $k \in \{1, \ldots, m\}$ and for every two neighboring dataset $D, D' \in (\mathcal{D}^{n/m})^m$, the following holds for every feasible set $T$:

$$P[\text{View}_{\mathcal{P}_k}(\mathcal{P}_k \leftrightarrow Q(D))] \in T] \leq e^\epsilon \cdot P[\text{View}_{\mathcal{P}_k}(\mathcal{P}_k \leftrightarrow Q(D')) \in T] + \delta$$

Take the hospitals in Figure 4.1. In this example, a database containing records of patients on treatment $X$ is distributed over three hospitals (name this set $H = \{H_1, H_2, H_3\}$). Suppose a malicious entity $E$ that has control over two of them, being $H' = \{H_1, H_2\}$. All hospitals engage in a $(\epsilon, \delta)$-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 $D \in (\mathcal{D}^{n/m})^m$. The goal of $E$ is to recover (a piece of) the database of $H_3$ by learning from $V$. The goal of $H_3$ is then for two neighboring databases $D, D' \in (\mathcal{D}^{n/m})^m$ 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.

Noise could be applied locally. The example mentioned in Section 2.2.4 could be extended to such a protocol with query family $q : \mathcal{D}^n \rightarrow [0, 1]$. First, compute $q(D_k)$ and a $b_k \in \{0, 1\}$ with $P[b_k = 1] = q(x_k)$. Based on a coin toss, either $q(D_k)$ is given or $b_k$.

In this thesis, the noise is added globally: the query is the function that is jointly evaluated in an MPC setting, the noise is generated using secret shared input and oblivious protocols and then a secret addition is performed of the true answer of the query and the noise value. This is the most straight-forward way to address this problem of combining DP and MPC and furthermore, global noise is a well-studied area in differential privacy.

### 4.3 Security requirements and challenges

This hybrid idea of using both secure multiparty computation and differential privacy in one program takes care of two problems: there is no reason for data to be stored at one specific site and it is possible to publish statistics in a privacy-preserving manner. This results in the following extension of the security guarantees from Section 3.2.4:

- **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 secure multiparty computation 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*. 

Chapter 4 Kimberley Thissen 23
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 make the hybrid model scalable to more parties, provide the ability of using other mechanisms and make it overall efficient.

4.4 Existing frameworks and applications

In this section, a couple of developed programs for combining MPC and DP are discussed: PrivaDA [10] and Jana [1] combine MPC and DP in their framework, Sharemind [1] is an MPC platform, which is often used for DP experiments. Furthermore, SCALE-MAMBA [26] served as one of the examples for the building blocks in this thesis.

4.4.1 PrivaDA

PrivaDA presents a generic architecture for computing differentially private statistics from distributed data. Central is the problem of releasing aggregated data coming from multiple parties. The aggregation part is handled by MPC and afterwards DP is used to perturb the data such that the data is safe to published.

The construction is organized in two stages: first, the aggregation phase and second, the perturbation phase. Their set-up is similar to the approach in this thesis. In the first phase, the parties securely compute the result and in the second phase, they jointly generate the input for the samplers so that noise is produced using secure protocols.

The program tackles the above challenges. It shows generality by providing different choices of noise-generating mechanisms. Whereas this thesis contains implementations of the Laplace mechanism, geometric mechanism and Gaussian mechanism, PrivaDA chose to implement the Laplace mechanism, geometric mechanism and exponential mechanism. Furthermore, it achieves DP in multiple scenarios, even if there are parties colluding. It claims to be resistant against pollution attacks, by adding minimal noise and only accepting client input that are contained in a set of valid answers.

The protocols mentioned in [10] achieve $\epsilon$-multiparty differential privacy according to definition 4.2.1 assuming an honest-but-curios model and less than half of the parties colluding.

4.4.2 Jana

The Jana system, developed by various entities, provides an MPC-secured database and a tool to get statistical information based on the database. It combines MPC, differential privacy (see Chapter 3) and encryption of data in all stages of the data. Jana uses the so-called Data as a Service model (DaaS), where result parties can access an actively updated shared data resource for a relatively long time.
Jana studies the trade-off between security and performance. In particular, Jana assumes end-to-end security: every state of data is encrypted and also assumed is the encryption cannot be broken, so the total focus of the study is optimizing the results of performance tests. In the end, only the security levels of the used cryptosystems have to be considered. Linear secret sharing is used to make sure query results are never out in the open within the system.

Input and output parties communicate through the SQL declarative query language. A series of SQL features, operators and functions are supported. After the queries of the result parties have been answered, another layer of protection is added based on differential privacy. This means that next to the usual SQL parameters, some parameter is added causing the noise on the results. This noise is also in secret shared form, generated by a distributed random generator. Combined the shares of the noise are added to the shares of the query result, and after that, the actual query result is computed.

Jana is currently deployed in several use cases, one them being the DARPA Brandeis program [31]. It is an ongoing research project. Like in this thesis, Jana tries to find an efficient hybrid of Secure Multiparty Computation and Differential Privacy. One of their currents aims is to further develop the noise generation. The differential privacy mechanism could potentially be leaking information. Asking for the same query may result in “cancelling out” the noise and thus revealing information about the data. A solution would be to add deterministic noise: all query outputs comes with their own unique noise value. Unfortunately, one could reverse engineer the distribution. This will also be discussed in this thesis.

4.4.3 Sharemind

Sharemind, developed by Cybernetica (Estonia), also provides a way for securely shared databases, but in this situation the database is distributed among various entities (whereas Jana works with a centralized database). This is because of the data-sharing problem nowadays: information travels from one place to another all the time. This flow gets restricted by governments that enforce privacy protection laws, distrust between certain parties or by not choosing for certain data share services. Sharemind provides tools for anonymous processing: data remains private during the entire process time, and the result parties are authorized before delivering the results.

The structure of Sharemind is pretty straight forward: the input parties provide the data, encrypt them according to a certain protocol and hand them over to the computing parties. The result parties send queries to the computing parties. The computing parties evaluate the query according to another certain protocol based on the collected input from the input parties without leaking the input data in the meanwhile. Again, the results get encrypted and gets sent to the result parties. They will decrypt and learn the result. Authorisations are done by access control and together with the secure channels, they are centralized by Sharemind. The programming is done in SecreC: a high-level language for writing privacy-preserving algorithms without any knowledge of the underlying cryptographic protocol.

An example of an application of Sharemind is the following case: universities wanted to know why Computer and Software Science students have a low graduation rate and if that had something to do with the jobs they do on the side. They had to consult two different resources: tax records and the educational records. The data import was secured using the tools of Sharemind and the queries executed by an independent research center. Another application is the implementation of functions that could predict collisions with satellites.
4.4.4 SCALE-MAMBA

SCALE-MAMBA [26] is an MPC software system developed at KU Leuven. It is based on a
couple of existing schemes, but aims to be a complete system, and not a set of components
which the user needs to compose together. The design of the protocols for secure evaluation
of the complex functions mentioned in Chapter 5 is partially based on theirs, hence, it was a
useful reference point for this thesis.

4.5 Approach

The goal of this project is to design and implement methods that generate randomness in a
secure multiparty computation setting in order to satisfy differential privacy properties and
to improve privacy preserving services for processing, e.g., health care records. The building
blocks necessary to reach this goal are given in Chapter 5. The corresponding protocols evalu-
ate specific functions, implemented in the MPyC framework which may include further specific
optimizations using the capabilities of the particular framework. The design of them are op-
timized for the MPC setting. In Chapter 6, noise generating mechanisms are implemented
based on mathematical constructions for random variable sampling in combination with the
differential privacy guarantees. These mechanisms are optimized for and tested within the
MPyC environment. In Chapter 7, use cases are given and their purpose is threefold:

- Provide a clear and hands-on example on how and why differential privacy works in
  practice.

- Elaborate on the effect of the randomness in terms of global noise and privacy budgets.

- Give an extensive example on how DP and MPC could be combined for useful practices.

Chapter 8 concludes this thesis with an overview of the achieved results and some suggestions
for further research.
Chapter 5

Building blocks

In order 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. The benefit of operations on fixed-point numbers is that they are in general quite fast on machines. Furthermore, the choice for fixed-point numbers offers sufficient and acceptable precision for the application of differential privacy.

Basic building blocks have been defined in the libraries of MPyC. Here, the more complex operations, such as trigonometric operations, are treated in detail. The focus of the design of the protocols is on achieving efficiency in combination with the desired precision and provable security guarantees.

This chapter starts with some preliminaries on the fixed-point number type and the origin of the approximations used in the implementations of the protocols. Then the experimental setting is described along with an explanation of the complexity analysis. After that, the protocols are given with some experimental results and complexity. The building blocks are used in Chapter 6 for the noise generating mechanisms.

5.1 Fixed-point numbers

Fixed-point numbers are a type of numbers consisting of an integer part and a fractional part. Usually, real numbers are represented by floating points. The difference between the two types is the position of the radix point. The decimal point of floating numbers can be moved, such that the exponent changes: one can make 2.55 arbitrarily large (2.55E+07) or arbitrarily small (2.55E-15), whereas the length of the fixed-point number (the length of the integer part plus the length of the fractional part) is fixed. Fixed-point numbers are used for systems where performance is more important than precision, because in general fixed-point arithmetic is faster than floating-point arithmetic.

An example

Recall that any number can be expressed as a sum of powers of 10, for example: $55 = 5 \cdot 10^0 + 5 \cdot 10^1 + 0 \cdot 10^2 + ...$. Therefore $55_{10}$ is a decimal representation. This can also be done in terms of powers of two: $55 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 0 \cdot 2^7 + ...$. Hence, $110111_2$ is a binary representation of the same number. When this number is divided by 2, the decimal representation is 27.5. The decimal point is used to denote a fractional part.
A binary point works the same: the binary representation of $27.5 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1}$ changes into 11011.12. Note that the binary point has shifted to the left after dividing by 2. This makes the shifting process equivalent to dividing by 2 (or multiplying with 2, depending on the direction of the shift). Formally, let a fixed-point number $x$ be represented as $s \cdot (d_{e-2}, ..., d_0, d_{-1}, ..., d_{-f})$ with value $x = s \cdot \sum_{i=\text{e}_f}^{\text{e}_2} d_i 2^i$ and $s \in \{-1, 1\}$ being the sign, $f$ being the length of the fractional part and $e$ being the length of the integer part (including the sign bit).

5.2 Computer approximations

The protocols in this chapter rely mainly on the approximation polynomials that are used to approximate the output of a function for specified input. The approximation polynomials and their coefficients are listed in [14].

Function subroutines are proposed in [14] to provide values of a function $f \in C$, for some function space $C$, in a specified domain at a specific precision. Three types of precision that can be considered while designing a function subroutine are the following:

1. Type I: Maximum precision depends on the word-type storage a machine has and then adapts the precision of the function subroutine to that. The design of such a function subroutine demands knowledge of the computer and $f$ itself. These routines take up a significant percentage of the running time.

2. Type II: Variable precision is where the precision is a variable and the routine is asked to evaluate $f$ for some input $x$ correct to $s$ significant figures. Examples are expansions such as Taylor and Chebychev series.

3. Type III: Relaxed precision states that subroutines only compute as much precision as needed for a certain task, or when it is extremely wasteful to compute unneeded digits.

To quantify the precision of an approximation, the function subroutines encounter gross errors: the difference between the true function value $f(x)$ and the generated numerical value $g(x)$ (the approximation of $f(x)$ with $x$ in $[a,b]$, $a,b \in \mathbb{R}$). This is a consequence of truncation errors and round-off errors caused by the individual arithmetical operations. The absolute error is the absolute value of the gross error, i.e.

$$E_A(x) = |f(x) - g(x)|$$

The relative error is the absolute value of the gross error divided by the true function value, i.e.

$$E_R(x) = \left| \frac{f(x) - g(x)}{f(x)} \right|$$

The approximations are given for a certain interval. This is due to the fact that the approximation converges to the true value of the corresponding function on that interval. In general, infinite expansions can be divided in two types of behavior with respect to convergence. The first type consists of those such as Taylor series, which evaluate at a single point and the convergence decreases rapidly as the distance from that point increases. The second type consists of those such as Fourier series: the coefficients depend upon the behavior of the function in a certain domain, rather than a single point. This type of series converge rapidly enough for computations and most of the error is contributed by the first omitted term.
A special type of optimum approximations is proposed: the so-called least maximum approximations. Suppose a certain real-valued and continuous function \( f \) on \([a, b]\) such that for every \( x \in [a, b] \) the value \( f(x) \) is available as an output of an existing computer program with some precision. Since this program is probably too expensive to run, \( f \) is approximated by another program using a polynomial \( P \in \mathcal{P}_n \) with a finite number of multiplications and additions and \( \deg(P) < n \). Then the distance of \( P \) and \( f \) is defined as the worst of the absolute errors, i.e.,

\[
\|f - P\| = \max_{a \leq x \leq b} |f(x) - P(x)|
\]

Here, \( P(x) \) is also referred to as minimax approximations and \([a, b]\) is the interval in which \( P(x) \) is defined. An important theorem that is pertinent at this point states that for each continuous function \( f \) defined on \([a, b]\) and for each \( n \), \( \mathcal{P}_n \) contains a unique element \( P \) of minimum distance from \( f \). This is a consequence of the following theorem, stated in [14]:

**Theorem 5.2.1 (Weierstrass approximation theorem).** Let \( f \) be a continuous function defined on \([a, b]\). Given \( \epsilon > 0 \), there exists a \( P \in \mathcal{P}_n \) such that \( \|P - f\| \leq \epsilon \).

The downside of using \( P \) is that convergence is relatively slow and a tolerable distance is achieved with relatively large \( n \). This could be solved by approximating with rational functions of the form \( \frac{P}{Q} \).

The minimax approximations are computed by means of variants of an algorithm credited to Remes, given in Section 3.2 of [14]. An approach is to reduce the domain of the approximation function. In cases like \( \sin(x) \), this is possible due to symmetry and periodicity. In other cases, arithmetic rules are applied like \( \exp(x + y) = \exp(x) \cdot \exp(y) \).

Secure protocols are designed to run independent of the input data and therefore all feasible cases have to be considered in the protocol. This results in taking relative long time to run the underlying algorithm. Nevertheless, it is important to consider every possibility for reducing the computation time. One of these approaches is called the Horner’s rule and for polynomials \( P_n(x) = \sum_{i=0}^{n} p_i x^i \) it is given as follows:

\[
\begin{cases}
V_n = p_n \\
V_k = x V_{k+1} + p_k, k = n - 1, n - 2, \ldots, 0 \\
P_n(x) = V_0
\end{cases}
\]

This form is frequently used as it only needs \( n \) multiplications and \( n \) additions.

The accuracy of the approximations is determined by means of a criterion of standardization. This criterion is based on the error curve. Let \( e(x) = f(x) - P_n(x) \) be the error curve. The coefficients of the approximations were truncated so that the error curve would be accurate to three significant figures at the critical points. Suppose the rounded error function \( \tilde{e}(x) = f(x) - \tilde{P}_n(x) \) and define \( \lambda := \max |e(x)| \). Then the condition for rounding is

\[
|\tilde{e}(x) - e(x)| \leq 10^{-3} \lambda
\]

For the precision of the approximations, which is represented by the relative accuracy, it holds that

\[
|\tilde{p}_i - p_i| \leq \frac{10^{-3} \lambda}{n + 1} \cdot \min_x \left| \frac{f(x)}{x^i} \right|
\]
5.3 Protocol analysis

The protocols stated in the following sections are analyzed in a similar fashion and in the same setup as described below. All experiments are done on a normal PC in a Jupyter Notebook (Anaconda) and in 3-party mode.

Precision and speed

As described in the previous section, the machine precision influences the exact precision of the approximation. 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 $10^{-5}$. In the following analysis, the default setting is used and the aim is to have a precision equivalent to the limits of the 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 5.1.

Complexity

Important aspects with respect to the complexity of an algorithm in MPyC are the number of random bits generated and the number of secure multiplications. If computed without the use of MPC, these operations would not take a significant amount of time to execute. However, 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.

The generation of random bits is used for, e.g., secure comparisons. The security of this protocol heavily relies on the generation of randomness. But since randomness is relatively costly to produce locally, one should aim to minimize this process.

Multiplying securely is done via the GRR [18] protocol, described in Section 3.2.3. To prevent leakage of sensitive information, a protocol like the multiplication protocol makes use of the randomization of (intermediate) values thereby making it harder to get information from the secret values. In comparison with this costly protocol, the performance time of the addition protocol is negligible. Therefore, additions are not considered when determining the complexity.

All information about the complexity of the building blocks i to be found in Table 5.1. In this table, $\theta = \lceil \log_2 \left( \frac{2f+1}{3.5} \right) \rceil$ 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.

5.4 two

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 $f$ the length of the fractional part. So 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 $2^{2i}$. The end result is $\prod_{b_i=1} 2^{2i} = 2^n$. A similar reasoning is applied to the case that $n$ is negative.

**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: | $2^n, n \in \mathbb{Z}$ |

$n \leftarrow n >> f$  \hspace{1cm} #shift $f$ bits to right  
$bp \leftarrow 1$  
$bn \leftarrow 1$  
$b \leftarrow \text{to bits}(n, l-f)$  \hspace{1cm} #bit decomposition of $n$ to the $l-f$-th bit  
$s \leftarrow b[-1]$  \hspace{1cm} #signed bit  

**for** $i$ **in** range(0, $f$) **do**  
\hspace{1cm} $t \leftarrow b[i]$  
\hspace{1cm} $bp \leftarrow bp + t \cdot (-bp + bp \cdot 2^{2i})$  
\hspace{1cm} $bn \leftarrow bn + (1-t) \cdot (-bn + bn \cdot (0.5)^{2i})$  

**end for**  
**return** $s \cdot (0.5 \cdot bn) + (1-s) \cdot bp$

Since the multiplications are done with so-called ‘machine numbers’ and no approximation functions are used, the results do not suffer from precision loss.

### 5.5 exp($x$)

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)`, see C.2 since $x'$ is always positive. The evaluation of $2^{x''}$ is approximated by $p_{1045}$ defined on domain $[0, 1]$ and whose coefficients are given in Table B.2. 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: $e^x$

$x \leftarrow 1.442695040889 \cdot x$
$s \leftarrow \text{sgn}(x)$
$x \leftarrow s \cdot x$
$x' \leftarrow \text{trunc}(x)$
$x'' \leftarrow x - x'$
$f \leftarrow 2^{x'}$
$g \leftarrow p_{1045}(x'')$
$h \leftarrow f \cdot g$
$h^{-1} \leftarrow \frac{1}{h}$
$k \leftarrow s \cdot s \cdot \left( (-\frac{s-1}{2}) \cdot h^{-1} + (\frac{s+1}{2}) \cdot h \right) - 1) + 1$

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 this thesis, loss of precision will probably not lead to failure of generating appropriate noise.

5.6 $\ln(x)$

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:

$$\ln(x) = \ln(\frac{x}{2^k} \cdot 2^k)$$

$$= \ln(e^{\ln(\frac{x}{2^k})} \cdot e^{k \cdot \ln(2)})$$

$$= \ln(\frac{x}{2^k}) + k \cdot \ln(2)$$

Here, $\frac{1}{2^k}$ and $k$ are retrieved by calling the method $\text{anorm()}$, see C.3. $\ln(\frac{x}{2^k})$ is approximated with $p_{2607}$, see B.1 and $\ln(2)$ is given by fixed number 0.6931471805599453. The corresponding protocol is given with input secret shared value $x$ and output the evaluation of $\ln(x)$:
Algorithm 3 Protocol for $\ln(x)$

**Input:** fixed-point number $x$

**Output:** $\ln(x)$

\[
v \leftarrow \text{anorm}(x) \quad \text{ (#modified version of mpc\_norm(), see C.3)}
g \leftarrow x \cdot v[0] \\
f \leftarrow p_{2607}(g) \\
k \leftarrow v[1] \quad \text{ (#v[1] is $k$ from scaling factor)}
g \leftarrow f + k \cdot 0.69314718055994530
\]

return $g$

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

## 5.7 $\sqrt{x}$

The computation of the square root of a given number $x$ is done in a similar fashion as the logarithm:

\[
\sqrt{x} = \sqrt{\frac{x}{2^k} \cdot 2^k} \\
= \sqrt{\frac{x}{2^k} \cdot \sqrt{2^k}} \\
= \sqrt{\frac{x}{2^k} \cdot 2^{\frac{k}{2}}}
\]

Here, $k$ is chosen such that $\frac{x}{2^k} \in \left[\frac{1}{2}, 1\right]$. For evaluating $2^{\frac{k}{2}}$, information about the parity of $k$ is needed. 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 $p_{0132}$, see [B.3] which has a relative error of $10^{-5.08}$ and that corresponds with a precision up to 16 fractional bits.

Algorithm 4 Protocol for $\sqrt{x}$

**Input:** fixed-point number $x$ and fractional length $f$

**Output:** $\sqrt{x}$

\[
t \leftarrow \text{is\_zero}(x) \quad \text{ (#checks whether $t$ is zero)}
v \leftarrow \text{anorm}(x) \quad \text{ (#modified version of mpc\_norm(), see C.3)}
[k] \leftarrow v[1] \quad \text{ (#v[1] is $k$ from scaling factor)}
p \leftarrow k \% 2
p \leftarrow p >> f
a \leftarrow p_{0132}(x \cdot v[0]) \quad \text{ (#v[0] is the scaling factor $1/2^k$)}
s \leftarrow (1 - p) \cdot (a \cdot \text{two}(\frac{k}{2})) + p \cdot (a \cdot \text{two}(\frac{k - 1}{2}) \cdot 1.41421356237309504880168872420)
\]

return $(1 - t) \cdot s$
The behavior of this method is as expected and satisfies the pre-determined goals. Notable is that there is no accuracy loss after computing with the approximation.

5.8 \( \cos(x) \) and \( \sin(x) \)

A secure way of the computation of the trigonometric functions \( \cos(x) \) and \( \sin(x) \) is based on the ideas of [4] and SCALE MAMBA [26]. 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 \( x' \in [0, \frac{1}{2}\pi] \), then \( b_2 = 1 \), else, \( b_2 = 0 \). This is done as described in Section 3.3.2. When this interval is identified, \( x' \) can be safely adjusted to interval \([0, \frac{1}{2}\pi]\) using the appropriate trigonometric rules.

3. \( \cos(x) \) and \( \sin(x) \), \( x \in [0, \frac{1}{2}\pi] \), are approximated by \( p_{3508} \) and \( p_{3347} \) respectively. The coefficients are stated in B.5 and B.4, respectively. With the information gathered in step 2 and the approximations, \( \sin(x) \) and \( \cos(x) \) are evaluated.

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

\begin{algorithm}
\caption{\texttt{trigsub(x)}}
\begin{algorithmic}
\State \textbf{Input:} fixed-point number \( x \)
\State \textbf{Output:} \( z \in [0, \frac{1}{2}\pi] \), \( b_1 \in \{0, 1\} \), \( b_2 \in \{0, 1\} \)
\State \( \pi \leftarrow \text{secfexp}(3.1415926535897932384) \)
\State \( p \leftarrow \frac{x}{\pi} \)
\State \( q \leftarrow \text{trunc}(p) \)
\State \( r \leftarrow x - 2\pi q \)
\State \( b_1 \leftarrow \text{ge}(r, \pi) \) \quad \#if \( r \geq \pi \) then \( b_1 = 1 \) else \( b_1 = 0 \)
\State \( f \leftarrow 2\pi - r \)
\State \( w \leftarrow r + b_1(f - r) \)
\State \( b_2 \leftarrow \text{ge}(w, \frac{1}{2}\pi) \) \quad \#if \( w \geq \frac{1}{2}\pi \) then \( b_2 = 1 \) else \( b_2 = 0 \)
\State \( v \leftarrow \pi - w \)
\State \( z \leftarrow w + b_2(v - w) \)
\State \text{\bf return} \( z, b_1, b_2 \)
\end{algorithmic}
\end{algorithm}

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 \( p_{3302} \). It has a relative error of \( 10^{-24.33} \) if computed exactly, which corresponds with a precision up to 80 fractional bits.
Combining Differential Privacy with MPC

Table 5.1: Complexity table: $\theta = \lceil \log_2 \left( \frac{2f+1}{3.5} \right) \rceil$ 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Multiplications</th>
<th>Scalar Multiplications</th>
<th>Randomness</th>
<th>Speed(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>two</td>
<td>$7f$</td>
<td>$l - \log_2 l - 1$</td>
<td>$f$</td>
<td>0.08</td>
</tr>
<tr>
<td>exp</td>
<td>$7l + 5f + 20 + 19$</td>
<td>$5l - 3 \log_2 l - 5$</td>
<td>$4l + (15 + 2\theta)f$</td>
<td>0.19</td>
</tr>
<tr>
<td>ln</td>
<td>$5l + 8$</td>
<td>$2l - \log_2 l - 2$</td>
<td>$l + 9f$</td>
<td>0.14</td>
</tr>
<tr>
<td>sqrt</td>
<td>$8l + 14f + 14$</td>
<td>$8l - 4 \log_2 l - 8$</td>
<td>$5l + 13f$</td>
<td>0.48</td>
</tr>
<tr>
<td>sin</td>
<td>$3f + 17$</td>
<td>$l - \log_2 l - 1$</td>
<td>$5l + 15f$</td>
<td>0.07</td>
</tr>
<tr>
<td>cos</td>
<td>$3f + 15$</td>
<td>$l - \log_2 l - 1$</td>
<td>$5l + 13f$</td>
<td>0.07</td>
</tr>
</tbody>
</table>

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

**Input:** fixed-point number $x$

**Output:** $\sin(x)$

$w, b_1, b_2 \leftarrow \text{trigsub}(x)$  #additional method, see Algorithm 5
$v \leftarrow w \cdot \frac{2}{\pi}$    #put $v$ 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
$y \leftarrow v \cdot p_{3302}(v \cdot v)$    #approximation polynomial, see B.4

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 $p_{3504}$. 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:** $\cos(x)$

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

return $b \cdot y$

The behavior of this method is as expected and satisfies the pre-determined goals. Notable is that there is no accuracy loss after computing with the approximation.

### 5.9 Security

The security of the above shown protocols is determined using the security requirements of Section B.2.4.
Correctness

All protocols are tested for correctness as described above. The results are sufficient, given the context of this thesis. Therefore, the protocols form useful building blocks for differential private mechanisms.

Privacy

All protocols consist of existing building blocks that have been designed to be oblivious, hence they leak no information. The additional protocols are designed in a similar fashion. Only the counters are public parameters, hence the new protocols leak no secret information. Therefore the protocols are secure in an honest-but-curious model.
Chapter 6

Mechanisms

In this chapter, some preliminaries are given on the mathematical background of sampling a random variable from a certain distribution. Then the theory is put into practice for constructing the secure protocols for the noise generating mechanisms with the special property of being $\epsilon$-differentially private. Some of these protocols are optimized for the MPC setting.

6.1 Mathematics of inverse sampling

In order to generate noise, random variables from a certain distribution have to be sampled. A way to do this is by the general inverse transform sampling method. Briefly, this is a method for generating numbers from a probability distribution using its inverted Cumulative Distribution Function (CDF). The CDF is defined as

$$F(x) = \Pr[X \leq x] = \int_{-\infty}^{x} \Pr[X = x]$$

Then the following steps are to be performed:

1. Determine the inverse $F^{-1}(x)$ of $F(x)$ such that $F(F^{-1}(x)) = x$.
2. Generate uniformly random variable $u$ on interval $(0,1)$.
3. The sampled random variable $X$ is then computed by $X = F^{-1}(u)$.

Proof. To show this method works, the samples produced by this method must follow the distribution with CDF $F(x)$. Suppose sample $X$ generated by this method using uniform random variable $u$. Then the following holds:

$$\Pr[X \leq x] = \Pr[F^{-1}(u) \leq x]$$
$$= \Pr[u \leq F(x)]$$
$$= F(x)$$

The second line follows from the fact that $F$ is an increasing function by definition and the third line follows from the fact that $0 \leq F(x) \leq 1$ and the CDF of a uniform random variable $u$ is $F_u(y) = y$ for all $y \in [0,1]$. □
6.1.1 Inverse sampling from a Laplace distribution

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

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

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

$$\frac{1}{2} \exp\left(\frac{F^{-1}(x)}{\lambda}\right) = x \Leftrightarrow$$

$$\frac{F^{-1}(x)}{\lambda} = \ln(2x) \Leftrightarrow$$

$$F^{-1}(x) = \lambda \cdot \ln(2x)$$

and

$$1 - \frac{1}{2} \exp\left(-\frac{F^{-1}(x)}{\lambda}\right) = x \Leftrightarrow$$

$$-\frac{F^{-1}(x)}{\lambda} = \ln(-2(x - 1)) \Leftrightarrow$$

$$F^{-1}(x) = -\lambda \cdot \ln(2x - 2)$$

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 uniformly random variable $u$ can take. The corresponding algorithm is given here with input $\lambda$ and output random variable $X$ following a Laplace distribution:

**Algorithm 8** Inverse sampling from Laplace distribution

**Input:** parameter $\lambda$

**Output:** R.v. $X \sim \text{Laplace distribution}$

Generate $U \sim \text{Unif}(0,1)$

if $U < 0.5$ then

$X \leftarrow \lambda \cdot \ln(2u)$

else

$X \leftarrow -\lambda \cdot \ln(2 - 2u)$

end if

return $X$
6.1.2 Inverse sampling from a Bernoulli distribution

For the Bernoulli distribution with parameter $p$, a sampled random variable has two possible outcomes: either 1 (with probability $p$) or 0 (with probability $1 - p$). Therefore, inverse sampling is easy using the following protocol with input parameter $p$ and output random variable $X$ following a Bernoulli distribution:

**Algorithm 9** Inverse sampling from Bernoulli distribution

- **Input:** parameter $p$
- **Output:** R.v. $X \sim$ Bernoulli distribution
  - Generate $U \sim Unif(0, 1)$
  - if $U \leq p$ then
    - $X \leftarrow 1$
  - else
    - $X \leftarrow 0$
  - end if
  - return $X$

6.1.3 Inverse sampling from a geometric distribution

For the geometric distribution with parameter $p$, the probability density function $f(x)$ and the cumulative density function $F(x)$, $x \in \mathbb{N}$ for random variable $X$ is given by respectively:

\[
  f(x) = \Pr[X = x] = (1 - p)^x p, \quad F(x) = 1 - (1 - p)^{x+1}
\]

In order to generate a random variable $X$, first a uniformly random variable $U$ from $(0, 1)$ is generated. Then the discrete version of the inverse transform sampling method is used to determine $F^{-1}(x)$. So $X$ is set to be $x$ if

\[
  \sum_{j=0}^{x-1} f(j) \leq U < \sum_{j=0}^{x} f(j) \iff
  \sum_{j=0}^{x-1} (1 - p)^j p \leq U < \sum_{j=0}^{x} (1 - p)^j p \iff
  F(x - 1) \leq U < F(x)
\]

Substituting results in:

\[
  1 - (1 - p)^x \leq U < 1 - (1 - p)^{x+1} \iff
  (1 - p)^{x+1} < 1 - U \leq (1 - p)^x
\]

Then

\[
  \ln(1 - U) \quad \ln(1 - p) - 1 < x \leq \frac{\ln(1 - U)}{\ln(1 - p)}
\]

This results in the following algorithm for sampling from the geometric distribution with input parameter $p$ and output random variable $X$ following a geometric distribution:
Combining Differential Privacy with MPC

Algorithm 10 Inverse sampling from geometric distribution

| Input: | parameter $p$ |
| Output: | R.v. $X \sim$ geometric distribution |

Generate $U \sim \text{Unif}(0,1)$

$X \leftarrow \left\lfloor \frac{\ln(1-U)}{\ln(1-p)} \right\rfloor$

return $X$

6.1.4 Inverse sampling from a standard Normal distribution

To sample from a Normal distribution, it suffices to sample from a standard Normal distribution. If one desires a random variable $Y$ from $\mathcal{N}(\mu, \sigma)$, the standard Normal sample $X$ is translated by $Y = \mu + \sigma \cdot X$. For sampling from $\mathcal{N}(0,1)$, there exist no closed form $F^{-1}(u)$. Therefore, other methods are consulted for inverse sampling from a Normal distribution. These are available spread over various lecture notes online. In this thesis, a selection of these methods are implemented and compared.

Acceptance-Rejection method

The idea is to sample from a more manageable distribution and accept a value with a certain probability. Suppose $X$ is the random variable sampled from some distribution with PDF $f(x)$. Then the following steps are to be performed:

1. Find an 'easy' invertible function $g(x)$ for which holds: $g(x) \geq f(x) \quad \forall x$.
2. Since $g(x)$ is not a PDF ($c = \int_{-\infty}^{+\infty} g(x)dx \geq \int_{-\infty}^{+\infty} f(x)dx = 1$), compute $r(x) = \frac{g(x)}{c}$.
   Now, $r(x)$ is a PDF!
3. Use the general inverse sampling method to sample $Y$ using $r(x)$ as the PDF.
4. Generate $U \sim \text{Unif}(0,1)$. If $U \leq \frac{f(Y)}{g(Y)}$, then $X = Y$, else go one step back and resample until acceptance.

There is a trade-off between the ease of choice of $r(x)$ and the minimization of the area in between $g(x)$ and $f(x)$. In case of sampling from $\mathcal{N}(0,1)$, $g(x) = \sqrt{2e/\pi}e^{-x^2/2}$ and the algorithm should accept if $U \leq e^{-\frac{(Y+1)^2}{2}}$ with $Y \sim \text{Exp}(1)$ with $\text{Exp}(1)$ the exponential distribution with parameter 1. Then $X = Y$ or $X = -Y$ with probability $\frac{1}{2}$. The implementation is stated in D.1

Box-Muller method

The idea is to transform samples to a space where sampling is easy. In this case, it is the Cartesian plane where

\[
\begin{align*}
    x &= r \cdot \cos(\theta) \\
    y &= r \cdot \sin(\theta)
\end{align*}
\]

are the coordinates with $r^2 = x^2 + y^2$ and $\tan(\theta) = \frac{y}{x}$. Assume a pair $(X,Y)$ with $X,Y \sim \mathcal{N}(0,1)$ independent. It can be shown that the PDF of $(X,Y)$ in terms of $r^2$ and $\theta$ is the joint density function of $\text{Exp}(\frac{1}{2})$ and $\text{Unif}[0,2\pi]$. Then the following steps are to be performed:
1. Sample $U_1, U_2 \sim Unif(0, 1)$

2. Compute $r^2 = -2 \ln(U_1)$ (derived from the inverse CDF of the exponential distribution) and $\theta = 2\pi \cdot U_2$

3. Compute $X = r \cos(\theta)$ and $Y = r \sin(\theta)$

The implementation is stated in [D.2]

**Marsaglia method**

To avoid using trigonometric functions, Marsaglia’s method would be a candidate. Then the following steps are to be performed:

1. Sample $W_1, W_2 \sim Unif[-1, 1]$ such that $V = W_1^2 + W_2^2 < 1$.

2. Compute $X = \sqrt{-2 \ln(V)} \cdot \frac{W_1}{\sqrt{V}}$ and $Y = \sqrt{-2 \ln(V)} \cdot \frac{W_2}{\sqrt{V}}$

The implementation is stated in [D.3]

**Irwin-Hall distribution**

A random variable following the Irwin-Hall distribution is given as $Y = \sum_{i=1}^{k} U_i$ with $U_i \sim Unif(0, 1)$ 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 where it states that $\sqrt{n}(S_n - \mu) = 1 \cdot (S_1 - 6) = Y - 6 \sim N(0, 1)$. Note that this is a crude approximation as the CLT normally holds for a sequence of random variables. The implementation is stated in [D.4]

**By approximation**

There exist a couple of approximations for the cumulative distribution function (CDF) $F^{-1}(x)$ of the standard Normal distribution. The following approximation has an absolute error $\leq 0.0045$:

$$X = \text{sgn} \left( U - \frac{1}{2} \right) \left( t - \frac{c_0 + c_1 t + c_2 t^2}{1 + d_1 t + d_2 t^2 + d_3 t^3} \right)$$

with $\text{sgn}(x) = \{1, 0, -1\}$ if $x$ is positive, zero or negative respectively, $t = \{-\ln[\min(U, 1 - U)]\}^{1/2}$ and coefficients $c_0 = 2.15517, c_1 = 0.802853, c_2 = 0.010328, d_1 = 1.432788, d_2 = 0.1189269, d_3 = 0.001308$. The implementation is stated in [D.5]

6.2 **Implementation of the Laplace Mechanism**

For the implementation in MPyC, it is desired that the execution of the protocol is independent from the actual value of the input. In Algorithm [8], a distinction is made between uniformly random variables $u$ that end up in $(0, 0.5)$ and $[0.5, 1)$. To avoid using a comparison for secret shared $u$, a random bit $b$ is generated that decides which case it is going to be.
Furthermore, $F^{-1}(x)$ can be rewritten in the following way:

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

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

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:

$$F^{-1}(y) = \lambda(\ln(y) - b(\ln(y) + \ln(1 - y))), y \in (0, 1)$$

(6.1)

This results in the following implementation with input Laplace parameter $\lambda$ and output a random variable $X$ following a Laplace distribution:

**Algorithm 11** Implementation of a method that samples from a Laplace distribution

**Input:** parameter $\lambda = \Delta/\epsilon$

**Output:** R.v. $X \sim$ Laplace distribution

1. $u \leftarrow \text{random(secfxp)}$ \#samples secret uniform r.v. with type fixed-point number
2. $b \leftarrow \text{random_bit(secfxp)}$ \#samples random bit with type fixed-point number
3. $X \leftarrow \lambda(\ln(u) - b(\ln(u) + \ln(1 - u)))$

**return** $X$

In the implementation, $u$ is sampled from the MPyC method $\text{mpc.random(secfxp)}$ which returns a random value in $[0.0, 1.0)$. Since $P[u \in (0, 0.5)] = P[u \in [0.5, 1.0]] \approx 1/2$, it is sufficient to generate a random bit with $Pr[b = 0.0] = Pr[b = 1.0] = 1/2$ and let it decide which case it will be. Note that $u = 1$ is not a valid input, but in the implementation of Algorithm 3 it will return a value instead of an error. Since the probability that $u = 1$ will be negligibly small, either scenario (returning a value versus returning an error) will not contribute to a failure of noise generating.

For validation purposes, the distribution is checked of the sampling method for 2000 random noise values displaying them in a **distplot** using the **seaborn** package of Python. The result displays the distribution of 2000 samples using the method $\text{random.Laplace}(0, 2, \text{size}=2000)$ of package **numpy** and the distribution of 2000 samples using the implementation 12 with parameter $\lambda = 2$ (see Figure 6.1). It is clear to see that both follow the same (Laplace) distribution approximately.

**Optimization**

In the setting of implementations, one aims for faster execution time while preserving validity and security. For the Laplace sampler, the current version of the algorithm could be replaced by Algorithm 12.
Figure 6.1: Distribution of 2000 Laplace samples from Python and MPyC’s method

Algorithm 12 Optimization of Algorithm 11

Input: parameter $\lambda = \Delta / \epsilon$
Output: R.v. $X \sim \text{Laplace distribution}$

$u \leftarrow \text{random(secfxp)}$

$b \leftarrow \text{random_bit(secfxp, signed=True)}$

$X \leftarrow b \cdot \ln(u)$

return $X$

The implementation of this algorithm results in the same distribution of samples as Algorithm 11. Furthermore, it has fewer natural logarithm evaluations than Algorithm 11, so the runtime is probably faster. Measuring the speed shows that the original sampler takes 0.30 seconds to generate a noise sample and the optimized sampler takes 0.17 seconds.

6.3 Implementation of the staircase mechanism

In 2014, the staircase mechanism was coined by Geng and Viswanath [12]. According to their research, the new mechanism could replace Laplace mechanism in any instance, while outperforming in any aspect. The probability density function of the staircase distribution is given as follows:

$$f_\gamma(x) = \begin{cases} a(\gamma) & \text{if } x \in [0, \gamma \Delta] \\ e^{-\epsilon} a(\gamma) & \text{if } x \in [\gamma \Delta, \Delta] \\ e^{-k\epsilon} \cdot f_\gamma(x - k\Delta) & \text{if } x \in [k\Delta, (k+1)\Delta) \text{ for } k \in \mathbb{N} \\ f_\gamma(-x) & \text{if } x < 0 \end{cases}$$

with $a(\gamma) = \frac{1-e^{-\epsilon}}{2\Delta(\gamma + e^{-\epsilon}(1-\gamma))}$, optimal setting for $\gamma = \frac{\sqrt{e^{-\epsilon}}}{1 + \sqrt{e^{-\epsilon}}}$ and $\Delta$ the sensitivity of the function. The inverse sample algorithm is also given in [12]. The input parameters are $\epsilon, \Delta$ and $\gamma \in [0,1]$. Then the random variable $X$ following the staircase distribution is

$$X \leftarrow S((1-B)((G + \gamma U)\Delta) + B((G + \gamma + (1-\gamma)U)\Delta))$$

with:
• $S$ the sign of the noise, generated with $\Pr[S = 1] = \Pr[S = -1] = \frac{1}{2}$.

• $G$ determining which interval $[G\Delta, (G+1)\Delta)$ the noise is in, generated from a geometric distribution with parameter $b = e^{-\epsilon}$

• $U$ a uniformly generated random variable in $[0, 1]$

• $B$ determining which subinterval of $[G\Delta, (G+\gamma)\Delta)$ and $[(G+\gamma)\Delta, (G+1)\Delta)$ the noise is in, generated with $\Pr[B = 0] = \frac{\gamma}{\gamma+(1-\gamma)b}$ and $\Pr[B = 1] = \frac{(1-\gamma)b}{\gamma+(1-\gamma)b}$

The distributions of $S$, $U$ and $B$ are trivial with respect to implementation. For geometric random variable $G$, the probability density function with parameter $b$ is expressed as follows:

$$\Pr[G = i] = (1 - b)b^i$$ for $i \in \mathbb{N}$

To match this with Algorithm 10, the parameter of this algorithm becomes $1 - b$ for $b = e^{-\epsilon}$.

The implementation of the noise samples in MPyC uses the implementations of the Bernoulli sampler and the geometric sampler implementation stated in C.5 and C.6 respectively.

To reduce the number of multiplications for faster runtime, the same reasoning as described in Section 3.3.2 is applied:

$$S \cdot ((1 - B) \cdot ((G + \gamma \cdot U) \cdot \Delta) + B \cdot ((G + \gamma + (1 - \gamma) \cdot U) \cdot \Delta)) =$$
$$S\Delta \cdot ((1 - B)(G + \gamma U) + B(G + \gamma + (1 - \gamma)U)) =$$
$$S\Delta \cdot (G + \gamma U - BG - B\gamma U + BG + B\gamma + BU - B\gamma U) =$$
$$S\Delta \cdot (G + \gamma U - 2B\gamma U + B\gamma + BU) =$$
$$S \cdot \Delta \cdot (G + \gamma \cdot U + B \cdot (-2 \cdot \gamma \cdot U + \gamma + U))$$

This results in a reduction of one multiplication. For the complete implementation, see Algorithm 13. To validate this algorithm, 2000 samples are generated suitable for count queries with $\Delta = 1$ and $\epsilon = 1$. They are plotted using Python’s seaborn package with distplot. In Figure 6.2, a normalized histogram of the samples is given along with a plot of 2000 samples drawn from a Laplace distribution. In this Figure, the staircase shape is clearly visible and it follows the shape of the Laplace distribution.

---

**Algorithm 13** Implementation of a method that samples from the staircase distribution

**Input:** privacy parameter $\epsilon$ and sensitivity $\Delta$

**Output:** R.v. $X \sim$ staircase distribution

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

$\gamma \leftarrow \frac{\alpha}{\sqrt{\alpha + 1}}$

$b1 \leftarrow \frac{(1-\gamma)\alpha}{\gamma+(1-\gamma)\alpha}$

$S \leftarrow$ random_bit(secexp, signed=True)

$G \leftarrow$ geometric$(1 - \alpha)$  \#samples as described in Section 6.1.3 see C.6

$U \leftarrow$ random(secexp)

$B \leftarrow$ bernoulli$(b1)$  \#samples as described in Section 6.1.2 see C.5

$B \leftarrow S \cdot \Delta \cdot (G + \gamma \cdot U + B \cdot (-2 \cdot \gamma \cdot U + \gamma + U))$

return $X$
Comparison with Laplace mechanism

In [12], it is claimed that the staircase distribution outperforms the Laplace in every way. In this section, the results of performance measured in time is given. Again, parameters $\epsilon = 1, \Delta = 1$ are used and the average time of generating 2000 noise samples is determined. Now, $\alpha, \gamma, b_1$ are precomputed before sampling staircase noise and the implementation of Algorithm 12 is used for Laplace noise sampling. On average, Laplace noise takes 0.14 seconds while the staircase noise takes 0.50 seconds. Probably this is due to the generation of randomness that takes up a lot of time.

A way to optimize the speed of the staircase generation, is to draw the random bits from a precomputed list or to vectorize the computation of the random bits. Such an implementation is beyond the scope of thesis.

6.4 Implementation of the geometric mechanism

The geometric mechanism is introduced by [13]. It is claimed to be optimal for especially count queries. In essence, it is a discretized version of the Laplace mechanism, so it adds integer noise values to the true answer of a query. In [3], a more general expression for this two-sided geometric distribution is given with scale parameter $s$ centered at $c \in \mathbb{Z}$. An integer random variable $Z$ is two-sided geometrically distributed, i.e. $Z \sim c + \text{Geo}(s)$ if its probability density function $f(z)$ is proportional to $e^{-|z-c|/s}$. From [3], it follows that the PDF and CDF is

$$f(z) = \frac{e^{1/s} - 1}{e^{1/s} + 1} \cdot e^{-|z-c|/s}, F(z) = \begin{cases} \frac{e^{1/s}}{e^{1/s} + 1} \cdot e^{-(c-z)/s}, z \leq c \\ 1 - \frac{1}{e^{1/s} + 1} \cdot e^{-(z-c)/s}, z > c \end{cases}$$

To sample from this distribution, one must derive an inverse sample function taking a uniformly distributed random variable $u$. This function is already stated in [3]:

$$F^{-1}(u) = c + \left[ s \cdot \text{sign}(1/2-u)(\ln(1 - |2u - 1|) + \ln(e^{1/s} + 1) - \ln(2)) \right] + \lfloor 2u \rfloor - 1$$

This method is proven to be $\epsilon/2$-differentially private for count queries. Since the geometric distribution should represent the discretized version of the Laplace distribution, 2000 samples
are generated of each distribution and the distribution of those sets of samples are compared in Figure 6.3.

This results in the following implementation in MPyC:

**Algorithm 14** Implementation that samples from the truncated geometric mechanism [3]

**Input:** privacy parameter $\epsilon$ and range length $N$

**Output:** R.v. $X \sim$ truncated two-sided geometric distribution

1. $u \leftarrow \text{random}($secfxp$)$
2. $m \leftarrow \text{ge}($secfxp$(0), 0.5 - u)$
3. $n \leftarrow 1 - 2 \cdot m$
4. $s \leftarrow 2 / \epsilon$
5. $X \leftarrow \text{trunc}((s \cdot m \cdot (\ln(1 - n \cdot (2 \cdot u - 1)) + \ln(e^{\frac{s}{2}} + 1) - 0.6931471805599453094172321214581)) + 1) + \text{trunc}(2 \cdot u - 1)$

**return** $X$

In [13], it is pointed out that the geometric mechanism in the current form could potentially put out values in $(-\infty, \infty)$ making the noise destroy the utility of the query output. A solution for that is to use a mapping that does not decrease the level of privacy. This is called post-processing and this does not degrade the privacy level, due to Theorem 2.2.4. The mapping used here truncates every noise value that is bigger than a certain threshold $N$ to $N$ and every noise value below zero is truncated to 0.

**Optimization**

The implementation of the inverse sample method of [3] requires two comparisons (see the second and third line of Algorithm 14), which are relatively expensive. In order to avoid this, 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 PDF \( f(x) = \frac{1-\alpha}{1+\alpha} \alpha^{x}, \alpha = e^{-\epsilon} : \)

\[
\sum_{|x| < i} f(x) \leq U < \sum_{|x| < i+1} f(x) \iff \sum_{|x| < i} \frac{1-\alpha}{1+\alpha} \alpha^{x} \leq U < \sum_{|x| < i+1} \frac{1-\alpha}{1+\alpha} \alpha^{x} \iff 1 - \frac{2\alpha^{i}}{1 + \alpha} \leq U < 1 - \frac{2\alpha^{i+1}}{1 + \alpha} \iff i = \pm \left\lfloor \left(\frac{\ln\left(\frac{(1-U)(1+\alpha)}{2}\right)}{-\epsilon}\right) \right\rfloor
\]

The third line follows from:

\[
\sum_{x \geq i} \frac{1-\alpha}{1+\alpha} \alpha^{x} = \frac{1-\alpha}{1+\alpha} \alpha^{i} \sum_{x \geq 0} = \frac{1-\alpha}{1+\alpha} \alpha^{i} \frac{1}{1-\alpha} = \frac{\alpha^{i}}{1+\alpha}
\]

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 \( b \) decides this case. Implementation of this sampler is given in Algorithm 15.

**Algorithm 15** Implementation of a new method that samples from the (two-sided) geometric distribution

**Input:** privacy parameters \( \epsilon \)

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

\[
\alpha \leftarrow e^{-\epsilon} \\
\text{random bit (secfxp, signed=True)} \\
U \leftarrow \text{random (secfxp)} \\
X \leftarrow s \cdot \text{trunc} \left( \frac{\ln\left(\frac{(1-U)(1+\alpha)}{2}\right)}{-\epsilon} \right)
\]

Sampling 2000 values with this implementation shows the same (two-sided) geometrical distribution as the original implementation, see Figure 6.4. Note that this implementation could be easily transformed into a truncated version by applied the same post-processing as in Algorithm 14.

Performance speed is compared between the two implementations and the new implementation resulted in an improvement of computation time. When sampling 100 noise values and averaging the execution time, the old method resulted in an average speed of 0.23 seconds, where the new method resulted in an average speed of 0.17 seconds.

### 6.5 Implementation of the Gaussian mechanism

The classical version of the Gaussian mechanism is given in Section 2.2.4. For the implementation, one needs to be able to sample from a standard Normal distribution in a fast and
secure way. Several methods were implemented based on the theory in Section 6.1.4. An overview of these implementations are to be found in Appendix D. Also, the complexity and sample speed is given in Table D.1 with the same parameters given in Section 5.3. Based on these analyses of the implementations of the methods, the Irwin-Hall distribution method fits best with the desired criteria (significant faster sample time with fewer expensive operations with respect to the other methods). In order to sample from the Gaussian mechanism, the following implementation is used:

**Algorithm 16** Implementation of a method that samples from the Gaussian distribution

<table>
<thead>
<tr>
<th>Input:</th>
<th>Sensitivity $\Delta_2$ (see Section 2.2.4), privacy parameters $\epsilon$ and $\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>R.v. $X \sim$ Gaussian distribution</td>
</tr>
</tbody>
</table>

$$ y \leftarrow \text{normalsampler4()} $$

#additional method to be found in Appendix D.4

$$ s \leftarrow \Delta_2 \cdot \sqrt{\frac{\ln(1.25/\delta)}{\epsilon}} $$

$$ X \leftarrow s \cdot y $$

return $X$

This method specifically shows how $\epsilon$ and $\delta$ influences the distribution of the random samples. In Figure 6.5 1000 samples were generated varying parameters $\epsilon \in \{0.5, 1, 2\}$ and $\delta \in \{0.05, 0.01, 0.1\}$. The graphs represent the distributions of a set of samples, given by Python’s package seaborn. It is clearly seen that the higher the value of $\epsilon$, the more likely that the noise value is relatively big and thus more privacy/less utility. Also, it is clearly seen that the higher the value of $\delta$, the more likely the noise will not mask the true answer of the query. This matches the definitions stated in Section 2.2.1.
Figure 6.5: The distribution of samples from the Gaussian mechanisms, varying $\epsilon$ and $\delta$
Chapter 7

Use cases for differential privacy

In this chapter, the concept of differential privacy and some of its aspects are illustrated with use cases. The first use case is a simplified but representative example on how differential privacy can hide the presence of an individual’s information in a database. The second use case shows examples of queries and what impact the choice of mechanisms has on the outcome. Furthermore, the concept of privacy budget is explained as well. The third use case is an example on how differential privacy and secure multiparty computation could be combined for a practical application.

7.1 Why does differential privacy work?

This case illustrates how differential privacy preserves the privacy of Bob as to whether or not he participated in Alice’s survey:

Alice is conducting a survey in order to investigate the relation between education and having a police record in a certain target group. Since the database resulting from the survey contains sensitive data about the respondents, privacy is desirable and Alice should not be able to get access to the raw data. There are already quite a few responses recorded in a database and the database itself is stored at a trusted data curator. The trusted data curator is equipped with a noise generating mechanism which is $\epsilon$-differentially private. When Alice queries the database, noise is added to the query result and then the noisy result is released to Alice.

The database contains the following attributes:

- **Education**: {Primary school, Secondary school, MBO, HBO, University’s degree, Other}. This entry represents the highest level of education of a respondent.
- **Police record**: {Y,N}. This entry represents the presence of a police record of a respondent.

To get some statistics on the responses of the survey, Alice asks the data curator to group the educational information and then asks for the number of Y’s in each group. Since this count query is done on disjoint datasets, each count is contained with i.i.d. noise $\sim$ Lap($\Delta/\epsilon$), $\epsilon = 0.5$ and the sensitivity of the query $\Delta$ is 1. The original counts, which Alice is not able to see, are given in Table 7.1. The results of the query, which Alice is able to see,
are given in Table 7.2. The results are noisy, i.e., the values displayed are the true answers to each count with a random number added to them. For this example, the Laplace mechanism is chosen to provide the noise.

Alice knows Bob from secondary school, she knows he did not pursue any other educations and she is able to determine that Bob belongs to the target group of the research. She then asks Bob to participate in the survey. Bob can do two things:

- Bob refuses to fill in the survey, leaving the dataset unchanged
- Bob fills in the survey, and increases the number of Y’s with 1.

Bob told Alice he has made his decision and Alice starts with her research. The goal of differential privacy is now to hide the decision of Bob. Alice queries the counts per educational group again.

The end results of the last query is given in Table 7.3. Note that it is impossible to reliably tell whether Bob participated in the survey or not, since the noise values could have been any value drawn from the Laplace distribution. Since there is no specific privacy budget and Alice is allowed to query whatever she wants, this example is somewhat simplified and only to illustrate how differential privacy works.

<table>
<thead>
<tr>
<th>Original counts</th>
<th>Intermediate counts</th>
<th>End counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary School</td>
<td>0</td>
<td>-2.61760291</td>
</tr>
<tr>
<td>Secondary school</td>
<td>5</td>
<td>6.61480475</td>
</tr>
<tr>
<td>MBO</td>
<td>24</td>
<td>24.62956658</td>
</tr>
<tr>
<td>HBO</td>
<td>9</td>
<td>5.8200702</td>
</tr>
<tr>
<td>University’s degree</td>
<td>2</td>
<td>1.35041081</td>
</tr>
<tr>
<td>Other</td>
<td>3</td>
<td>2.8987157</td>
</tr>
</tbody>
</table>

Table 7.1: Counts | Table 7.2: Noisy answer I | Table 7.3: Noisy answer II

A couple of observations are made based on this example:

- This example illustrates that differential privacy hides the existence of a person’s record, thus preserves its privacy.
- This example also illustrates that the results could be too noisy for practical uses.
- Discrete noise could be a better fit here.

7.2 Application of differential privacy to interesting scenarios

The following use case applies interesting concepts and ideas within differential privacy to a publicly available dataset with more than 30,000 records and 15 attributes. First, queries with known sensitivity are given. Second, the importance of choice of mechanisms is highlighted. Third, the difference between releasing appropriate noise on a query result versus database entries is analyzed and at last, the notion of privacy budget is explained by means of an example.
Table 7.4: Race counts

<table>
<thead>
<tr>
<th>Race</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amer-Indian-Eskimo</td>
<td>311</td>
</tr>
<tr>
<td>Asian-Pac-Islander</td>
<td>1039</td>
</tr>
<tr>
<td>Black</td>
<td>3124</td>
</tr>
<tr>
<td>Other</td>
<td>271</td>
</tr>
<tr>
<td>White</td>
<td>27816</td>
</tr>
</tbody>
</table>

Determining the sensitivity is quite a challenge and could be catastrophic to the utility of the response, as described in Section 2.2.6 and the previous use case. For these three types of queries, the sensitivity is known in advance:

- **COUNT query**: upon request, this query counts the number of a certain event in a dataset. The sensitivity $\Delta$ is 1, since removing a record influences the count with at most 1. Formally, $q_c(D) = \sum_{i=1}^{n} d_i$ with $d_i \in \{0, 1\}$, indicating the presence of the event, in a database $D \in \mathcal{D}^n$.

- **SUM query**: upon request, this query sums up certain values in a dataset. The sensitivity $\Delta$ is the maximum value in the database. Formally, $q_s(D) = \sum_{i=1}^{n} d_i$ with $d_i \in \{0, ..., M\}$ elements in a database $D \in \mathcal{D}^n$. Then $\Delta = M$.

- **AVG query**: upon request, this query computes the average of certain values in a dataset. The sensitivity $\Delta$ is the maximum value in the database over the total number of values. Formally, $q_a(D) = \frac{1}{n} \sum_{i=1}^{n} d_i$ with $d_i \in \{0, ..., M\}$ elements in a database $D \in \mathcal{D}^n$. Then $\Delta = \frac{M}{n}$.

The dataset used for the following experiments is well-known in the field of data science: In numerous papers, the Adult dataset is used to show various applications of differential privacy. This dataset is made available by the UCI Machine Learning Repository [7]. The detailed dataset description is given in Appendix E. The database is meant for experimenting on binary class classification and specifically to predict whether the income of a person exceeds 50K per year based on some census data. In the following experiments, it is used to examine relationships between sex, race, age, income and the number of hours per week they work. The examined subset of the Adult dataset consists of columns sex, race, age, income, hoursperweek. The values of column income is transformed in a binary representation, mapping '\(<= 50K\)' to 0 and '\(> 50K\)' to 1.

**Choice of noise generating mechanisms**

The first experiment is conducted on the effect of the choice for the mechanism. The Laplace mechanism and the geometric mechanism are considered as noise generating mechanisms that add noise to the true answer of a query. The true answers of the count of records per race is given in Table 7.4. In order to analyze the behavior of the noise and the mechanism that generates this, both Laplace and geometric noise is added to each of these counts with $\epsilon \in \{0.01, 0.05, 0.01, 0.2, 0.3, 0.4, 0.5\}$ and then the relative error of the noisy output with respect to the original count is computed.
Combining Differential Privacy with MPC

This information is useful, as the counts differ in size relatively. This is repeated 1000 times, and the means of the relative errors are displayed in Figure 7.1.

It is clear to see that both of the mechanisms behave the same with respect to the relative error, which makes sense since the geometric mechanism is the discretized version of the Laplace mechanism. Still, one could prefer integer output over a decimal output when it comes to count queries, see Section 7.1.

Another observation is that the relative error for smaller counts is likely to be bigger than the relative error for bigger counts. On one hand, this makes sense, as the noise values only depend on the privacy level and the sensitivity of \( q_c \) so the distribution is the same for every possible value for the count. On the other hand, \( \epsilon \) must be wisely chosen considering, for example, the feasible values for a certain count or the size of the database.

Noisy databases or noisy answers?

The second experiment shows the difference between two situations. For queries, \( q_s \) and \( q_a \), the two experiments are performed as follows:

- First, the setting of Figure 7.2 is created. Proportional noise is added to the data entries for various values of \( \epsilon \), resulting in a new database. Then the sum and average are queries from the new database.
- Second, the setting of Figure 7.3 is created. The original database is queried for the sum and average. Then the noise is added proportional to the query’s sensitivity for various values of \( \epsilon \).

For the first setting, noisy databases are generated. All entries are replaced by noisy entries, i.e., the sum of the original entry value and the noise drawn from the Laplace distribution using parameter \( \lambda = M/\epsilon \) with \( M = 100 \) (this is a reasonable estimate for the maximum value). The noise will hide the maximum difference between two neighboring databases and will therefore provide \( \epsilon \)-differential privacy. Applying noise this way is expected to destroy utility, and the following results will support this conclusion.

In Figure 7.4a and Figure 7.4b, the results are displayed for the experiments on the noisy databases for the mean and sum respectively. Here, 1000 noisy databases are generated and queried for the sum and mean of column \( \text{age} \) and the averaged absolute error from 1000
experiments is presented. The absolute error is a good measure to investigate whether adding noise to database entries will destroy the utility or not, compared to adding noise to the query result.

For the second setting, the original database is queried for its sum and mean of column age and appropriately chosen noise is added to the true answer. The averaged absolute error from 1,000 experiments is presented in Figure 7.4c and Figure 7.4d for the mean and sum respectively.

It is clearly visible how adding noise directly to the entries of a database could be disastrous for the utility. This makes sense, as the noise is effectively protecting against arbitrary queries instead of specific ones. Other reasons are: choice of query, choice of $\epsilon$ and the database itself. When using an upper bound, like the noise distribution used in this experiment, privacy is preserved to some degree, but the results are completely useless. The choice of the privacy budget $\epsilon$ has influence on the utility. This is clearly explained in Section 2.2.2. The optimal choice of $\epsilon$ is still a challenge and must be determined by, e.g., empirical experiments or by taking into account other factors (like the context of the research, feasible values etc.). If the database has a relatively big number of records $n$, it is easier to hide a single record. When $n$ is relatively small, the scale of the distribution must be enlarged accordingly and this could result in bad utility.

When comparing the graphs in Figure 7.4, it is important to note the scale. Though the behavior of the absolute error is similar, as expected and in accordance with the theory, the absolute distance to the true answer varies per choice of $\epsilon$. The goal of this experiment is to show that adding noise to the query result is a better idea for the utility than naively adding noise to the database entries. It would have been easier to release an $\epsilon$-differential private database and then not have to worry about a trusted third party processing the queries. Unfortunately, the utility is then affected significantly.
Combining Differential Privacy with MPC

(c) Original database queried for noisy mean  (d) Original database queried for noisy sum

Figure 7.4: Difference between querying original database and synthetic database
The privacy budget

The last interesting concept in the area of differential privacy is the notion of spending a certain privacy budget. The problem arises when a query covers multiple attributes. There is much more difference to hide than with querying just a simple statistic on a single attribute.

Suppose the data curator holds the database and uses a noise generating mechanism that preserves \( \epsilon \)-differential privacy for some fixed \( \epsilon \) by adding noise to the query result. The researchers are interested in two questions:

- How many women are in the dataset and how many of them earn more than 50K?
- How many white people work more than the average number of hours per week?

Suppose the privacy budget is fixed, namely \( \epsilon = 0.2 \). Note that the set of data needed to answer the first question and the set of data needed to answer the second question are disjoint. Both Theorems 2.2.2 and 2.2.3 apply to this situation.

For the first question, the number of women is needed. This is a simple count query, so the true answer will get noise \( N_1 \sim \text{Lap}(1/\epsilon_1) \) added to it. Similarly, the noise \( N_2 \) on the number of women that earns more than 50K is also Laplace distributed with parameter \( 1/\epsilon_2 \). Since they are both count queries, a reasonable suggestion is then to pick \( \epsilon_1 = \epsilon_2 = 0.1 \), which results in spending \( \max\{\epsilon_1, \epsilon_2\} = 0.1 \) of the privacy budget.

For the second question, the average number of hours per week must be computed and the result of that has noise \( N_3 \sim \text{Lap}(M/(n+\epsilon_3)) \) added to it. For \( M \), a reasonable estimate is proposed: considering the context, \( M = 80 \). Here, \( n \) is considered to be public. Then, for the number of white people, this is again a simple count query, so the added noise is \( N_4 \sim \text{Lap}(1/\epsilon_4) \) distributed.

A reasoning for picking the privacy parameters in a smart way would be to check the probability that the noise is out of some bound. For instance, take the last count query. Suppose it is not desirable to have an absolute distance bigger than \( k \) between the true answer and the response, i.e. \( |N_4| \geq k \). By the properties of the Laplace distribution, this probability is \( e^{-k\epsilon_4} \). Then the probability that this happens for \( k = 30 \) is 0.05 if \( \epsilon_4 = 0.1 \) is chosen. Since this is exactly the budget that is left, \( \epsilon_3 = 0.1 \) is chosen and that concludes the reasoning for the choice of \( \epsilon \).

In the case that the same (count) query is asked over and over by the data analysts, this privacy budget helps to protect the privacy, and especially it helps from noise cancelling out and revealing more information than was intended. For example, in case that the data analyst requested the same query multiple times (say \( k \) times), the data curator could return \((q_1(D) + \text{Lap}(k/\epsilon),...,q_k(D) + \text{Lap}(k/\epsilon))\), with \( q(D) = q_1(D) = ... = q_k(D) \) the true answer to the count query. Then, according to the sequential composition theorem, \( \epsilon \)-differential privacy is still guaranteed.
7.3 Logistic regressing with differential privacy and multiparty computation

This case describes a situation for which the combination of differential privacy and multiparty computation would benefit for all involved parties:

Suppose several hospitals are in possession of records about people entering the ICU with blood poisoning (sepsis). Their APACHE II score is being measured: a quantification that indicates the severity of the health condition. Its value varies from 0 to 71, computed based on several measurements. Higher scores correspond to more severe condition and a higher risk of death. The hospitals keep track of records of patients with their APACHE II score and an indication whether they died within 30 days or not. All the data combined could result in a useful indication for the probability that someone with a certain APACHE II score will die in 30 days. Two problems: hospitals might not want to share their data with the other hospitals as the data might affect their reputation. Also the privacy of the patients must be protected.

In this case, MPC could provide secure database sharing. This way, the hospitals will not have to reveal the actual entries of the database, but they can perform computations on them which will lead to the desired model.

Logistic regression

The model could be easily solved using logistic regression: a statistical method for predicting binary classes. It is a special case of linear regression, in which the target variable is categorical. Here, the categorical variable indicates the event. The binary representation of this is given in the databases with 0 in the event of survival after 30 days and 1 in the event of death within 30 days. Furthermore, suppose the database has \( k \) attributes and for each patient, there is a record consisting of \( k \) entries, which makes a record representable as a \( k \)-dimensional vector \( x = (x_1, ..., x_k) \). With linear regression the model is as follows:

\[
y = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k
\]

where \( y \) is the dependent variable and \( \beta = (\beta_0, ..., \beta_i, ..., \beta_k) \) the vector including the coefficients for the model that measures the relation between \( x \) and \( y \). Now, \( y \) is the categorical variable, so \( y \) should follow a Bernoulli distribution for a certain parameter \( p \). This is established by applying the Sigmoid function \( f(x) = 1/(1 - e^{-x}) \) to the model, resulting in:

\[
f(\beta, x) = \frac{1}{1 - e^{-\beta_0 - \beta_1 x_1 - ... - \beta_k x_k}}
\]

The estimation of \( \beta \) is done through maximum likelihood, resulting in solving the following optimization problem:

\[
f(\beta, x) > T
\]

for some threshold \( T \). The optimization problem can be solved using Newton iterations: \( \beta \) is approximated more accurately in every iteration until convergence is reached.
Combining MPC with DP

Complex computations following from the iterations are relatively expensive in an MPC setting. A strategy to optimize MPC computations is to open intermediate results, so that a significant part of the computation could be performed locally. A downside is that the intermediate results could leak information about the database.

This is where differential privacy comes in: if noise can be added to these intermediate $\beta$ results, privacy can be preserved while the opening of $\beta$ could still be used for the remaining iterations. For this to work, the assumption is made that the input $x$ is normalized. In the implementation of FRESCO’s logistic regression function \cite{11}, the logistic regression function is based on the above discussed idea, with the addition of a fixed number of iterations $N$. That way, the privacy budget is divided appropriately ($\frac{\epsilon}{N}$ for each iteration/opening). Every iteration, the noise generating mechanism adds a noise vector to the $\beta$ vector. This vector is based on the noise vector sample strategy suggested by \cite{6}. First, the norm and the direction of this vector is sampled according to a predefined distribution. Then a vector following the Laplace mechanism with that norm and direction is sampled. The behavior of the convergence of $\beta$ will be different due to the added noise in every round. So for some optimal value of $N$, the resulting model will likely need less iterations then existing algorithms and thus will be faster while resulting in a similar model.

This example of combining MPC with DP illustrates how both forces can be joined to not only preserve overall privacy, but also optimize the protocol in terms of speed. By applying differential privacy techniques, the complex computations of logistic regression can be done locally as they do not reveal any new information. This speeds up the protocol significantly. This use case shows the potential of both of the techniques and it could be considered for other complex MPC protocols as well.
Chapter 8

Final remarks

This chapter presents the conclusions of this project and some suggestions for further research are given.

8.1 Conclusions

The goal of this project 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 secure multiparty computation and differential privacy.

Chapter 2 and 3 serve as background information on the relevant concepts in this setting: differential privacy and secure multiparty computation. Differential privacy is a worst-case privacy guarantee, based on the generation of randomness, thus a suitable candidate for the input of this project. Secure multiparty computation is a way to securely compute and generated noise so that all computations on the distributed databases preserve the privacy of the individuals contained in the databases. It makes sense to look into combining both techniques.

Chapter 4 shows the two concepts can work together and the theory behind the combination is described extensively. Existing frameworks show some work that is already done in this field. Their protocols served as base for the protocols in this thesis. However, it does not necessarily mean that their implementations are optimal in the environment of MPyC: a Python-based package for secure multiparty computations.

Adapting the ideas in Chapter 4, building blocks for the randomness generation protocols are designed and implemented, given in Chapter 5. 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. Furthermore, the benefits of computing with fixed-point numbers is exploited and the number of operations are minimized:

- Evaluating \( f(x) = 2^x, x \in \mathbb{Z} \), is done by means of a bit decomposition on the input. The result is used to determine the positions of the 1’s.
• Evaluating $f(x) = e^x, x \in R$ for some feasible domain $R$, is done by changing to base 2 and splitting up the input in an integer part and a fractional part. Then the secure protocol above is evaluated using the integer part and an approximation function is evaluated using the fractional part. Multiplying the results yields the answer.

• Evaluating $f(x) = \ln(x), x \in R$ for some feasible domain $R$, is done by scaling down the input to $[\frac{1}{2}, 1]$ and evaluating an approximation function of the scaled input. Multiplying the approximation with a correction on the scaling yields the answer.

• Evaluating $f(x) = \sqrt{x}, x \in R$ for some feasible domain $R$, is done in a similar way as the previous protocol. In addition, a parity check is done in order to apply the right correction.

• Evaluating $f(x) = \cos(x), x \in \mathbb{R}$ and $f(x) = \sin(x), x \in \mathbb{R}$, is done by reducing the input to $[0, \frac{1}{2}\pi]$ while keeping track whether the input modulo $2\pi$ falls in $[0, \frac{1}{2}\pi], [\frac{1}{2}\pi, \pi]$ or $[\pi, 2\pi]$. Then the approximation is corrected using this information.

Chapter 6 presents research on sampling random variables. 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.

• The most classical example of a differential private noise generating mechanism is the Laplace mechanism. The output of the implementation follows the Laplace distribution.

• Securely generated random variables from the staircase mechanism [12] result in longer computation time than the Laplace mechanism, and thus it does not outperform the Laplace mechanism in this setting.

• The geometric mechanism samples discrete noise and its closed form is stated in [3]. Another approach of sampling from the same distribution is applied to increase performance speed and this has succeeded.

• Several methods for sampling from a standard Normal distribution are investigated. Based on the performance, the winning method is used for the implementation of the Gaussian mechanism.

Chapter 7 presents experiments that are done on databases using the implemented mechanisms and their privacy preserving behavior is analyzed by means of use cases. The following observations are made:

• Differential privacy works for hiding the existence of one individual’s record, thus preserves privacy, but it also potentially destroys the usability of the outcome.

• Challenging concepts like sensitivity and privacy budgets remain complicated, regardless of the MPC setting. In this case, the sensitivity ends up destroying the usability of results when the noise is applied directly to the entries of the database. Furthermore, some suggestions are made for spending the privacy budget.

• The combination of differential privacy and multiparty computation does have its benefits, although it is not straightforward to apply.
8.2 Further research

In this thesis, the idea of combining DP and MPC is elaborated. The contributions lie in the implementations of basic mechanisms and the underlying building blocks. Therefore, they serve as a base for further development of the combination. The following section provides ideas that were out of scope of this thesis, but deserve additional attention.

User-friendly application

Further development of a framework that both supports MPC and DP techniques could result in a hands-on application that provides secure communication between the data owners and data analysts. Eventually, a user-friendly interface is desired for querying distributed databases without violating privacy. For this extensive project, many challenges lie ahead. First, the application must be scalable for all possible database sizes, the magnitude of the entries and the number of parties holding the databases. Furthermore, sensible parameters for the noise generating mechanisms must be chosen to ensure privacy while at the same time maintaining some utility of the results. Moreover, there has to be dealt with the interactive setting of querying the databases: either a privacy budget must be fixed or a more suitable mechanism has to be chosen. Finally, the application has to run the protocols with reasonable speed.

Local sensitivity

As shown in Chapter 7, generating noise proportional to the global sensitivity could result in relative big values. A promising concept in the area of differential privacy is local sensitivity: local sensitivity calibrates the noise in such a way that it is proportional to the input. Then the magnitude of the noise is less likely to destroy the utility of a query result, since it is relatively small with respect to the database. A drawback is that the magnitude of the noise could possibly reveal too much about the input. Therefore, an optimal upper bound on the local sensitivity must be set so that privacy is still preserved. In [24], they claim that this bound will add input-specific noise with smaller magnitude than the worst-case noise involving the global sensitivity. This upper bound could be computed in an MPC setting based on all private databases. Noise is then calibrated using the result of this computation. The challenge here is to prevent leaking unintended information by computing these upper bounds and comparing them.

Adapting approximation polynomials

All polynomials used in the protocols of Chapter 5 are approximations of functions, valid for a certain interval with a fixed precision. Instead of hardcoding these polynomials in the protocols, the approximations could be computed 'on the fly' taking into account the characteristics of the input of the protocols. The protocol would generate the coefficients of the approximation polynomial based on the setting of the datatype of the input (bit-length, fractional length), the desired speed of a protocol (how many terms are needed), etc. The protocol could be a better fit for other applications depending on the desired qualities of the protocols. The challenge is to implement this in an efficient way with a reasonable performance speed while maintaining the desired accuracy and stability.
8.3 Recommendations

The design of the secure protocols in this thesis is based on a similar approach: reduce the input to a certain domain, evaluate some approximation function and correct its output with the information from the scaling. This approach sets an example for further development of secure protocols for other complex functions.

Combining secure multiparty computation and differential privacy does have its benefits. In Chapter 7, an example shows how differential privacy can increase performance speed for evaluating secure protocols. 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 secure multiparty computation 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.
Bibliography


Combining Differential Privacy with MPC


Combining Differential Privacy with MPC


[34] VIFF. http://viff.dk 2009.

## Appendix A

## Notation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Set of all positive integers including 0</td>
</tr>
<tr>
<td>$Z$</td>
<td>Set of all integers including 0</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>Set of all real numbers</td>
</tr>
<tr>
<td>$k, n, m, i, j, \ell$</td>
<td>Integers</td>
</tr>
<tr>
<td>$D$</td>
<td>Database</td>
</tr>
<tr>
<td>$D^n$</td>
<td>Database space containing $n$ records</td>
</tr>
<tr>
<td>$K$</td>
<td>Noise generating mechanism</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>Sensitivity of a query</td>
</tr>
<tr>
<td>$q$</td>
<td>Query</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Privacy budget</td>
</tr>
<tr>
<td>$X$</td>
<td>Random variable sampled from some distribution</td>
</tr>
<tr>
<td>$U, u$</td>
<td>Uniformly distributed random variable</td>
</tr>
<tr>
<td>$A$</td>
<td>Adversary</td>
</tr>
<tr>
<td>$\mathcal{P}_i$</td>
<td>$i$-th party in MPC protocol</td>
</tr>
<tr>
<td>$b$</td>
<td>Random bit with value 0 or 1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Linear regression coefficients</td>
</tr>
<tr>
<td>$l$</td>
<td>Bit length of fixed-point number</td>
</tr>
<tr>
<td>$f$</td>
<td>Fractional length of fixed-point number</td>
</tr>
<tr>
<td>$[a, b]$</td>
<td>Closed interval from $a$ to $b$ with $a, b \in R$ and $R$ some space</td>
</tr>
<tr>
<td>$(a, b)$</td>
<td>Open interval from $a$ to $b$ with $a, b \in R$ and $R$ some space</td>
</tr>
<tr>
<td>${a, ..., b}$</td>
<td>Set of elements, including $a, b \in R$ with $R$ some space</td>
</tr>
<tr>
<td>$[x]$</td>
<td>A value $x$ rounded down to its nearest integer</td>
</tr>
</tbody>
</table>
Appendix B

Approximation polynomials

In this Appendix, the tables with coefficients are given of the approximation polynomials of the book: Computer Approximations by Hart et al. [14]. The polynomials are given in the form

\[ p(x) = \sum_{i=0}^{n} p_i x^i \]

with subscript < ID > being the reference number of the polynomial in the book. Furthermore, in the implementation of the approximations, the polynomials follow Horner’s scheme to reduce the number of multiplications to \( n \), see Section 5.2. In the tables below, \( p_i \) is the \( i \)-th coefficient given in the form \( b \cdot 10^c \).

B.1 Approximation of \( \ln(x) \approx p(x) \) with \( x \in \left[ \frac{1}{2}, 1 \right] \)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( c )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
<td>-0.30674666858</td>
</tr>
<tr>
<td>1</td>
<td>+2</td>
<td>+0.1130516183486</td>
</tr>
<tr>
<td>2</td>
<td>+2</td>
<td>-0.2774666470302</td>
</tr>
<tr>
<td>3</td>
<td>+2</td>
<td>+0.5149518504454</td>
</tr>
<tr>
<td>4</td>
<td>+2</td>
<td>-0.6669583732238</td>
</tr>
<tr>
<td>5</td>
<td>+2</td>
<td>+0.5853503340958</td>
</tr>
<tr>
<td>6</td>
<td>+2</td>
<td>-0.3320167436859</td>
</tr>
<tr>
<td>7</td>
<td>+2</td>
<td>+0.1098927015084</td>
</tr>
<tr>
<td>8</td>
<td>+1</td>
<td>-0.161300738935</td>
</tr>
</tbody>
</table>

Table B.1: \( p_{2607} \)
B.2 Approximation of $\exp(x) \approx p(x)$ with $x \in [0, 1]$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$c$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
<td>+0.10000000077443021686</td>
</tr>
<tr>
<td>1</td>
<td>+0</td>
<td>+0.693147180426163827795756</td>
</tr>
<tr>
<td>2</td>
<td>+0</td>
<td>+0.2402251071017064605384</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>+0.55504068620466379157744</td>
</tr>
<tr>
<td>4</td>
<td>-2</td>
<td>+0.9618341225880462374977</td>
</tr>
<tr>
<td>5</td>
<td>-2</td>
<td>+0.1332730359281437819329</td>
</tr>
<tr>
<td>6</td>
<td>-3</td>
<td>+0.15510746059052573978</td>
</tr>
<tr>
<td>7</td>
<td>-4</td>
<td>+0.14197847399765606711</td>
</tr>
<tr>
<td>8</td>
<td>-5</td>
<td>+0.1863347724137967076</td>
</tr>
</tbody>
</table>

Table B.2: $p_{1045}$

B.3 Approximation of $\sqrt{x} \approx p(x)$ with $x \in \left[\frac{1}{2}, 1\right]$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$c$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>+0.22906994529</td>
</tr>
<tr>
<td>1</td>
<td>+1</td>
<td>+0.1300669049</td>
</tr>
<tr>
<td>2</td>
<td>+0</td>
<td>-0.9093210498</td>
</tr>
<tr>
<td>3</td>
<td>+0</td>
<td>+0.5010420763</td>
</tr>
<tr>
<td>4</td>
<td>+0</td>
<td>-0.1214683824</td>
</tr>
</tbody>
</table>

Table B.3: $p_{0132}$

B.4 Approximation of $\sin\left(\frac{1}{2}\pi x\right) \approx x \cdot p(x^2)$ with $x \in [0, 1]$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$c$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1</td>
<td>+0.1570796326794896619231320964</td>
</tr>
<tr>
<td>1</td>
<td>+0</td>
<td>-0.6459640975062462536555871056</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>+0.7692626246167045139353553</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>-0.4681754135318688008607342</td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>+0.1604411847873590411177681</td>
</tr>
<tr>
<td>5</td>
<td>-5</td>
<td>-0.35988432352085121801604</td>
</tr>
<tr>
<td>6</td>
<td>-7</td>
<td>+0.569217292094588814792</td>
</tr>
<tr>
<td>7</td>
<td>-9</td>
<td>-0.6688034921656659764</td>
</tr>
<tr>
<td>8</td>
<td>-11</td>
<td>+0.60669134335379795</td>
</tr>
<tr>
<td>9</td>
<td>-13</td>
<td>-0.437542099051245</td>
</tr>
<tr>
<td>10</td>
<td>-15</td>
<td>+0.2502654060171</td>
</tr>
</tbody>
</table>

Table B.4: $p_{3347}$
B.5 Approximation of $\cos(x) \approx p(x^2)$ with $x \in [0, \frac{1}{2}\pi]$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$c$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+0</td>
<td>+0.9999999999999999999999914771</td>
</tr>
<tr>
<td>1</td>
<td>+0</td>
<td>-0.4999999999999999999991637437</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>+0.416666666666666666530411988</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>-0.1388888888888888880310186415</td>
</tr>
<tr>
<td>4</td>
<td>-4</td>
<td>+0.2480158730158702330451573</td>
</tr>
<tr>
<td>5</td>
<td>-6</td>
<td>-0.27557319223932256421489</td>
</tr>
<tr>
<td>6</td>
<td>-8</td>
<td>+0.2087675698165412591559</td>
</tr>
<tr>
<td>7</td>
<td>-10</td>
<td>-0.114707451267755432394</td>
</tr>
<tr>
<td>8</td>
<td>-13</td>
<td>+0.47794543940649917</td>
</tr>
<tr>
<td>9</td>
<td>-15</td>
<td>-0.15612263428827781</td>
</tr>
<tr>
<td>10</td>
<td>-18</td>
<td>+0.39912654507924</td>
</tr>
</tbody>
</table>

Table B.5: $p_{3508}$
Appendix C

Supporting methods

This appendix displays all supporting methods written and used to execute the building block protocols of Chapter 5 and the noise generating mechanisms from Chapter 6.

C.1 Truncation

Method trunc(a) floors the input a to the nearest integer

```python
def trunc(a):
    f = type(a).field.frac_length
    return a - mpc.from_bits(mpc.to_bits(a, f)) * 2**-f
```

C.2 Evaluating $2^n$ for positive integers

Method twoe(n) computes $2^n$ for positive integers, only used for the method used in Section 5.5 for evaluating $e^x$ with $n = \lfloor |x| \rfloor$. This method is a shorter variant of two(n), see Section 5.4.

```python
@mpc.coroutine
async def twoe(n):
    stype = type(n)
    await mpc.returnType(stype)
    l = stype.bit_length
    Zp = stype.field
    f = Zp.frac_length
    n = await mpc.gather(n)
    n >>= f
    tobits = mpc.to_bits(stype(n),1-f)
    beginp = 1
    for i in range(0,f):
        t = tobits[i]
        beginp = beginp + (t) * (-beginp+beginp*2**(2**(i)))
    return beginp
```
C.3 Modified scaling method

In the libraries of MPyC, the method called \texttt{norm(a)} scales down any input number \( a \neq 0 \) to the interval \([\frac{1}{2}, 1]\) such that there is a \( c = \frac{1}{2^k} \) such that \( a \cdot c \in [\frac{1}{2}, 1] \). The modified version, given below, also outputs \( k \) as the second element in the output (\( c \) being the first).

```python
def anorm(a):
    x = mpc.to_bits(a)
    b = x[-1]
    s = 1 - b * 2
    x = x[:-1]
    _1 = type(a)(1)

def _norm(x):
    n = len(x)
    if n == 1:
        t = s * x[0] + b
        return 2 - t, t, 1-t
    i0, nz0, i2 = _norm(x[:n//2])
    i1, nz1, c = _norm(x[n//2:])
    i0 *= (1 << ((n + 1) // 2))
    i2 += (n + 1) // 2
    return mpc.if_else(nz1, [i1, _1, c], [i0, nz0, i2])

l = type(a).bit_length
f = type(a).field_frac_length
z = _norm(x)
return (s * z[0] * (2 ** (f - (1 - 1)))), (f-1)-z[2]
```

C.4 Exponential distribution sampler

This method samples \( X \sim \text{Exp}(\lambda) \) i.e. the random variable \( X \) has the exponential distribution with positive parameter \( \lambda \). The inverted cumulative distribution function of the exponential distribution is well-known and easy to implement. The probability density function is given as \( f(x) = \lambda e^{-\lambda x} \) for \( \lambda > 0 \). This sampler is needed for the implementation of the acceptance-rejection method for standard Normal sampling, given in Appendix \[D.1\].

```python
def sampleexp(l):
    y = secrand.random(secfxfp)
    elt = (-ln(1-y))/l
    return elt
```
C.5 Bernoulli distribution sampler

Section 6.1.2 contains the detailed description of this sampler.

```python
def bernoulli(p):
    q = secfxp(p)
    y = secrand.random(secfxp)
    b = mpc.ge(y, q)
    elt = 1 - b
    return elt
```

C.6 Geometric distribution sampler

Section 6.1.3 contains the detailed description of this sampler.

```python
def geometric(p):
    q = secfxp(p)
    y = secrand.random(secfxp)
    inp = ln(1-y)/ln(1-q)
    elt = trunc(inp)
    return elt
```
Appendix D

Inverse standard Normal sampler

In this appendix, all samplers from the standard Normal distribution are displayed, written for MPyC. One of them is used to sample from the Gaussian distribution in Section 6.5 based on important characteristics, also given in this appendix. The mathematical foundation of these samplers is to be found in Section 6.1.4.

D.1 Acceptance-rejection method

```python
def normalsampler1 ():
    y = sampleexp(1)
    x = (-y−1)*(y−1)/2
    z = exp(x)
    u = secrand.random(secfxp)
    b = mpc.ge(z,u)
    c = int(mpc.run(mpc.output(b)))
    if c == 1:
        bit = mpc.random_bit(secfxp)
        return 2*y*bit − y
    else:
        return normalsampler1()
```

D.2 Box-Muller

```python
def normalsampler2 ():
    u1 = secrand.random(secfxp)
    u2 = secrand.random(secfxp)
    theta = 2*3.1415926535897932384626433832795*u2
    r = sqrt(-2*ln(u1))
    x = r*(cos(theta))
    y = r*(sin(theta))
    return x,y
```
D.3 Marsaglia

```python
def normalsampler3():
    w1 = secrand.uniform(secfxp, -1.0, 1.0)
    w2 = secrand.uniform(secfxp, -1.0, 1.0)
    v = (w1*w1)+(w2*w2)
    b = mpc.ge(1, v)
    c = int(mpc.run(mpc.output(b)))
    if c == 1:
        s = sqrt(v)
        l = sqrt(-2*ln(v))
        x = l*(w1/s)
        y = l*(w2/s)
        return x, y
    else:
        return normalsampler3()
```

D.4 Irwin-Hall

```python
def normalsampler4():
    u1 = secrand.random(secfxp)
    u2 = secrand.random(secfxp)
    u3 = secrand.random(secfxp)
    u4 = secrand.random(secfxp)
    u5 = secrand.random(secfxp)
    u6 = secrand.random(secfxp)
    u7 = secrand.random(secfxp)
    u8 = secrand.random(secfxp)
    u9 = secrand.random(secfxp)
    u10 = secrand.random(secfxp)
    u11 = secrand.random(secfxp)
    u12 = secrand.random(secfxp)
    sum6 = u1+u2+u3+u4+u5+u6+u7+u8+u9+u10+u11+u12-6
    return sum6
```
Combining Differential Privacy with MPC

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Multiplications</th>
<th>Scalar Multiplications</th>
<th>Randomness</th>
<th>Speed (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.1</td>
<td>20l + 4θ + 3f + 32</td>
<td>9l − 5 log₂ l − 9</td>
<td>10l + 28f + 4θf + f² + 1</td>
<td>0.54</td>
</tr>
<tr>
<td>D.2</td>
<td>23l + 6f + 61</td>
<td>12l − 7 log₂ l − 12</td>
<td>18l + 54f + 4lf</td>
<td>1.21</td>
</tr>
<tr>
<td>D.3</td>
<td>51l + 2θ + 51</td>
<td>22l + 11 log₂ l − 22</td>
<td>15l + 48f + 4θf + 8lf</td>
<td>1.38</td>
</tr>
<tr>
<td>D.4</td>
<td>12</td>
<td>0</td>
<td>12l</td>
<td>0.02</td>
</tr>
<tr>
<td>D.5</td>
<td>23l + 33</td>
<td>10l − 5 log₂ l − 10</td>
<td>10l + 30f + 4lf</td>
<td>1.28</td>
</tr>
</tbody>
</table>

Table D.1: Complexity and speed of standard Normal inverse sample methods

Figure D.1: Distribution of 250 standard Normal samples generated by 6 different methods

D.5 Approximation

```python
def normalsampler5():
    u = secrand.random(secfxp)
    s = mpc.sgn(u−0.5)
    comp = [None]*2
    comp[0] = u
    comp[1] = 1−u
    m = mpc.min(comp)
    t = sqrt(-2*ln(m))
    b = -(2.515517+t*(0.802853+0.010328*t))/
        (1+t*(1.432788+t*(0.189269+0.001308*t)))
    z = s*(t+b)
    return z
```
Appendix E

UCI adult data description

The UCI’s adult dataset is a well-known dataset in the field of data mining and machine learning. Acquired from the US Census data (1994), it was made available in 1996 for research purposes. Its purpose is to predict whether an individual’s income exceeds 50K per year based on a collection of its attributes. It has 32561 entries and 15 attributes, and it is available at https://archive.ics.uci.edu/ml/datasets/adult. On the next page, a detailed overview of the dataset is given along with the type of data and its possible values.
Table E.1: Dataset description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Element description</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>Numerical</td>
<td>17, ..., 90</td>
</tr>
<tr>
<td>fnlwgt</td>
<td>Numerical</td>
<td>12285, ..., 1490400</td>
</tr>
<tr>
<td>education</td>
<td>Categorical</td>
<td>1st-4th, 5th-6th, 7th-8th, 9th, 10th, 11th, 12th, Assoc-acdm, Assoc-voc, Bachelors, Doctorate, HS-grad, Masters, Preschool, Prof-school, Some-college</td>
</tr>
<tr>
<td>education-num</td>
<td>Numerical</td>
<td>1, ..., 16</td>
</tr>
<tr>
<td>marital-status</td>
<td>Categorical</td>
<td>Divorced, Married-AF-spouse, Married-civ-spouse, Married-spouse-absent, Never-married, Separated, Widowed</td>
</tr>
<tr>
<td>relationship</td>
<td>Categorical</td>
<td>Husband, Not-in-family, Other-relative, Own-child, Unmarried, Wife</td>
</tr>
<tr>
<td>race</td>
<td>Categorical</td>
<td>Amer-Indian-Eskimo, Asian-Pac-Islander, Black, Other, White</td>
</tr>
<tr>
<td>sex</td>
<td>Categorical</td>
<td>Female, Male</td>
</tr>
<tr>
<td>capital-gain</td>
<td>Numerical</td>
<td>0, ..., 99999</td>
</tr>
<tr>
<td>capital-loss</td>
<td>Numerical</td>
<td>0, ..., 99999</td>
</tr>
<tr>
<td>hours-per-week</td>
<td>Numerical</td>
<td>0, ..., 9</td>
</tr>
<tr>
<td>native-country</td>
<td>Categorical</td>
<td>?, Cambodia, Canada, China, Columbia, Cuba, Dominican-Republic, Ecuador, El-Salvador, England, France, Germany, Greece, Guatemala, Haiti, Holand-Netherlands, Honduras, Hong, Hungary, India, Iran, Ireland, Italy, Jamaica, Japan, Laos, Mexico, Nicaragua, OutlyingUS(Guam-USVI-etc), Peru, Philippines, Poland, Portugal, PuertoRico, Scotlend, South, Taiwan, Thailand, Trinidad&amp;Tobago, UnitedStates, Vietnam, Yugoslavia</td>
</tr>
<tr>
<td>income</td>
<td>Categorical</td>
<td>&lt;= 50K, &gt; 50K</td>
</tr>
</tbody>
</table>