



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Thesis Type

**This is  
the Title**

Student Name

December 9, 2021



**ENCRYPTO**  
CRYPTOGRAPHY AND  
PRIVACY **ENGINEERING**

Cryptography and Privacy Engineering Group  
Department of Computer Science  
Technische Universität Darmstadt

Supervisors: M.Sc. Supervisor Name  
Prof. Dr.-Ing. Thomas Schneider

## **Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt**

Hiermit versichere ich, Student Name, die vorliegende Thesis Type ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

---

## **Thesis Statement pursuant to §23 paragraph 7 of APB TU Darmstadt**

I herewith formally declare that I, Student Name, have written the submitted Thesis Type independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, December 9, 2021

---

Student Name

## **Abstract**

This will be the abstract.

## **Acknowledgments**

If you like, you can add acknowledgments here.

# Contents

---

<b>1 Preliminaries</b>	<b>1</b>
1.1 Differential Privacy . . . . .	1
1.1.1 Traditional Methods for Privacy Preservation . . . . .	1
1.1.2 Randomized Response . . . . .	4
1.1.3 Differential Privacy Formalization . . . . .	5
1.1.4 Motivating Example of Differential Privacy . . . . .	7
1.1.5 Differential Privacy Mechanisms . . . . .	11
1.1.6 Properties of Differential Privacy . . . . .	12
1.1.7 Discussion about Differential Privacy . . . . .	13
<b>2 Design and Implementation of MPC-DP Protocols</b>	<b>16</b>
<b>3 Secure MPC-DP Protocol Implementations</b>	<b>19</b>
3.1 Algorithms Overview . . . . .	20
3.1.1 Snapping Mechanism . . . . .	20
3.1.2 Secure Noise Generation . . . . .	20
3.1.3 Discrete Gaussian Mechanism . . . . .	21
3.2 Notations . . . . .	21
3.3 MPC Techniques . . . . .	22
3.3.1 Branching . . . . .	22
3.3.2 Loop . . . . .	22
3.4 Snapping Mechanism . . . . .	24
3.4.1 Implementations of Snapping Mechanism . . . . .	25
3.4.2 MPC-DP Protocols . . . . .	29
3.5 Secure Noise Generation . . . . .	36
3.5.1 Laplacian Mechanism . . . . .	37
3.5.2 Gaussian Mechanism . . . . .	41
3.6 Discrete Gausisan Noise . . . . .	44
<b>List of Figures</b>	<b>46</b>
<b>List of Tables</b>	<b>47</b>
<b>List of Abbreviations</b>	<b>48</b>

<b>A</b>	<b>Appendix</b>	<b>49</b>
A.1	Algorithms . . . . .	49
A.2	MPC Protocols: Building Blocks . . . . .	53

1st chapter is "Introduction" and contains motivation, problem statement, & outline

# 1 Preliminaries

TODO: detailed introduce double precision floating point in preliminaries part

The IEEE 754 double-precision binary floating-point binary64 [Com19] (64 bits) is represented as:

$$(-1)^S (1.d_1 \dots d_{52})_2 * 2^{(e_1 \dots e_{11})_2 - 1023}$$

where  $S \in \{0, 1\}$  is the sign,  $d_i \in \{0, 1\}$  for  $i \in [52]$  and  $e_j \in \{0, 1\}$  for  $j \in [11]$ .  $(\cdot)_2$  is the binary representation of integers.  $(d_1, \dots, d_{52})$  is the significand field bits and  $e = (e_1 \dots e_{11})_2 - 1023$  is the biased exponent.

## 1.1 Differential Privacy

This section examines differential privacy in a formal mathematical view. We first briefly introduce classical privacy definitions and their limitations. Then, with an example of a questionnaire survey, we present differential privacy and formalize it. Further, we provide a discussion about differential privacy regarding properties and security.

### 1.1.1 Traditional Methods for Privacy Preservation

Suppose a technology company has collected massive data from billions of users and wants to make the data available to academic researchers such as data analysts. However, the data contains users' sensitive information such as financial information, medical records, or political ideologies. Because the technology company has an obligation to preserve the users' privacy, it must take specific measures before releasing the data.

A common attack is the re-identification attack [Swe97] that combine the released data with publicly available information to re-identify individuals. One traditional approach against re-identification attacks is to deploy methods that satisfy the  $k$ -anonymity definition [SS98] to anonymize the released data and prevent the data subjects from being re-identified. More specifically, the  $k$ -anonymity requires that for all individuals whose information appears in the dataset, each individual's information cannot be distinguished from at least  $k - 1$  other individuals. Tab. 1.1 shows medical records from a fictitious hospital of upstate New York. The attributes are divided into two groups: the non-sensitive attributes and the sensitive

you miss to mention that the data is anonymized before the attack

attribute. The value of the sensitive attributes must be kept secret for each individual in the records. We want to guarantee that no attacker can identify the patient and discover the *Condition* by combining the records with other publicly available information.

[SS98] introduces two techniques to achieve  $k$ -anonymity: data generalization and suppression. The former method makes the data less informative by mapping specific attribute values to a broader value range, and the latter method removes specific attribute values. After generalization, the values of attribute *Age* of the first eight records are replaced by value ranges such as  $< 30$  and  $\geq 40$ . The values of attribute *Nationality* are suppressed by being replaced with \*. Finally, the records in Tab. 1.2 satisfy the 4-anonymity requirement. For example, given one patient's non-sensitive attribute values (Zip Code: 130\*\*, Age:  $< 30$ ), there are at least three other patients with the same non-sensitive attribute values but with different *Condition* values (Heart Disease, Viral Infection).

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	13053	28	Russian	Heart Disease
2	13068	29	American	Heart Disease
3	13068	21	Japanese	Viral Infection
4	13053	23	American	Viral Infection
5	14853	50	Indian	Cancer
6	14853	55	Russian	Heart Disease
7	14850	47	American	Viral Infection
8	14850	49	American	Viral Infection
9	13053	31	American	Cancer
10	13053	37	Indian	Cancer
11	13068	36	Japanese	Cancer
12	13068	35	American	Cancer

Table 1.1: Inpatient microdata [MKG07].



	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	130**	< 30	*	Heart Disease
2	130**	< 30	*	Heart Disease
3	130**	< 30	*	Viral Infection
4	130**	< 30	*	Viral Infection
5	1485*	≥ 40	*	Cancer
6	1485*	≥ 40	*	Heart Disease
7	1485*	≥ 40	*	Viral Infection
8	1485*	≥ 40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

Table 1.2: 4 – anonymous inpatient microdata [MKGV07].

Definition  $k$  – *anonymity* alleviates re-identification attacks but is still vulnerable to the homogeneity attack and the background knowledge attack [MKGV07]. One example of the background knowledge attack is that, suppose we know one patient about thirty years old has visited the hospital and is in the records Tab. 1.2, then we could conclude that he has cancer. Afterward,  $l$  – *Diversity* [MKGV07] is proposed to overcome the shortcoming of  $k$  – *anonymity* by preventing the homogeneity of sensitive attributes in the equivalent classes. However, definition  $l$  – *Diversity* suffers from skewness and similarity attacks [LIV07]. Then in 2007, [LIV07] introduced the concept of  $t$  – *closeness* as an enhancement of  $l$  – *diversity*. However, [LIV09] shows that the  $t$  – *closeness* significantly affects the quantity of valuable information the released data contains.

Instead of releasing the anonymized data, a more promising approach is to limit the data analyst's access by deploying a trusted and trustworthy curator who manages all the individual's data in a database. The curator answers the data analysts' queries, protects each individual's privacy, and ensures that the database can provide statistically useful information. However, protecting privacy in such a system is nontrivial. For instance, the curator must prohibit queries targeting a specific individual, such as "Does Bob suffers from heart disease?". In addition, a single query that seems not to target individuals may still leak sensitive information when several such queries are combined. Instead of releasing the actual result, releasing approximate statistics can sometimes prevent the above attack. However, [DN03] shows that the adversary can reconstruct the entire database when sufficient queries are allowed and the approximate statistics error is bound to a certain level. Therefore, there are fundamental limits between what privacy protection can achieve and what useful statistical information it can provide. Finally, the problem turns into finding a theory that can interpret the relation between preserving privacy and providing valuable statistical information. Differential

privacy [Dwo06] is a robust definition that can support quantitative analysis of how much useful statistical information should be released while preserving a desired level of privacy.

### 1.1.2 Randomized Response

We will introduce differential privacy and start with a very early differentially private algorithm, the Randomized Response [DN03].

Suppose a psychologist wishes to study the psychological impact of cheating on high school students. The psychologist first needs to find out the number of students who have cheated. Undoubtedly, most students would not admit honestly if they had cheated in exams. More precisely, there are  $n$  students, and each student has a sensitive information bit  $X_i \in \{0, 1\}$ , where 0 denotes *never cheated* and 1 denotes *have cheated*. Every student want to keep their  $X_i$  secret, but they need to answer whether they have cheated. Then, each student send the psychologist an answer  $Y_i$  which may be equal to  $X_i$  or a random bit. Finally, the psychologist collects all the answers and tries to get an accurate estimation of the fraction of cheating students  $CheatFraction = \frac{1}{n} \sum_{i=1}^n X_i$ .

The strategy of students can be expressed with following formulas

$$Y_i = \begin{cases} X_i & \text{with probability } p \\ 1 - X_i & \text{with probability } 1 - p \end{cases} \quad (1.1)$$

Where  $p$  is the probability that student  $i$  honestly answers the question.

Suppose all students take the same strategy to answer the question either honestly ( $p = 1$ ) or dishonestly ( $p = 0$ ). Then, the psychologist could infer their sensitive information bit exactly since he knows if they are all lying or not. To protect the sensitive information bit  $X_i$ , the students have to take another strategy by setting  $p = \frac{1}{2}$ , i.e., each student either answer honestly or lie but with equal probability. In this way, the answer  $Y_i$  does not depend on  $X_i$  any more and the psychologist could not infer anything about  $X_i$  through  $Y_i$ . However,  $\frac{1}{n} \sum_{i=1}^n Y_i$  is distributed as a binomial random variable  $binom \sim \frac{1}{n} Binomial(n, \frac{1}{2})$  and completely independent of  $CheatFraction$ .

So far, we have explored two strategies: the first strategy ( $p = 0, 1$ ) leads to a completely accurate answer but not privacy preserving, the second strategy ( $p = \frac{1}{2}$ ) is perfectly private but not accurate. A more practical strategy is to find the trade-off between two strategies by setting  $p = \frac{1}{2} + \gamma$ , where  $\gamma \in [0, \frac{1}{2}]$ .  $\gamma = \frac{1}{2}$  corresponds to the first strategy where all students are honest, and  $\gamma = 0$  corresponds to the second strategy where everyone answers randomly. Therefore, the students can increase their privacy protection level by setting  $\gamma \rightarrow 0$  or provide more accurate result by setting  $\gamma \rightarrow \frac{1}{2}$ . To measure the accuracy of this strategy, we start with the  $Y_i$ 's expectation  $\mathbb{E}[Y_i] = 2\gamma X_i + \frac{1}{2} - \gamma$ , thus  $\mathbb{E}\left[\frac{1}{2\gamma} \left(Y_i - \frac{1}{2} + \gamma\right)\right] = X_i$ . For nature estimator  $\tilde{C} = \frac{1}{n} \sum_{i=1}^n \left[\frac{1}{2\gamma} \left(Y_i - \frac{1}{2} + \gamma\right)\right]$ , we have  $\mathbb{E}[\tilde{C}] = CheatFraction$ . The variance of  $\tilde{C}$  is

what's a natural estimator?

I have seen this example before  
→ give where you have it  
from

$$\text{Var}[\tilde{C}] = \text{Var}\left[\frac{1}{n} \sum_{i=1}^n \left[\frac{1}{2\gamma} \left(Y_i - \frac{1}{2} + \gamma\right)\right]\right] = \frac{1}{4\gamma^2 n^2} \sum_{i=1}^n \text{Var}[Y_i]. \quad (1.2)$$

With Chebyshev's inequality: For any real random variable  $Z$  with expectation  $\mu$  and variance  $\sigma^2$ ,

$$P(|X - \mu| \geq t) \leq \frac{\sigma^2}{t^2}, \quad (1.3)$$

we have

*can you show this step?*

$$|\tilde{C} - \text{CheatFraction}| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right), \quad (1.4)$$

where the error term  $|\tilde{C} - \text{CheatFraction}| \rightarrow 0$  as  $n \rightarrow \infty$  with high probability. The conclusion is that the error increases as the privacy protection level increases  $\gamma \rightarrow 0$ . To maintain the accuracy, more data  $n \rightarrow \infty$  is needed. To further quantify the privacy and accuracy, we need to define differential privacy.

### 1.1.3 Differential Privacy Formalization

For the formalization of differential privacy, we adapted the terms and definitions from [DR<sup>+</sup>14].

#### Terms and Definitions

*Database.* The database  $D$  consists of  $n$  entries of data from a data universe  $\mathcal{X}$  and is denoted as  $D \in \mathcal{X}^n$ . In the following, we will use the words database and dataset interchangeably.

Take Tab. 1.3 as an example. The database contains the names and exam scores of five students. The database is represented by its rows. The data universe  $\mathcal{X}$  contains all the combinations of student names and exam scores.

Name	Score
Alice	80
Bob	100
Charlie	95
David	88
Evy	70

Table 1.3: Database example.

*Data Curator.* A data curator is trusted to manage and organize the database, and its primary goal is to ensure that the database can be reused reliably. In terms of differential

privacy, the data curator is responsible for preserving the privacy of individuals represented in the database. The curator can also be replaced by cryptographic protocols such as secure multiparty protocols [GMW19].

*Adversary.* The adversary plays the role of a data analyst interested in learning sensitive information about the individuals in the database. In differential privacy, any legitimate data analyst of the database can be an adversary.

**Definition 1.1.1** (Privacy Mechanism [DR<sup>+</sup>14]). *A privacy mechanism  $M : \mathcal{X}^n \times \mathcal{Q} \rightarrow \mathcal{Y}$  is an algorithm that takes databases, queries as input and produces an output string, where  $\mathcal{Q}$  is the query space and  $\mathcal{Y}$  is the output space of  $M$ .*

The query process is as Fig. 1.1 shows, a data curator manages the database and provides an interface that deploys a privacy mechanism for a data analyst/adversary to query. After the querying, the data analyst/adversary receives an output.

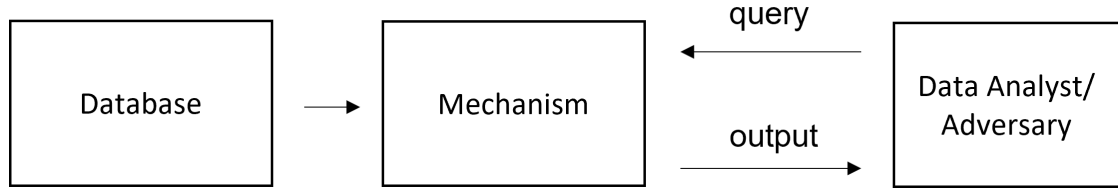


Figure 1.1: DP setting.

**Definition 1.1.2** (Neighboring Databases [DR<sup>+</sup>14]). *Two databases  $D_0, D_1 \in \mathcal{X}^n$  are called neighboring if they differ in exact one entry. This can be expressed as  $D_0 \sim D_1$ .*

**Definition 1.1.3** (Differential Privacy [DR<sup>+</sup>14]). *A privacy mechanism  $M : \mathcal{X}^n \times \mathcal{Q} \rightarrow \mathcal{Y}$  is  $(\epsilon, \delta)$ -differential privacy if for any two neighboring databases  $D_0, D_1 \in \mathcal{X}^n$ , and for all  $T \subseteq \mathcal{Y}$ , we have  $\Pr[M(D_0) \in T] \leq e^\epsilon \cdot \Pr[M(D_1) \in T] + \delta$ , where the randomness is over the choices made by  $M$ .*

Roughly, the differential privacy implies that the distribution of  $M$ 's output for all neighboring databases is similar.  $M$  is called  $\epsilon$ -DP (or pure DP) when  $\delta = 0$ , and  $(\epsilon, \delta)$ -DP (or approximate DP) when  $\delta \neq 0$ .

**Definition 1.1.4** ( $L_1$  norm). *The  $L_1$  norm of a vector  $\vec{X} = (x_1, x_2, \dots, x_n)^T$  measures the sum of the magnitudes of the vectors  $\vec{X}$  contains and denoted by  $\|\vec{X}\|_1 = \sum_{i=1}^n |x_i|$ .*

**Definition 1.1.5** ( $L_2$  norm). *The  $L_2$  norm of a vector  $\vec{X} = (x_1, x_2, \dots, x_n)^T$  measures the shortest distance of  $\vec{X}$  to origin point and denoted by  $\|\vec{X}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$ .*

**Definition 1.1.6** ( $\ell_t$ -sensitivity [DR<sup>+</sup>14]). The  $\ell_t$ -sensitivity of a query  $f : \mathcal{X}^n \rightarrow \mathbb{R}^k$  is defined as  $\Delta_t^{(f)} = \max_{D_0, D_1} \|f(D_0) - f(D_1)\|_t$ , where  $D_0, D_1$  are neighboring databases and  $t \in \{1, 2\}$ .

Recall the Differential Privacy definition 1.1.3 attempts to blur the contribution of any individual in the database using the notion of neighboring databases. Therefore, the sensitivity is a natural quantity when considering differential privacy since it calculates the upper bound of how much  $f$  can change when modifying a single entry.

**Definition 1.1.7** (Laplace distribution). The Laplace distribution with location parameter  $\mu = 0$  and scale parameter  $b$ , has the following probability density function:

$$\text{Laplace}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}. \quad (1.5)$$

$x \sim \text{Laplace}(x|b)$  denotes drawing a random variable  $x$  from a Laplace distribution.

**Definition 1.1.8** (Normal distribution). The normal (or Gaussian) distribution with mean  $\mu$  and standard deviation parameter  $\sigma$ , has the following probability density function:

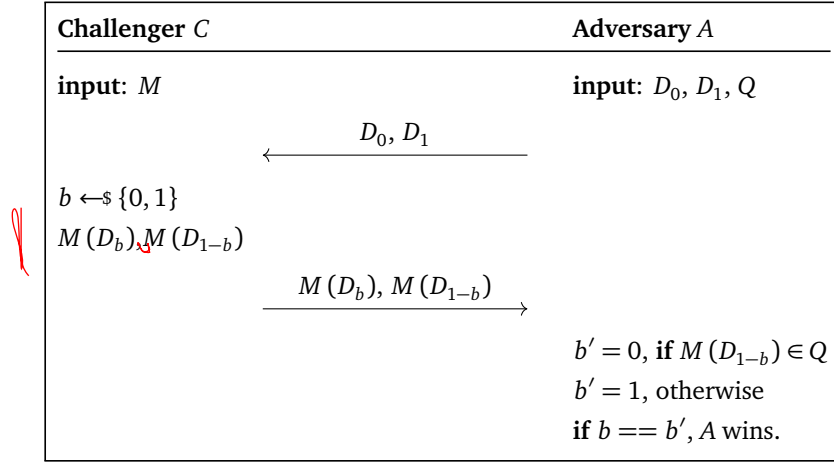
$$\text{Gauss}(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma}\right)^2}. \quad (1.6)$$

$x \sim \text{Gauss}(x|b)$  denotes drawing a random variable  $x$  from a Gaussian distribution.

### 1.1.4 Motivating Example of Differential Privacy

The previous example about randomized response § 1.1.2 indicates that we need DP to solve the trade-off problem between learning useful statistics and preserving the individuals' privacy. In other words, the psychologist wants to find the fraction of students who have cheated in the exam while guaranteeing that no students suffer from privacy leakage by participation in the questionnaire. To illustrate how DP solves such problems, we adapt the example from [Zum15]. Consider a game as Prot. 1.1 shows,

- A challenger implements a function  $M$  that can calculate useful statistical information. An adversary proposes two data sets  $D_0$  and  $D_1$  that differ by only one entry and a test set  $Q$ .
- Given  $M(D_0), M(D_1)$  in a random order, the adversary aims to tell apart  $D_0$  and  $D_1$ . If the adversary succeeds, the privacy is violated.
- The challenger's goal is to choose  $M$  such that  $M(D_0)$  and  $M(D_1)$  look similar to prevent from being distinguished by the adversary.
- $M$  is called  $\epsilon$ -differentially private iff:  $\left| \frac{\Pr[M(D_0) \in Q]}{\Pr[M(D_1) \in Q]} \right| \leq e^\epsilon$ .

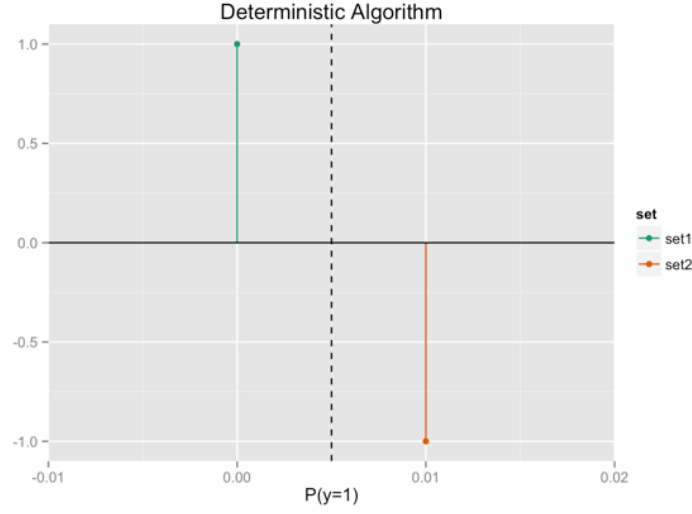
**Protocol 1.1:** A motivating example of differential privacy.

Suppose the adversary  $A$  has chosen two data sets:

- $D_0 = \{0, 0, 0, \dots, 0\}$  (100 zeros)
- $D_1 = \{1, 0, 0, \dots, 0\}$  (0 one and 99 zeroes).

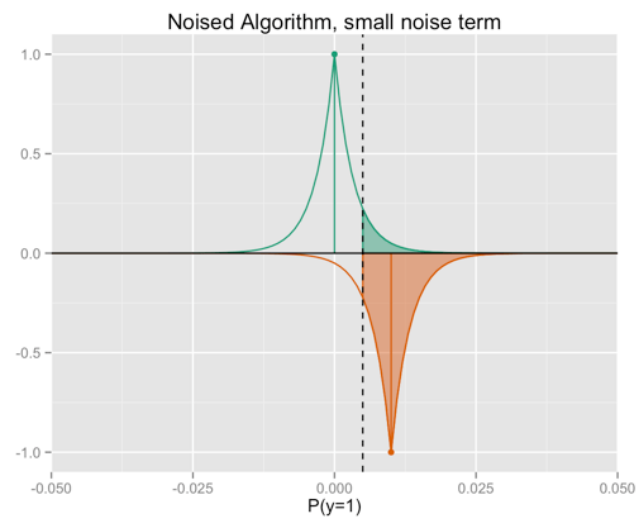
The testing set  $Q$  is an interval  $[T, 1]$ , where the threshold  $T$  is chosen by the adversary. The threshold  $T$  is set such that when the adversary has  $T < M(D) < 1$ , he knows  $M$  has input  $D = D_1$  (or  $D = D_0$ , when  $0 < M(S) \leq T$ ).

*The Deterministic Case.* Suppose the challenger wants to calculate the mean value of data sets and chooses  $M(D) = \text{mean}(D)$ . Since  $M(D_0) = 0$  and  $M(D_1) = 0.01$ , the adversary can set  $Q = [0.005, 1]$  and identify precisely the  $D$  used in  $M(D)$  every time they play the game. In Fig. 1.2, the green line represents the distribution of  $M(D_0)$ , whereas the orange line represents the distribution of  $M(D_1)$ . They are plotted upside down for clarity. The vertical dotted line represents the threshold  $T = 0.005$  which separates  $D_0$  and  $D_1$  perfectly.



**Figure 1.2:** Deterministic algorithm (need reproduce).

*The Indeterministic Case.* The challenger needs to take some measures to *blur* the difference between  $M(D_0)$  and  $M(D_1)$ . Suppose the challenger decides to add Laplace noise  $lap \sim \text{Laplace}(b = 0.05)$  to the result of  $M(D)$  as Fig. 1.3 shows. The shaded green region is the chance that  $M(D_0)$  will return a value greater than the adversary's threshold  $T$ . In other words, the probability that the adversary will mistake  $D_0$  for  $D_1$ . In contrast, the shaded orange area is the probability that the adversary identify  $D$  for  $D_1$ . The challenger can decrease the adversary's probability of winning by adding more noise as Fig. 1.4 shows, where the shaded green and orange areas are almost of the same size. Comparing  $M(D)$  with  $T$  is no longer reliable to distinguish  $D_0$  and  $D_1$ . In fact, we have  $\epsilon = \log\left(\frac{\text{green area}}{\text{orange area}}\right)$ , where  $\epsilon$  express the degree of differential privacy and a smaller  $\epsilon$  guarantee a stronger privacy protection. Although the challenger can add more noise to decrease the adversary's success probability, the mean estimation accuracy also decreases.



did you plot  
the figures yourself?  
otherwise  
cite /  
save

Figure 1.3: Indeterministic algorithm with small noise ( $b = 0.005$ ) (need reproduce).

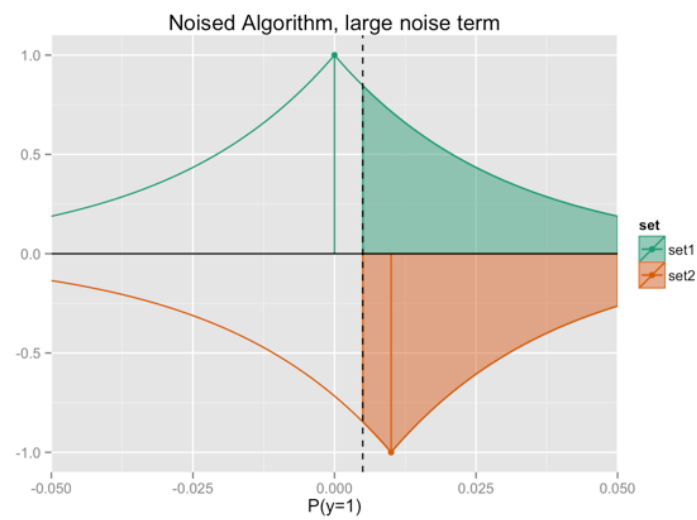


Figure 1.4: Indeterministic algorithm with large noise ( $b = 0.05$ ) (need reproduce).



### 1.1.5 Differential Privacy Mechanisms

Differential privacy is a formal framework to quantify the trade-off between privacy and the accuracy of query results. In this part, we introduce mechanisms to realize DP.

#### $\epsilon$ -Differential Privacy

**Definition 1.1.9** (Laplace Mechanism [DR<sup>+</sup>14]). Let  $f : \mathcal{X}^n \rightarrow \mathbb{R}^k$ . The Laplace mechanism is defined as  $M(X) = f(X) + (Y_1, \dots, Y_k)$ , where the  $Y_i$  are independent Laplace random variables drawn from distribution  $\text{Laplace}(Y_i|b) = \frac{1}{2b} e^{-\frac{|Y_i|}{b}}$  with  $b = \frac{\Delta f_i}{\epsilon}$ .

reference  
back to  
definition of  
sensitivity

**Theorem 1.** Laplace mechanism preserves  $\epsilon$ -DP [DR<sup>+</sup>14].

*Proof.* Let  $D_0$  and  $D_1$  be any two neighbouring databases that differs in one entry. Let  $Pr_{D_0}(z)$  and  $Pr_{D_1}(z)$  be the probability density functions of  $M(D_0)$  and  $M(D_1)$  evaluated at a point  $z \in \mathbb{R}^k$ . To prove differential privacy, it necessary to show that the ratio  $\frac{Pr_{D_0}(z)}{Pr_{D_1}(z)}$  is bounded by  $\epsilon$ , for any arbitrary  $z$  and neighboring  $D_0$  and  $D_1$ .

$$\begin{aligned} \frac{Pr_{D_0}(z)}{Pr_{D_1}(z)} &= \frac{\prod_{i=1}^k \exp\left(-\frac{\epsilon |f(D_0)_i - z_i|}{\Delta}\right)}{\prod_{i=1}^k \exp\left(-\frac{\epsilon |f(D_1)_i - z_i|}{\Delta}\right)} \\ &= \prod_{i=1}^k \exp\left(-\frac{\epsilon (|f(D_0)_i - z_i| - |f(D_1)_i - z_i|)}{\Delta}\right) \\ &\leq \prod_{i=1}^k \exp\left(\frac{\epsilon |f(D_1)_i - f(D_0)_i|}{\Delta}\right) \\ &= \exp\left(\frac{\epsilon \sum_{i=1}^k |f(D_1)_i - f(D_0)_i|}{\Delta}\right) \\ &= \exp\left(\frac{\epsilon \|f(D_1) - f(D_0)\|_1}{\Delta}\right) \\ &\leq \exp(\epsilon). \end{aligned}$$

□

**Definition 1.1.10** (Privacy Loss [DR<sup>+</sup>14]). Let  $X$  and  $Y$  be two random variables. The privacy loss random variable  $\mathcal{L}_{X||Y}$  is distributed by drawing  $t \sim Y$ , and outputting  $\ln\left(\frac{\Pr[X=t]}{\Pr[Y=t]}\right)$ .

The definition of *Privacy Loss* relies on the assumption that the supports of  $X$  and  $Y$  are equal, where  $\text{supp}(f) = \{x \in X : f(x) \neq 0\}$ . Otherwise, the privacy loss is undefined since  $\Pr\{Y = t\} = 0$ .

From the definition of  $\epsilon$ -DP, it is not difficult to see that  $\epsilon$ -DP corresponds to  $|\mathcal{L}_{D_0||D_1}|$  being bounded by  $\epsilon$  for all neighboring databases  $D_0, D_1$ . In other words,  $\epsilon$ -DP says that the absolute value of the privacy loss random variable is bounded by  $\epsilon$  with probability 1.

### $(\epsilon, \delta)$ -Differential Privacy

$\epsilon$ -DP has ~~high~~<sup>strong</sup> privacy requirements which leads to adding too much noise and affecting the accuracy of the queries. We introduce an relaxation of  $\epsilon$ -DP,  $(\epsilon, \delta)$ -DP.

**Definition 1.1.11** (Gaussian Mechanism [DR<sup>+</sup>14]). Let  $f : \mathcal{X}^n \rightarrow \mathbb{R}^k$ . The Gaussian mechanism is defined as  $M(X) = f(X) + (Y_1, \dots, Y_k)$ , where the  $Y_i$  are independent Gaussian random variables drawn from distribution  $\text{Gauss}(Y_i | \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{Y_i - \mu}{\sigma}\right)^2}$  with  $\mu = 0$ ,  $\sigma^2 = 2 \ln \left( \frac{1.25}{\delta} \cdot \left( \frac{\Delta_2^{(f)}}{\epsilon^2} \right)^2 \right)$ .

Gaussian mechanism is proved to satisfy  $(\epsilon, \delta)$ -DP [DR<sup>+</sup>14].

Similar to  $\epsilon$ -DP,  $(\epsilon, \delta)$ -DP can also be interpreted regarding privacy loss: the absolute value of the privacy loss random variable is bounded by  $\epsilon$  with probability  $1 - \delta$  [DR<sup>+</sup>14, Lemma 3.17]. In other words, with probability  $\delta$ , the privacy of databases is breached.

### 1.1.6 Properties of Differential Privacy

One reason for the success of differential privacy is its convenient properties which make it possible to deploy differential privacy mechanisms in a modular fashion.

#### Post-Processing

**Theorem 2.** Let  $M : \mathcal{X}^n \rightarrow \mathcal{Y}$  be  $(\epsilon, \delta)$ -DP mechanism, and let  $F : \mathcal{Y} \rightarrow \mathcal{Z}$  be an arbitrary randomized mapping. Then  $F \circ M$  is  $(\epsilon, \delta = 0)$ -DP [DR<sup>+</sup>14].

The Post-Processing properties implies the fact that once a database is privatized, it is still private after further processing.

## Group Privacy

**Theorem 3.** Let  $M : \mathcal{X}^n \rightarrow \mathcal{Y}$  be  $(\epsilon, \delta)$ -DP mechanism. For all  $T \subseteq \mathcal{Y}$ , we have  $\Pr[M(D_0) \in T] \leq e^{k\epsilon} \cdot \Pr[M(D_1) \in T] + \delta$ , where  $D_0, D_1 \in \mathcal{X}^n$  are two databases that differ in exactly  $k$  entries [DR<sup>+</sup>14].

Differential privacy can also be defined when considering two databases with more than one different entries. The larger privacy decay rate  $e^{k\epsilon}$  means a smaller  $\epsilon$  and more noise are necessary to guarantee the same level of privacy. entry difference

## Basic Composition

**Theorem 4.** Suppose  $M = (M_1 \dots M_k)$  is a sequence of  $(\epsilon_i, \delta_i)$ -differential private mechanisms, where  $M_i$  is chosen sequentially and adaptively. Then  $M$  is  $(\sum_{i=1}^n \epsilon_i, \sum_{i=1}^n \delta_i)$ -DP [DR<sup>+</sup>14].

Basic Composition provides a way to evaluate the overall privacy when  $k$  privacy mechanisms are applied on the same dataset and the results are released.

### 1.1.7 Discussion about Differential Privacy

#### Local and Central Differential Privacy

Since DP is a definition rather than a specific algorithm, there are many ways to realize DP. Two common modes of DP are centralized differential privacy [DR<sup>+</sup>14] and local differential privacy [DN03].

In centralized DP, all data is stored centrally and managed by a trusted curator before the differentially private mechanism is applied. As Fig. 1.5 shows, the raw data from clients are first collected in a centralized database, then the curator applies the privacy mechanism and answers the queries  $f(x)$  with  $f'(x)$ . The local DP mode is, as Fig. 1.6 shows, where the clients first apply a privacy mechanism on the data, then send the perturbed data to the curator. An advantage of local DP mode is that no trusted central curator is needed since the data is perturbed independently before sending. However, the disadvantage is that the collected data may contain too much noise and decrease the utility. VS

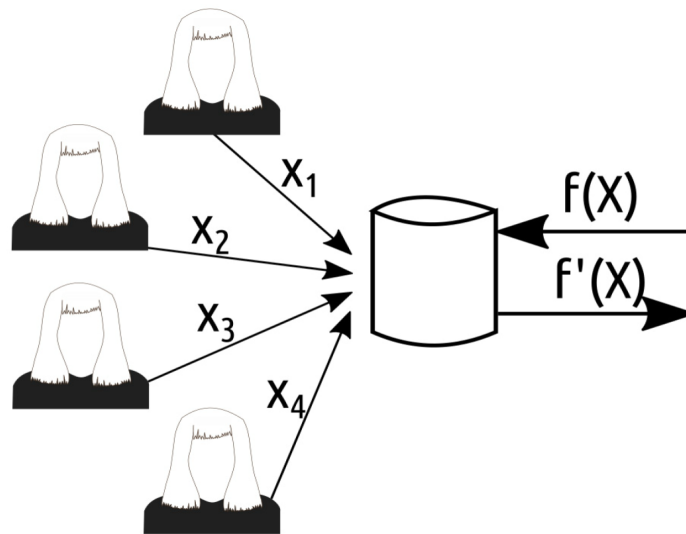


Figure 1.5: Centralized DP mode (need reproduce).

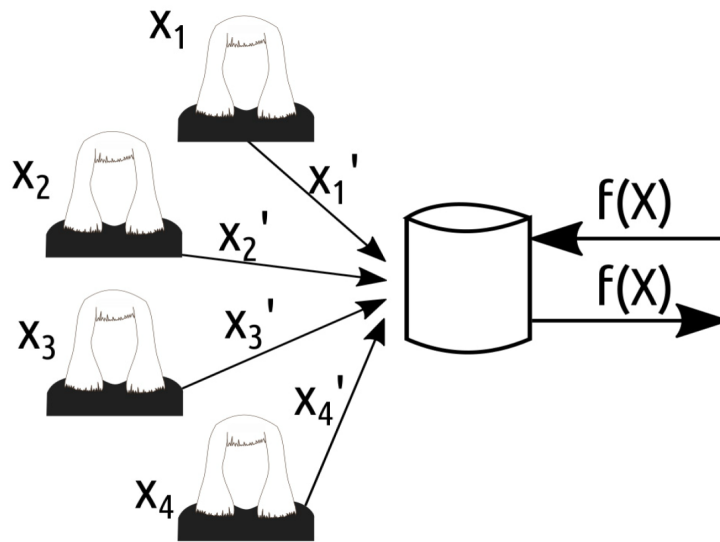


Figure 1.6: Local DP mode (need reproduce).

### Advantages of Differential Privacy

From the example § 1.1.4, we found that DP can still protect privacy even if the adversary knows the database. Generally speaking, DP ensures privacy protection by making no assumption about the adversary's auxiliary information (even when the adversary is the data provider) or computational strategy (regarding the complexity of modern cryptography) [Vad17]. In addition, DP provides a quantitative theory about safely releasing data and maintaining certain level of accuracy.

### Challenges of Differential Privacy

DP provides a method to guarantee and quantify individual privacy at the theoretical level. However, it faces a series of practical challenges.

*Sensitivity Calculation.* For certain types of data, the sensitivity is not difficult to calculate. Take a database with human ages as an example, the ages should be bounded between 0 and 150 (longest human lifespan is 122 years and 164 days according to [Whi97]). However, the data with an unbounded value range brings great challenges. A common way is to roughly estimate the value range and limit the data within that range. For example, if the value range estimation is  $[a, b]$ , then all values smaller than  $a$  are replaced by  $a$  and all values bigger than  $b$  are replaced by  $b$ . Finally, the sensitivity is  $\frac{b-a}{n}$ . In other words, the estimation decides the sensitivity. However, if value range  $[a, b]$  is chosen too wide, the utility is potentially destroyed because of the large noise. If the value range  $[a, b]$  is chosen too narrow, the utility is also potentially destroyed because too many values beyond  $[a, b]$  are replaced.

*Implementation of DP mechanisms.* The theory of DP operates on the real number field. However, because of the actual machine's limitation, the implementation of DP mechanisms based on floating-point or fixed-point number only provides an approximation of the mathematical abstractions. [Mir12] shows the irregularities of floating-point implementations of the Laplacian mechanism results in a porous distribution over double-precision number and leads to the breach of differential privacy. Further, [GMP16] prove that any DP private mechanism that perturbs data by adding noise can break the DP when implemented with a finite precision regardless of the actual implementation.

## 2 Design and Implementation of MPC-DP Protocols

---

In this chapter, we restate the research problem formally and introduce the MPC-DP protocol that combines secure multi-party computation protocols and differential privacy mechanisms.

Let a dataset  $D = (D_1, \dots, D_n)$  be distributed among the users  $(U_i)_{i \in [n]}$  who do not trust each other, where user  $U_i$  owns data  $D_i$ . The users  $(U_i)_{i \in [n]}$  wish to jointly compute a public function  $f$  on their private inputs  $(pre(D_i))_{i \in [n]}$  and get the result of  $f(pre(D_1), \dots, pre(D_n))$ , where  $pre$  stands for the pre-processing function of their local data  $(D_i)_{i \in [n]}$  before further operations. The users also require that:

1. The computation of  $f$  should reveal nothing but the result.
2. The computation result of  $f(pre(D_1), \dots, pre(D_n))$  should be exactly the same as the result, where a trusted server has access to the whole dataset  $D$  and compute  $f(pre(D))$  locally, i.e.,  $f(pre(D_1), \dots, pre(D_n)) = f(pre(D))$ .
3. The result needs to satisfy the differential privacy, i.e., an adversary cannot infer from the computation result if a particular user's data was used in the computation of  $f$ .

To fulfill the first two requirements, they can deploy a MPC protocol  $\Pi^f$  which takes inputs  $(pre(D_i))_{i \in [n]}$  and reveals only the computation result. For the third requirement, we can design a MPC-DP protocol which combines  $\Pi^f$  with a differential privacy mechanism. Note that in following text, we omit the data pre-processing procedure  $pre(\cdot)$  for simplicity and use  $f(D)$  to represent  $f(D_1, \dots, D_n)$  since the second requirement can be satisfied with MPC protocols.

Before describing the MPC-DP protocol, let's consider a trivial example that combines MPC protocols with DP mechanisms. Suppose the users  $(U_i)_{i \in [n]}$  wish to calculate the sum of their local data  $(D_i)_{i \in [n]}$  with a public function  $f(x_1, \dots, x_n) = \sum_{j=1}^n x_j$  (assume  $f$  has sensitivity  $\Delta_2^f = 1$ ). Meanwhile, the users require that only the computation result of  $f$  should be revealed and differential privacy must be satisfied. To achieve this, each user first adds noise  $r_i$  to their local data  $D_i$  and get  $y_i = D_i + r_i$  (satisfies  $(\epsilon, \delta)$ -DP), for  $r_i \sim \mathcal{N}(0, \sigma^2)$ ,  $\sigma^2 = \frac{2}{\epsilon^2} \cdot \ln\left(\frac{1.25}{\delta}\right)$  and  $i \in [n]$ . Then, they run the MPC protocol  $\Pi^f$  with their private inputs

$y_i$  to compute function  $f$  and after reconstruction they get:

$$\begin{aligned} f(y_1, \dots, y_n) &= \sum_{j=1}^n y_j \\ &= \sum_{j=1}^n (D_j + r_j). \end{aligned} \tag{2.1}$$

Because of the infinite divisibility of the normal distribution [PR96], we have  $\sum_{j=1}^n r_j \sim \mathcal{N}(0, \sigma_{sum}^2)$  for  $\sigma_{sum}^2 = \sum_{j=1}^n \sigma^2$ . Therefore, the computation result  $f(y_1, \dots, y_n)$  satisfies  $(\frac{\epsilon}{\sqrt{n}}, \delta)$ -DP.

In general, the users in above example can achieve  $(\frac{\epsilon}{\sqrt{n}}, \delta)$ -DP without considering the corrupted users. However, in realistic scenarios, corrupted users can help an adversary to infer the information of certain users by subtracting  $r_i$  from the result  $f(y_1, \dots, y_n)$ , where  $i$  is the index of the corrupted users. In the worst case when  $n - 1$  users are corrupted, the differential privacy guarantee of  $f(y_1, \dots, y_n)$  degenerates from  $(\frac{\epsilon}{\sqrt{n}}, \delta)$ -DP to  $(\epsilon, \delta)$ -DP. If it is necessary for the users to have  $f(y_1, \dots, y_n)$  satisfying  $(\frac{\epsilon}{\sqrt{n}}, \delta)$ -DP, one solution is to ask each user to add *stronger* noise  $r'_i \sim \mathcal{N}(0, n \cdot \sigma^2)$  locally before running  $\Pi^f$ . However, the excessive noise  $\sum_{j=1}^n r'_j$  in the result can lead to severe deviation which can reduce or completely break its utility.

We can see from the above example that the MPC-DP protocol needs a deeper integration of MPC protocols and DP mechanisms. To guarantee the practicality (utility, privacy protection) of the MPC-DP protocol, we propose several requirements on it by comparing it with the centralized data server scenario, i.e., a trusted server (has access to all the users' data  $D$ ) locally computes  $f(D)$  and adds noise to it to preserve differential privacy.

1. The MPC-DP protocol should achieve the same privacy protection level as in the centralized data server scenario, and the amount of noise  $r$  (in above example:  $r = \sum_{j=1}^n r_j$ ) should be no more than that in the centralized data server scenario.
2. The differential privacy guarantee of the MPC-DP protocol should not degenerate in the presence of corrupted users.

We design the MPC-DP protocol that can be used directly among  $n$  users, or under an outsourcing scenario, i.e., an arbitrary amount of users share their private inputs to  $N$  ( $N \ll n$ ) non-colluding computing parties. The private input shares look random to the computing parties since it is secretly shared. Then, the computing parties execute the  $N$ -party MPC-DP protocol, where they compute  $f$  and add noise shares to get the noisy secret-shared results. Finally, the noisy secret-shared results are sent back to the users for reconstruction. We assume semi-honest adversaries that follow the protocol specifications but wish to infer additional information.

Specifically, for  $i \in [n]$ , user  $U_i$  secret shares his inputs  $D_i$  to  $N$  computing parties  $P_0, \dots, P_N$ . For  $j \in [N]$ , computing party  $P_j$  runs MPC protocols for function  $f$  with other computing parties using his secret-shared value  $\langle D_1 \rangle_j, \dots, \langle D_n \rangle_j$  (received from  $n$  users) and get  $\langle f(D) \rangle_j$ . Then, computing party  $P_j$  runs the MPC protocols for distributed noise generation (DNG) with other computing parties and get a secret-shared noise  $\langle r \rangle_j$ . Finally, computing party  $P_j$  computes  $\langle M(D) \rangle_j = \langle f(D) \rangle_j + \langle r \rangle_j$  and sends it to the users responsible for the reconstruction. Note that during the distributed noise generation, each computing party only gets a secret-shared noise  $\langle r \rangle$  and even  $N - 1$  colluding computing parties cannot reconstruct the noise  $r$ . After the reconstruction, only the noisy result  $M(D) = f(D) + r$  is revealed which realizes function  $f$  and guarantees differential privacy. More important is that we prevent excessive noise and ensure utility since the noise  $r$  is generated by all the parties instead of by each party independently (as in the above example).

**Protocol:**  $\Pi^{MPC-DP}(\langle D_1 \rangle, \dots, \langle D_n \rangle)$

**Input:**  $\langle D_1 \rangle, \dots, \langle D_n \rangle$

**Output:**  $\langle M(D) \rangle = \langle f(D) \rangle + \langle r \rangle$

1 : Parties run  $\langle f(D) \rangle = \Pi^f(\langle D_1 \rangle, \dots, \langle D_n \rangle)$ .

2 : Parties run  $\langle r \rangle = \Pi^{DNG}$ .

3 : Parties compute  $\langle M(D) \rangle = \langle f(D) \rangle + \langle r \rangle$  and output  $\langle M(D) \rangle$ .

**Protocol 2.1:** General framework for MPC-DP protocol.

$\Pi^{MPC-DP}$  describes the general framework of the MPC-DP protocol. Note in  $\Pi^{MPC-DP}$ , the parties first calculate  $\langle f(D) \rangle = \Pi^f(\langle D_1 \rangle, \dots, \langle D_n \rangle)$  and add it with the distributed generated noise  $\langle r \rangle$  to guarantee differential privacy. In following text, we assume that each party has computed  $\langle f(D) \rangle$  already and focus more on the process of  $\Pi^{DNG}$ , i.e., the distributed generation of noise.

**TODO:** introduction MPC-DP protocols of textbook(unsecure) implementations



### 3 Secure MPC-DP Protocol Implementations

---

In this chapter, we explore the *secure* implementation (under floating-point arithmetic) of MPC-DP protocols that combine secure multi-party computation protocols and differential privacy mechanisms.

Recall that differential privacy mechanisms [DMNS06; DKM<sup>+</sup>06]  $M(D)$  guarantees differential privacy by adding appropriately chosen random noise to the query function  $f(D)$  of database  $D$ , which can be expressed as:

$$M(D, f(D)) = f(D) + Y,$$

where  $Y$  is the noise term and its calculation depends on the *sensitivity* of  $f$ , type of the sampling probability distribution, level of the desired differential privacy strength.

Meanwhile, the security analysis of differential privacy mechanisms are based on following assumptions:

1. Computations are performed on the set of real numbers which usually require machines to have infinite precision.
2. The sampling of noise from a probability distribution needs to be precise.

However, for the practical implementations of differential privacy mechanisms, we only have machines with finite precision and typically use the floating-point arithmetic to approximate calculations of real numbers. [Mir12] shows that the irregularities of floating-point implementations of the Laplacian mechanism with textbook noise sampling methods can lead to catastrophic differential privacy breaching. To guarantee differential privacy under floating-point implementations, [Mir12] proposes the snapping mechanism to avoid such security issues by rounding and smoothing the output of the Laplacian mechanism. [Tea20] introduces alternative approaches which can be applied to both Laplacian and Gaussian mechanisms and yield lower error than the snapping mechanism. [CKS20] provides algorithms to sample the discrete Gaussian noise for the query function  $f(D) \in \mathbb{Z}$ .

In this chapter, we first review above approaches, then integrate and transform them into our MPC-DP protocols.

### 3.1 Algorithms Overview

#### 3.1.1 Snapping Mechanism

[Mir12] demonstrates an attack, which can be executed with a few queries to the Laplacian mechanism and exploits the facts that the inversion sampling methods [Dev86] leads to a porous distribution over double-precision floating-point numbers. The snapping mechanism [Mir12] is proposed to prevent above attack and can guarantee differential privacy under floating-point implementations, and defined as

$$M_S(f(D), \lambda, B) = \text{clamp}_B(\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*) \rfloor_\Lambda),$$

where  $f(D) \in \mathbb{D}$  is a query function of database  $D$ . Function  $\text{clamp}_B(x)$  outputs  $B$  if  $x > B$ ,  $-B$  if  $x < -B$  and  $x$  otherwise.  $\oplus$  and  $\otimes$  are the floating-point implementations of addition and multiplication. Sign  $S$  is distributed uniformly over  $\{-1, 1\}$  and  $U^*$  is a uniform distribution over  $(0, 1)$ .  $\text{LN}(\cdot)$  is the natural logarithm implementation under floating-point arithmetic.  $\lambda$  is a parameter that controls the privacy strength. Function  $\lfloor x \rfloor_\Lambda$  rounds  $x$  to the nearest multiple of  $\Lambda$ . Note that the snapping mechanism  $M_S(f(D), \lambda, B)$  executes fully under floating-point implementations and can be proved to satisfy  $(\frac{1}{\lambda} + \frac{2^{-49}B}{\lambda})$ -DP for  $\lambda < B < 2^{46} \cdot \lambda$ .

#### 3.1.2 Secure Noise Generation

[Tea20] proposes alternative algorithms to implement both Laplacian and Gaussian mechanisms under floating-point arithmetic securely and yields lower error than the snapping mechanism. Since the floating-point sampler cannot approximate continuous real numbers with infinite precision, secure noise generation algorithms sample integers of certain probability distribution and scaled them to simulate continuous random variables. The privacy mechanism is defined as

$$M_{\text{SNG}}(f(D), r) = f_r(D) + ir,$$

where  $r$  is a resolution parameter that controls the scale of the simulated continuous distribution.  $i$  is an integer corresponding to certain probability distribution and scaled by  $r$ .  $f_r(D) \in \mathbb{D}$  is the result after rounding  $f(D) \in \mathbb{R}$  to the nearest multiple of  $r$ .

### 3.1.3 Discrete Gaussian Mechanism

[CKS20] introduces the algorithms to sample noise  $Y \sim \mathcal{N}_{\mathbb{Z}}(\mu, \sigma)$  from a discrete Gaussian distribution to achieve differential privacy for query function  $f(D) \in \mathbb{Z}$  as

$$M_{DGauss}(f(D)) = f(D) + Y.$$

Note that the discrete Gaussian mechanism is designed to provide differential privacy for query function  $f(D) \in \mathbb{Z}$  unlike the snapping mechanism and secure noise generation algorithms.

## 3.2 Notations

**TODO: introduce share conversion in preliminary part.**

For  $a, b, c \in \mathbb{N}$ , let  $[a]$  denotes  $\{x \in \mathbb{Z} \mid 1 \leq x \leq a\}$ ,  $[a \dots b)$  denotes  $\{x \in \mathbb{Z} \mid a \leq x < b\}$ ,  $(a, b)$  denotes  $\{x \in \mathbb{R} \mid a < x < b\}$ , and  $[a, b]$  denotes  $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ . Let  $\{a, b, c\}$  denotes a 3-element set that has elements  $a, b$  and  $c$ , and  $(a_i)_{i \in [n]} = (a_1, \dots, a_n)$  denotes a sequence consist of  $n$  elements.

Let  $\mathbb{D}$  denotes the numbers under floating-point implementations, and  $\mathbb{D} \cap (a, b)$  denotes the floating-point numbers in interval  $(a, b)$ .

The notation  $\bar{b}_{(l)}$  means repeating the bit  $b$  for  $l$  times. In the binary representation of floating-point numbers,  $a\bar{b}$  means repeating the bit  $b$  after  $a$  until the whole significant or exponent field are filled.  $(\cdot)_2$  is the binary representation of an integer.

The number of parties is denoted as  $N$  and parties as  $P_1, \dots, P_N$ . For value  $x$  that is shared between  $N$  parties, the shares is denoted as  $\langle x \rangle^S = (\langle x \rangle_1^S, \dots, \langle x \rangle_N^S)$  with subscript  $i \in [N]$  denoting the  $i$ -th share of  $x$  hold by party  $P_i$ , and the superscript  $S \in \{A, B, Y\}$  denoting the sharing types ( $A$ : arithmetic sharing,  $B$ : Boolean sharing,  $Y$ : Yao sharing).

The bold font  $\langle \mathbf{x} \rangle^{B,D}$  represents a vector of  $\ell$  Boolean sharing bits which is interpreted as an  $\ell$ -bit number of data type  $D$ . The second superscript  $D \in \{UI, FP, FL\}$  denoting its data types ( $UI$ : unsigned integer,  $FP$ : fixed-point number,  $FL$ : floating-point number). Note that the superscript data type  $\langle \cdot \rangle^D$  and subscript party index  $\langle \cdot \rangle_i$  are omitted when not affect the understanding of context.

For the logical operations of Boolean sharing bit  $\langle \cdot \rangle^B$ , we have XOR ( $\oplus$ ), AND ( $\wedge$ ) and NOT.

Let  $\langle \mathbf{a} \rangle^{B,D} \odot \langle \mathbf{b} \rangle^{B,D}$  denotes the arithmetic operations of Boolean shared bit vectors  $\langle \mathbf{a} \rangle^{B,D}$  and  $\langle \mathbf{b} \rangle^{B,D}$ , where  $\odot \in \{+, -, \times, \div, >, =\}$ .  $*$  and  $\cdot$  are used interchangeable as  $\times$ .

For other arithmetic operations ( $\ln(a)$ ,  $2^a$ ,  $e^a$ ,  $|a|$ ,  $\lfloor a \rfloor$ ,  $\lceil a \rceil$ ,  $a \bmod b$ ) of Boolean sharing bit vector  $\langle \mathbf{a} \rangle^{B,D}$ , they are denoted as  $\text{LN}(\langle \mathbf{a} \rangle^{B,D})$ ,  $\text{POW2}(\langle \mathbf{a} \rangle^{B,D})$ ,  $\text{EXP}(\langle \mathbf{a} \rangle^{B,D})$ ,  $\text{ABS}(\langle \mathbf{a} \rangle^{B,D})$ ,  $\text{Floor}(\langle \mathbf{a} \rangle^{B,D})$ ,  $\text{Ceil}(\langle \mathbf{a} \rangle^{B,D})$  and  $\text{MOD}(\langle \mathbf{a} \rangle^{B,D}, \langle \mathbf{b} \rangle^{B,D})$ .

The AND operation between a Boolean sharing bit  $\langle a \rangle^B$  and a vector of Boolean sharing bits  $\langle \mathbf{b} \rangle^{B,D}$  is denoted as  $\langle a \rangle^B \wedge \langle \mathbf{b} \rangle^{B,D}$ , which represents the bitwise  $\wedge$  operations between  $\langle a \rangle^B$  and every Boolean sharing bit  $\langle b \rangle^B \in \langle \mathbf{b} \rangle^{B,D}$ .

For data type conversions, we denote  $\langle \mathbf{a} \rangle^{B,FL} = \text{UI2FL}(\langle \mathbf{a} \rangle^{B,UI})$  as the conversion from a Boolean sharing bits vector representing unsigned integer to a floating-point number. Other data type conversion operations such as  $\langle \mathbf{a} \rangle^{B,UI} = \text{FL2UI}(\langle \mathbf{a} \rangle^{B,FL})$  are defined similarly.

### 3.3 MPC Techniques

We applied certain techniques to convert algorithms into computations of MPC.

#### 3.3.1 Branching

We convert **IF** branching in algorithm  $\text{Alg}^{Branching}$  **Algorithm 3.1** into MPC computations as

$$\text{output} = \langle a \rangle^B \wedge \langle b \rangle^B \oplus \text{NOT}(\langle a \rangle^B) \wedge \langle c \rangle^B.$$

<p><b>Protocol:</b> <math>\text{Alg}^{Branching}</math></p> <hr style="width: 50%; margin-left: 0;"/> <p style="text-align: center;">...</p> <p>1: <b>IF</b> <math>\langle a \rangle^B</math></p> <p>2:     <b>RETURN</b> <math>\langle b \rangle^B</math></p> <p>3: <b>ELSE</b></p> <p>4:     <b>RETURN</b> <math>\langle c \rangle^B</math></p> <p style="text-align: center;">...</p>
--

**Algorithm 3.1:** Algorithm example with **IF** branching.

#### 3.3.2 Loop

Suppose  $\text{Alg}^{Lottery}$  **Algorithm 3.2** is a lottery algorithm (charging  $b$  amount of money for each draw) that you can draw unlimited times, and the reward  $r$  decreases as the number of draws increases.  $\text{Alg}^{Toss}$  output either 1 (win) or 0 (lose) with the same probability  $p = 0.5$ .

**Algorithm:**  $Algo^{Lottery}$ 

```

...
1: trials  $\leftarrow 0$ 
2: WHILE TRUE
3:    $a \leftarrow Algo^{Toss}(b)$ 
4:   IF  $a$ 
5:     RETURN  $r - trials$ 
6:   trials  $\leftarrow trials + 1$ 
...

```

**Algorithm 3.2:** Algorithm example with **WHILE** loop.

To transform  $Algo^{Lottery}$  into MPC computations, the parties first assume that they will get the reward (i.e., **WHILE** loop terminates when  $\langle a_j \rangle^B == 1$  for the first time) after  $iter$  trials with very high probability. Then, the parties run  $\Pi^{Toss}$  for  $iter$  times to compute  $\langle a_j \rangle^B$  for  $j \in [0 \dots iter)$ . Since they don't know when  $\langle a_j \rangle^B == 1$  happens for the first time, they run  $\Pi^{PreOr}$  in line 4 to calculate  $e_j = \bigvee_{k=0}^j a_k$  such that  $e_0 = 0, \dots, e_{s-1} = 0$  and  $e_s = 1, \dots, e_{iter-1} = 1$ , where  $s$  is the smallest index in interval  $[0 \dots iter)$  making  $\langle a_j \rangle^B == 1$ . In line 6, they calculate and get  $p_s = 1, p_t = 0$  for  $t \in [0 \dots iter) \setminus s$ . Finally, the parties AND the reward of each iteration with  $p_j$  for  $j \in [0 \dots iter)$  and XOR these values to extract the correct reward  $x$ .

**Protocol:**  $\Pi^{Lottery}$ 

```

...
1: Each party locally set trial = 0.
2: FOR  $j = 0$  TO  $iter - 1$ 
3:   Parties run  $\langle a_j \rangle^B = \Pi^{Toss}(\langle b \rangle^{B, UI})$ .
4: Parties compute  $(\langle e_0 \rangle^B, \dots, \langle e_{iter-1} \rangle^B) = \Pi^{PreOr}(\langle a_0 \rangle^B, \dots, \langle a_{iter-1} \rangle^B)$ .
5: FOR  $j = 0$  TO  $iter - 1$ 
6:   Each party locally compute  $\langle p_j \rangle^B = \langle e_j \rangle^B \oplus \langle e_{j-1} \rangle^B$ , where  $\langle p_0 \rangle^B = \langle e_0 \rangle^B$ .
7: Parties compute  $\langle x \rangle^{B, UI} = \bigoplus_{j=0}^{iter-1} \langle p_j \rangle^B \wedge (\langle r \rangle^{B, UI} - j)$ .
...

```

**Protocol 3.1:** MPC protocol for algorithm with **WHILE** loop.

### 3.4 Snapping Mechanism

In this section, we first introduce the snapping mechanism [Mir12] and implementations under floating-point arithmetic, then provide the correspond MPC-DP protocol.

Before introducing the snapping mechanism, recall that the Laplacian mechanism under real number operations is defined as:

$$M(D) = f(D) + Y.$$

The Laplacian mechanism guarantees differential privacy of query function  $f(D) \in \mathbb{R}$  by adding it with noise  $Y \sim \text{Lap}(b)$ .  $Y$  can be generated with the inversion sampling method [Dev86] as following:

$$Y = S \cdot \lambda \ln(U),$$

where  $S \in \{-1, +1\}$  is the sign,  $\lambda$  controls the magnitude of  $Y$ , and  $U$  is a uniform random variable in interval  $(0, 1]$ . When the above Laplacian mechanism is implemented under floating-point arithmetic, the calculation of natural logarithm leads to a porous distribution of  $M(D)$ . As introduced in section **TODO: preliminary about floating-point security concerns**, the porous distribution and rounding effects of floating-point arithmetic can lead to privacy breaching of the Laplacian mechanism. To mitigate the attack, [Mir12] proposes the snapping mechanism to improve the Laplacian mechanism under floating-point arithmetic with rounding and clamping operations.

The snapping mechanism is defined as followings:

$$M_S(f(D), \lambda, B) = \text{clamp}_B(\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*) \rfloor_\Lambda).$$

$f(D) \in \mathbb{D}$  is the query function of database  $D$ . Function  $\text{clamp}_B(x)$  limits the output in interval  $[-B, B]$  and outputs  $B$  if  $x > B$ ,  $-B$  if  $x < -B$  and  $x$  otherwise.  $\oplus$  and  $\otimes$  are the floating-point implementations of addition and multiplication. Sign  $S$  denotes the sign of the noise and is distributed uniformly over  $\{-1, 1\}$ .  $U^*$  is a *uniform* distribution over  $\mathbb{D} \cap (0, 1)$  with probability proportional to its *unit in the last place* (ulp), i.e. spacing between two consecutive floating-point numbers.  $\text{LN}(\cdot)$  is the natural logarithm implementation under floating-point with exact rounding.  $\Lambda$  is the smallest power of two greater than or equal to  $\lambda$ , and we have  $\Lambda = 2^n$  such that  $2^{n-1} < \lambda \leq 2^n$  for  $n \in \mathbb{Z}$ .  $\lfloor \cdot \rfloor_\Lambda$  rounds its input to the nearest multiple of  $\Lambda$  exactly by manipulating the binary floating-point representation of the input. Note that the snapping mechanism assumes that the *sensitivity*  $\Delta_1^f$  of query function  $f$  is 1, which can be extended to arbitrary query function  $f'$  with *sensitivity*  $\Delta_1^{f'} \neq 1$  by scaling  $f'$  with  $f = \frac{f'}{\Delta_1^{f'}}$ .

**Theorem 5** ([Mir12]). *The snapping mechanism  $M_S(f(D), \lambda, B)$  satisfies  $\left(\frac{1}{\lambda} + \frac{2^{-49}B}{\lambda}\right)$ -DP for query function  $f$  with sensitivity  $\Delta_1^f = 1$  when  $\lambda < B < 2^{46} \cdot \lambda$ .*

TODO: prove about snapping mechanism DP properties and correctness

### 3.4.1 Implementations of Snapping Mechanism

Since [Mir12] only proposes the idea of snapping mechanism without implementation details, we review the snapping mechanism implementations from [Cov19] and adapt them into MPC-DP protocols in section § 3.4.2, other implementations of snapping mechanism are known [GS17; Cov21].

We introduce the implementations of the snapping mechanism consistently with the calculation order:

1. Calculation of  $\text{clamp}_B(\cdot)$ .
2. Generation of  $U^*$  and  $S$ .
3. Floating-point arithmetic operations:  $\text{LN}(\cdot)$ ,  $\oplus$ ,  $\otimes$ .
4. Calculation of  $\Lambda$ .
5. Calculation of  $\lfloor \cdot \rfloor_\Lambda$ .

Note that the floating-point arithmetic operations is available under floating-point implementations.

#### Calculation of $\text{clamp}_B(\cdot)$

Given input  $x$  and  $B > 0$ , the calculation of  $\text{clamp}_B(x)$  is defined as following:

$$\text{clamp}_B(x) = \begin{cases} B, & \text{if } x > B \\ -B, & \text{if } x < -B \\ x & \text{if } -B \leq x \leq B \end{cases} \quad (3.1)$$

### Generation of $U^*$ and $S$

Sign  $S \in \{-1, 1\}$  is a random variable that can be generated by tossing an unbiased coin, where *head* correspond to  $-1$  and *tail* to  $1$ .

$U^*$  is the *uniform* distribution over  $\mathbb{D} \cap (0, 1)$  and can be represented in the IEEE 754 double-precision binary floating-point binary64 [Com19] (64 bits) as:

$$U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023},$$

As the snapping mechanism [Mir12] required, each floating-point number sampled from  $U^*$  should be output with probability proportional to its ulp.

We sample a floating-point number from  $U^*$  using  $Alg^{RandFloat1}$  with the methods from [Wal74; Mir12], i.e., independently sampling a random variable  $x \sim Geo(p = 0.5)$  from a geometric distribution with  $Alg^{Geometric}$  Algorithm A.1 and setting  $U^*$ 's biased exponent  $e = 1023 - x$ , then sampling  $U^*$ 's significant bits  $(d_1, \dots, d_{52})$  uniformly from  $\{0, 1\}^{52}$ .

**Algorithm:**  $Alg^{RandFloat1}$

**Input:** None

**Output:**  $U^* \in \mathbb{D} \cap (0, 1)$

1 :  $(d_1, \dots, d_{52}) \leftarrow \{0, 1\}^{52}$

2 :  $x \leftarrow Alg^{Geometric}$

3 :  $e \leftarrow 1023 - x$

4 : **RETURN**  $U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$

**Algorithm 3.3:** Algorithm for random floating-point number from  $U^*$ .

Because  $U^*$ 's significant bits are sampled uniformly from  $\{0, 1\}^{52}$ , we have that the floating-point numbers (with the same biased exponent  $e - 1023$ ) are distributed uniformly in  $U^*$ . Further,  $x \sim Geo(p = 0.5)$  guarantees that the probability of sampling a floating-point number from  $U^*$  is proportional to its ulp.

Intuitively, *uniformly* sampling a floating point number can be thought as first drawing a real number in interval  $(0, 1)$  at random and rounding it to the nearest floating-point number. However, the floating-point numbers are discrete and not equidistant. For example, there are exactly  $2^{52}$  representable reals in interval  $[\cdot 5, 1)$  and  $2^{52}$  reals in interval  $[\cdot 25, \cdot 5)$ . If we only use the floating-point numbers with equal distance to each other in interval  $(0, 1)$ , then a large part of floating-point numbers would be ignored. The technique here is to sample a floating-point number with probability proportional to its ulp (spacing to its consecutive neighbor). With  $x \sim Geo(p = 0.5)$ , we assign a total probability  $p = 0.5$  for taking the floating-point numbers between  $(0, 1)$  with exponent  $e - 1023 = -1$  and a total probability  $p = 0.25$  for the floating-point numbers between  $[0, 0.5)$  with exponent  $e - 1023 = -2$  and



so forth. In this way, we can sample fine-grained floating-point numbers (with smaller biased exponent  $e - 1023$ ) in dense area and the total probability of floating-point numbers (with different biased exponents) in interval  $\mathbb{D} \cap (0, 1)$  being sampled is  $\sum_{i=1}^{\infty} \frac{1}{2^i} \approx 1$ .

**TODO:** image about uniform sampling of floating point number

### Calculation of $\Lambda$

The snapping mechanism uses input  $\lambda$  to calculate  $\Lambda$ , which is the smallest power of two greater than or equal to  $\lambda$ , i.e.,  $2^{n-1} < \lambda \leq 2^n$  for  $\Lambda = 2^n$  and  $n \in \mathbb{Z}$ .

We can represent  $\lambda$  in floating-point format as:

$$\lambda = (-1)^0 (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023}.$$

The calculation of  $\Lambda$  can be divided into two cases: (1)  $\lambda = 2^n$  for  $n \in \mathbb{Z}$ , which requires  $(d_i)_{i \in [52]} = 0$ ; (2)  $2^{n-1} < \lambda < 2^n$  for  $n \in \mathbb{Z}$ . For the first case, we can get  $\Lambda = \lambda$  since  $\lambda = 2^n$  is already a power of two. For the second case, we can calculate  $\Lambda$  by increasing the exponent of  $\lambda$  by 1 and setting all its significant bits  $(d_i)_{i \in [52]}$  to 0.

In summary we have

$$\Lambda = \begin{cases} \lambda, & \text{if for } i \in [52], \forall i : d_i = 0 \\ (1.\overline{0})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023 + 1}, & \text{if for } i \in [52], \exists i : d_i \neq 0 \end{cases} \quad (3.2)$$

### Calculation of $\lfloor \cdot \rfloor_{\Lambda}$

$\lfloor x \rfloor_{\Lambda}$  can be done in three steps:

1.  $x' = \frac{x}{\Lambda}$
2. Round  $x'$  to the nearest integer, yielding  $x''$
3.  $\lfloor x \rfloor_{\Lambda} = \Lambda \cdot x''$

**1.**  $x' = \frac{x}{\Lambda}$  We first represent  $x$  in floating-point format and get:

$$x = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023}.$$

Since  $\Lambda = 2^n$  is a power of two, the division can be performed by subtracting  $n$  from the exponent of  $x$  and we finally get:

$$x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023 - n}.$$

**2. Round  $x'$  to the nearest integer** Let  $y = (e_1 \dots e_{11})_2 - 1023 - n$  and we have

$$x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^y.$$

The rounding process of  $x'$  can be categorized into five cases depending on its unbiased exponent  $y$ .

**Case 1:**  $y \geq 52$

According to [Com19], when the exponent  $y$  is greater than or equal to 52,  $x'$  is an integer. Therefore, it is not necessary to round  $x'$  and we get  $x'' = x'$ , where

$$x'' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^y.$$

**Case 2:**  $y = 0$

If  $y = 0$ , we have

$$x' = (-1)^S (1.d_1 d_2 \dots d_{52})_2 \times 2^0.$$

The rounding result  $x''$  depends on  $d_1$  ( $x'' = (-1)^S \times 2^0$  if  $d_1 = 0$ ,  $x'' = (-1)^S \times 2^1$  if  $d_1 = 1$ ) and we have:

$$x'' = (-1)^S (1.\overline{0})_2 \times 2^{d_1}.$$

**Case 3:**  $y \in \{1, \dots, 51\}$

We represent  $x'$  by right-shifting the radix point  $y$  times (multiply by  $2^y$ ), omit the exponent term, and get:

$$x' = (-1)^S (1d_1 \dots d_y . d_{y+1} \dots d_{52})_2.$$

Note that  $(1d_1 \dots d_y)_2$  are the integer part and  $(.d_{y+1} \dots d_{52})_2$  is the fraction part. We have  $(.d_{y+1})_2 = 0.5$  when  $d_{y+1} = 1$ , and  $(.d_{y+1})_2 = 0$  when  $d_{y+1} = 0$ . Therefore, rounding  $x'$  to the nearest integer means carrying 1 to the integer part  $(1d_1 \dots d_y)_2$  if  $d_{y+1} = 1$ , or keeping the integer part unchanged if  $d_{y+1} = 0$ . In both cases, the bits in the fraction part are set to zeros. An edge case is when  $(d_i)_{i \in [y]} = 1$  and  $d_{y+1} = 1$ , then we get  $(d_i)_{i \in [y]} = 0$  after rounding the fraction part by incrementing the integer part up by 1. Therefore we can round  $x'$  by adding the exponent  $y$  with one and setting all the significant bits to zeros. Let  $(d'_1 \dots d'_y)_2 = (d_1 \dots d_y)_2 + 1$ .

In summary we have three subcases (case 3a, case 3b, case 3c):

$$x'' = \begin{cases} (-1)^S (1.d'_1 \dots d'_y \overline{0})_2 \times 2^y, & \text{if } d_{y+1} = 1 \text{ and for } i \in [y], \exists i : d_i = 0 \\ (-1)^S (1.\overline{0})_2 \times 2^{y+1}, & \text{if } d_{y+1} = 1 \text{ and for } i \in [y], \forall i : d_i = 1 \\ (-1)^S (1.d_1 \dots d_y \overline{0})_2 \times 2^y, & \text{if } d_{y+1} = 0 \end{cases} \quad (3.3)$$

**Case 4:**  $y = -1$

$x'$  can be represent without exponent term as:

$$x' = (-1)^S (0.1d_1d_2 \dots d_{51})_2.$$

Because the digit after radix is always 1,  $x''$  is always round to  $(-1)^S \times 2^0$ , which can be represented in IEEE-754 double-precision binary floating-point as:

$$x'' = (-1)^S (1.\bar{0})_2 * 2^{(01111111111)_2 - 1023}.$$

**Case 5:**  $y < -1$

$x'$  can be represent without exponent term as:

$$x' = (-1)^S (0.01d_1d_2 \dots d_{50})_2.$$

Because the digit after radix is always 0,  $x''$  is always round to 0. We set  $x'' = \pm 0$  which can be represented in IEEE-754 double-precision binary floating-point as

$$\begin{aligned} +0 &= (-1)^0 (1.\bar{0})_2 * 2^{(0000000000)_2 - 1023}, \\ -0 &= (-1)^1 (1.\bar{0})_2 * 2^{(0000000000)_2 - 1023}. \end{aligned}$$

**3. Multiply  $x''$  by  $\Lambda$**  In floating-point arithmetic, multiply  $x''$  by  $\Lambda = 2^n$  can be calculated by adding  $n$  to the exponent of  $x''$ . One exception is when  $x'' = \pm 0$  since  $x'' = (-1)^S (1.\bar{0})_2 \times 2^{(0000000000)_2 - 1023 + n}$  doesn't represent the values  $\pm 0$  in IEEE-754 double-precision binary floating-point.

### 3.4.2 MPC-DP Protocols

We describe the MPC-DP protocol for the snapping mechanism based on the implementations in section § 3.4.1 and general framework Prot. 2.1. The underlying algorithms and building blocks can be found in § A.1, § A.2.

Recall the snapping mechanism is define in § 3.4 as:

$$M_S(f(D), \lambda, B) = \text{clamp}_B([\text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)]_\Lambda).$$

In the MPC-DP settings, the snapping mechanism can be reformulated as:

$$\langle M_S(f(D), \lambda, B) \rangle = \text{clamp}_B([\text{clamp}_B(\langle f(D) \rangle) \oplus \langle S \rangle \otimes \lambda \otimes \text{LN}(\langle U^* \rangle)]_\Lambda). \quad (3.4)$$

In Eq. (3.4), we assume that each party has already computed  $\langle f(D) \rangle$  and follows the implementation steps § 3.4.1 to design the MPC-DP protocol for the snapping mechanism.  $\langle M_S(f(D), \lambda, B) \rangle$  is the share of the snapping mechanism's output and  $\langle S \rangle \otimes \lambda \otimes \text{LN}(\langle U^* \rangle)$  is the noise share.  $\lambda$  controls the magnitude of the generated noise and is publicly known. Therefore the calculation of  $\Lambda$  can be done locally by each party or calculated by one party and broadcast to others. The generation of  $U^*$  and  $S$ , calculation of  $[\cdot]_\Lambda$  and  $\text{clamp}_B(\cdot)$  need to be executed in MPC protocols.

### Calculation of $\text{clamp}_B(\cdot)$

As described in section § 3.4.1, function  $\text{clamp}_B(x)$  outputs  $B$  if  $x > B$ ,  $-B$  if  $x < -B$  and  $x$  otherwise.

$\Pi^{\text{Clamp}}(\langle x \rangle^{B,FL}, B)$  realizes function  $\text{clamp}_B(x)$  by computing the correct output with conditions as:

$$\begin{aligned} \langle x_{\text{clamp}_B} \rangle^{B,FL} = & \langle \text{cond}_{x < -B} \rangle^B \wedge -B \\ & \oplus \langle \text{cond}_{x > B} \rangle^B \wedge B \\ & \oplus \langle x \rangle^{B,FL} \wedge \text{NOT}(\langle \text{cond}_{x < -B} \rangle^B \vee \langle \text{cond}_{x > B} \rangle^B) \end{aligned} \quad (3.5)$$

**Protocol:**  $\Pi^{\text{Clamp}}(\langle x \rangle^{B,FL}, B)$

**Input:**  $\langle x \rangle^{B,FL}, B$

**Output:**  $\langle x_{\text{clamp}_B} \rangle^{B,FL}$

- 1: Parties compute  $\langle \text{cond}_{x < -B} \rangle^B = (\langle x \rangle^{B,FL} < -B)$ .
- 2: Parties compute  $\langle \text{cond}_{x > B} \rangle^B = (\langle x \rangle^{B,FL} > B)$ .
- 3: Parties compute  $\langle x_{\text{clamp}_B} \rangle^{B,FL}$ .

**Protocol 3.2:** MPC protocol for  $\text{clamp}_B(\cdot)$ .

### Generation of $U^*$ and $S$

Each party can generate  $\langle S \rangle^B$  by running  $\Pi^{\text{RandBits}}(1)$  locally.

As discussed in § 3.4.1,  $U^*$  is the uniform distribution over  $\mathbb{D} \cap (0, 1)$ , which can be represented in IEEE-754 double-precision binary floating-point as:

$$U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}.$$

As  $\Pi^{\text{RandFloat1}}$  shows, we first generate the uniform significand bits share  $\langle d_1 \rangle^B, \dots, \langle d_{52} \rangle^B$  with  $\Pi^{\text{RandBits}}(52)$ , then generate a geometric random variable share  $\langle x \rangle^{B,UI}$  with  $\Pi^{\text{Geometric}}$  to build the biased exponent  $\langle e \rangle^{B,UI}$ , where  $e = 1023 - x$ . Finally, the parties compute and obtain shares of the floating-point number  $U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$ . Note that the geometric random variable  $x > 0$  and  $\langle x \rangle^{B,UI}$  represents an unsigned integer.

**Protocol:**  $\Pi^{RandFloat1}$ 
**Input:** None

**Output:**  $\langle U^* \rangle^{B,FL}$ , where  $U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$ 

 1 : Parties run  $(\langle d_1 \rangle^B, \dots, \langle d_{52} \rangle^B) = \Pi^{RandBits}(52)$ .

 2 : Parties run  $\langle x \rangle^{B,UI} = \Pi^{Geometric}$ .

 3 : Parties compute  $\langle e \rangle^{B,UI} = 1023 - \langle x \rangle^{B,UI}$ 
**Protocol 3.3:** MPC protocol for uniform random variable  $U^* \in \mathbb{D} \cap (0, 1)$ .

**Calculation of  $\lfloor \cdot \rfloor_\Lambda$** 

 As discussed in § 3.4.1,  $\lfloor x \rfloor_\Lambda$  can be calculated in three steps:

1.  $x' = \frac{x}{\Lambda}$
2. Round  $x'$  to the nearest integer, yielding  $x''$
3.  $\lfloor x \rfloor_\Lambda = \Lambda \cdot x''$

**1.  $x' = \frac{x}{\Lambda}$**  Let  $x = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023}$ . For  $\Lambda = 2^n$ , we calculate  $x' = \frac{x}{\Lambda}$  by subtracting  $n$  from the exponent of  $x$  and get

$$x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(z_1 \dots z_{11})_2 - 1023},$$

 where  $(z_1 \dots z_{11})_2 = (e_1 \dots e_{11})_2 - n$ .

**Protocol:**  $\Pi^{XDivideLambda}(\langle e \rangle^{B,UI}, n)$ 
**Input:**  $\langle e \rangle^{B,UI}, n$ 
**Output:**  $\langle z \rangle^{B,UI}$ , where  $z = e - n$ 

 1 : Parties compute  $\langle z \rangle^{B,UI} = \langle e \rangle^{B,UI} - n$ 
**Protocol 3.4:** MPC protocol for  $x' = \frac{x}{\Lambda}$ .

**2. Round  $x'$  to the nearest integer** Suppose we have:

$$x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(y_1 \dots y_{11})_2}.$$

Let  $(z_1 \dots z_{11})_2 - 1023 = (y_1 \dots y_{11})_2$ . Then, we represent  $x'$  in IEEE-754 double-precision binary floating-point as:

$$x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(z_1 \dots z_{11})_2 - 1023}.$$

We construct a MPC protocol  $\Pi^{\text{RoundDouble2Int}}(\langle \mathbf{d} \rangle^{B,UI}, \langle \mathbf{z} \rangle^{B,UI})$  for rounding  $x'$  to the nearest integer based on the value of  $y = (y_1 \dots y_{11})_2$  as [paragraph 3.4.1](#) discussed. The parties first calculate the conditions and rounding result for five cases. Then, we use the MPC techniques [§ 3.3](#) to output the correct rounding result.

**Case 1:**  $y \geq 52$ , line 2

In line 2, the parties compute the condition for **Case 1** by checking if  $y \geq 52$

**Case 2:**  $y = 0$ , line 3 – 5

In line 3, the parties compute the condition for **Case 2** by checking if  $y == 0$ . In line 4, the parties calculate the biased exponent bits  $z_{d_1+1023}$ . In line 5, each party locally set the significant bits.

**Case 3:**  $y \in \{1, \dots, 51\}$ , line 6 – 17

In line 6, the parties calculate the condition for **Case 3** by checking if  $1 \leq y \leq 51$ . In line 7, the parties run  $\Pi^{\text{Binary2Unary}}(\langle y \rangle^{B,UI}, 52)$  to get  $p_1, \dots, p_{52}$ , where  $p_j = 1$  for  $j \in [y]$  and other bits equal to zero. In line 8, the parties extract the first  $y$  bits from the significant field and set the rest bits to zero. In line 9, 10, each party locally compute  $c_1, \dots, c_{52}$  such that only  $c_{y+1} = 1$  and other bits equal to zero. In line 11, the parties extract the value of  $d_{y+1}$  from the significant field. In line 12, the parties count number of bits in  $d_1, \dots, d_y$  that equal to one. In line 13, the parties compute  $(d'_1 \dots d'_y)_2 = (d_1 \dots d_y)_2 + 1$  by transforming it into arithmetic operations of unsigned integer and fill the rest  $52 - y$  bits of the significant field with zeros. In line 14, 15, 17, the parties compute the subconditions of **Case 3**. In line 16, the parties compute the biased exponent bits  $z_{y+1+1023}$  for case 3b.

**Case 4:**  $y = -1$ , line 18, 19

In line 18, the parties compute the condition for **Case 4** by checking if  $y == -1$ . In line 19, each party locally set the biased exponent bits  $z_{0, \bar{1}_{(10)}}$ .

**Case 5:**  $y < -1$ , line 20, 21

In line 20, the parties compute the condition for **Case 5** by checking if  $y < -1$ . In line 21, each party locally set the biased exponent bits  $z_{\bar{0}_{(11)}}$ .

Finally, the parties compute the significant bits  $\langle \mathbf{d}_{\text{round}} \rangle^{B,UI}$  and biased exponent  $\langle \mathbf{z}_{\text{round}} \rangle^{B,UI}$  with

$$\begin{aligned}
 \langle \mathbf{d}_{\text{round}} \rangle^{B,UI} = & \langle \text{cond}_{\text{case1}} \rangle^B \wedge \langle \mathbf{d} \rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case3a}} \rangle^B \wedge \left\langle \mathbf{d}_{1', \dots, y', \bar{0}_{(52-y)}} \right\rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case3c}} \rangle^B \wedge \left\langle \mathbf{d}_{1, \dots, y, \bar{0}_{(52-y)}} \right\rangle^{B,UI} \\
 & \oplus \left( \langle \text{cond}_{\text{case2}} \rangle^B \oplus \langle \text{cond}_{\text{case3b}} \rangle^B \oplus \langle \text{cond}_{\text{case4}} \rangle^B \oplus \langle \text{cond}_{\text{case5}} \rangle^B \right) \wedge \left\langle \mathbf{d}_{\bar{0}_{(52)}} \right\rangle^{B,UI}
 \end{aligned} \tag{3.6}$$

$$\begin{aligned}
 \langle \mathbf{z}_{\mathbf{round}} \rangle^{B,UI} = & \left( \langle \text{cond}_{\text{case1}} \rangle^B \oplus \langle \text{cond}_{\text{case3a}} \rangle^B \oplus \langle \text{cond}_{\text{case3c}} \rangle^B \right) \wedge \langle \mathbf{z} \rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case2}} \rangle^B \wedge \langle \mathbf{z}_{d_1+1023} \rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case3b}} \rangle^B \wedge \langle \mathbf{z}_{y+1+1023} \rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case4}} \rangle^B \wedge \langle \mathbf{z}_{\mathbf{0}, \bar{1}_{(10)}} \rangle^{B,UI} \\
 & \oplus \langle \text{cond}_{\text{case5}} \rangle^B \wedge \langle \mathbf{z}_{\bar{\mathbf{0}}_{(11)}} \rangle^{B,UI}
 \end{aligned} \tag{3.7}$$

**Protocol:**  $\Pi^{\text{RoundDouble2Int}}(\langle \mathbf{d} \rangle^{B,UI}, \langle \mathbf{z} \rangle^{B,UI})$

**Input:**  $\langle \mathbf{d} \rangle^{B,UI} = (\langle d_1 \rangle^B, \dots, \langle d_{52} \rangle^B)$ ,  $\langle \mathbf{z} \rangle^{B,UI} = (\langle z_1 \rangle^B, \dots, \langle z_{11} \rangle^B)$

**Output:**  $\langle \mathbf{d}_{\text{round}} \rangle^{B,UI}, \langle \mathbf{z}_{\text{round}} \rangle^{B,UI}$

- 1: Parties compute  $\langle \mathbf{y} \rangle^{B,UI} = \langle \mathbf{z} \rangle^{B,UI} - 1023$ .
- 2: Parties compute  $\langle \text{cond}_{\text{case1}} \rangle^B = (\langle \mathbf{y} \rangle^{B,UI} \geq 52)$ .
- 3: Parties compute  $\langle \text{cond}_{\text{case2}} \rangle^B = (\langle \mathbf{y} \rangle^{B,UI} == 0)$ .
- 4: Parties compute  $\langle \mathbf{z}_{d_1+1023} \rangle^{B,UI} = \langle d_1 \rangle^B \wedge 1 + 1023$ .
- 5: Each party locally set  $\langle \mathbf{d}_{\bar{0}_{(52)}} \rangle^{B,UI} = (\bar{0}_{(52)})$ .
- 6: Parties compute  $\langle \text{cond}_{\text{case3}} \rangle^B = (\langle \mathbf{y} \rangle^{B,UI} \geq 1) \wedge (\langle \mathbf{y} \rangle^{B,UI} \leq 51)$ .
- 7: Parties run  $(\langle p_1 \rangle^B, \dots, \langle p_{52} \rangle^B) = \Pi^{\text{Binary2Unary}}(\langle \mathbf{y} \rangle^{B,UI}, 52)$ .
- 8: Parties compute  $\langle \mathbf{d}_{1,\dots,y,\bar{0}_{(52-y)}} \rangle^{B,UI} = (\langle p_1 \rangle^B \wedge \langle d_1 \rangle^B, \dots, \langle p_{52} \rangle^B \wedge \langle d_{52} \rangle^B)$ .
- 9: **FOR**  $j = 1$  **TO** 51
  - 10: Each party locally compute  $\langle c_{j+1} \rangle^B = \langle p_j \rangle^B \oplus \langle p_{j+1} \rangle^B$ , where  $\langle c_1 \rangle^B = 0$ .
  - 11: Parties compute  $\langle d_{y+1} \rangle^B = \oplus_{j=1}^{52} \langle c_j \rangle^B \wedge \langle d_j \rangle^B$ .
  - 12: Parties compute  $\langle \mathbf{d}_{\text{sum}(1,\dots,y)} \rangle^{B,UI} = \sum_{j=1}^{52} \text{B2A}(\langle p_j \rangle^B \wedge \langle y_j \rangle^B)$ .
  - 13: Parties compute  $\langle \mathbf{d}_{1',\dots,y',\bar{0}_{(52-y)}} \rangle^{B,UI} = \langle \mathbf{d}_{1,\dots,y,\bar{0}_{(52-y)}} \rangle^{B,UI} + \text{POW2}(52 - \langle \mathbf{y} \rangle^{B,UI})$ .
  - 14: Parties compute  $\langle \text{cond}_{\text{case3a}} \rangle^B = (\langle \mathbf{d}_{\text{sum}(1,\dots,y)} \rangle^{B,UI} < \langle \mathbf{y} \rangle^{B,UI})$ .
  - 15: Parties compute  $\langle \text{cond}_{\text{case3b}} \rangle^B = (\langle \mathbf{d}_{\text{sum}(1,\dots,y)} \rangle^{B,UI} == \langle \mathbf{y} \rangle^{B,UI})$ .
  - 16: Parties compute  $\langle \mathbf{z}_{y+1+1023} \rangle^{B,UI} = \langle \mathbf{z} \rangle^{B,UI} + 1 + 1023$ .
  - 17: Parties compute  $\langle \text{cond}_{\text{case3c}} \rangle^B = \text{NOT}(\langle d_{y+1} \rangle^B)$ .
  - 18: Parties compute  $\langle \text{cond}_{\text{case4}} \rangle^B = (\langle \mathbf{y} \rangle^{B,UI} == (-1))$ .
  - 19: Each party locally set  $\langle \mathbf{z}_{0,\bar{1}_{(10)}} \rangle^{B,UI} = (0, \bar{1}_{(10)})$ .
  - 20: Parties compute  $\langle \text{cond}_{\text{case5}} \rangle^B = (\langle \mathbf{y} \rangle^{B,UI} < (-1))$ .
  - 21: Each party locally set  $\langle \mathbf{z}_{\bar{0}_{(11)}} \rangle^{B,UI} = (\bar{0}_{(11)})$ .
  - 22: Parties compute  $\langle \mathbf{d}_{\text{round}} \rangle^{B,UI}, \langle \mathbf{z}_{\text{round}} \rangle^{B,UI}$ .

**Protocol 3.5:** Protocol for rounding  $x'$  to the nearest integer.

**3. Multiply  $x''$  by  $\Lambda$**  Suppose we have:

$$x'' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(z_1 \dots z_{11})_2 - 1023}.$$



$\Pi^{MulDoubleNPow2}(\langle \mathbf{d} \rangle^{B,UI}, \langle \mathbf{z} \rangle^{B,UI}, n)$  calculate  $x'' \cdot \Lambda$  using similar way as [Prot. 3.5](#) to transform the branching (case 1:  $x'' = \pm 0$ ; case 2:  $x \neq \pm 0$ ) into MPC computations as line 1, 2 shows. In line 2, the parties add the biased exponent of  $x'' \cdot \Lambda$  with  $n$  if  $x'' \neq \pm 0$ , keep the biased exponent of  $x'' \cdot \Lambda$  same as  $x''$  otherwise.

<b>Protocol:</b> $\Pi^{MulDoubleNPow2}(\langle \mathbf{d} \rangle^{B,UI}, \langle \mathbf{z} \rangle^{B,UI}, n)$	
<b>Input:</b>	$\langle \mathbf{d} \rangle^{B,UI} = (\langle d_1 \rangle^B, \dots, \langle d_{52} \rangle^B), \langle \mathbf{z} \rangle^{B,UI} = (\langle z_1 \rangle^B, \dots, \langle z_{11} \rangle^B), n$
<b>Output:</b>	$\langle \mathbf{z}_{add(n)} \rangle^B$
1 :	Parties compute $\langle cond_{\neq \pm 0} \rangle^B = (\vee_{j=1}^{52} \langle d_j \rangle^B) \vee (\vee_{j=1}^{11} \langle z_j \rangle^B)$ .
2 :	Parties compute $\langle \mathbf{z}_{add(n)} \rangle^{B,UI} = \langle cond_{\neq \pm 0} \rangle^B \wedge (\langle \mathbf{z} \rangle^{B,UI} + n)$ .

**Protocol 3.6:** MPC protocol for multiplying  $x''$  by  $\Lambda$ .

### MPC-DP Protocol for Snapping Mechanism

We present the MPC-DP protocol  $\Pi^{SnappingMechanism}(\langle f(D) \rangle, \lambda, B, n)$  for the snapping mechanism in [Eq. \(3.4\)](#):

$$\langle M_S(f(D), \lambda, B) \rangle = \text{clamp}_B(\lfloor \text{clamp}_B(\langle f(D) \rangle) \oplus \langle S \rangle \otimes \lambda \otimes \text{LN}(\langle U^* \rangle) \rfloor_\Lambda). \quad (3.8)$$

In line 5, we set the sign of  $\langle Y_{LapNoise} \rangle^{B,FL}$  by directly multiplying its floating-point format sign bit share  $\langle S_{Y_{LapNoise}} \rangle^B$  with  $\langle S \rangle^B$ , rather than converting  $\langle S \rangle^B$  into a floating-point number  $\langle S \rangle^{B,FL}$  (that equals to  $\pm 1$ ) and multiplying it with  $Y_{LapNoise}$ .

Note:

1.  $\Lambda = 2^n$  and can be calculate from the public known  $\lambda$  without MPC protocols as [section § 3.4.1](#).
2. Output  $\langle \mathbf{x}_{SM} \rangle^{B,FL} = \text{clamp}_B(\lfloor \text{clamp}_B(\langle f(D) \rangle) \oplus \langle S \rangle \otimes \lambda \otimes \text{LN}(\langle U^* \rangle) \rfloor_\Lambda)$ .
3.  $\langle \mathbf{d} \rangle^{B,UI}$  and  $\langle \mathbf{z} \rangle^{B,UI}$  are the significand and biased exponent field bits of  $\langle \mathbf{x} \rangle^{B,FL}$ .
4.  $\langle \mathbf{d}' \rangle^{B,UI}$  and  $\langle \mathbf{z}' \rangle^{B,UI}$  are the significand and biased exponent field bits of  $\langle \mathbf{x}' \rangle^{B,FL}$ .
5.  $\langle \mathbf{d}'' \rangle^{B,UI}$  and  $\langle \mathbf{z}'' \rangle^{B,UI}$  are the significand and biased exponent field bits of  $\langle \mathbf{x}'' \rangle^{B,FL}$ .

**Protocol:**  $\Pi^{\text{SnappingMechanism}}(\langle f(D) \rangle, \lambda, B, n)$

**Input:**  $\langle f(D) \rangle^{B,FL}, \lambda, B, n$

**Output:**  $\langle x_{SM} \rangle^{B,FL}$

- 1: Parties run  $\langle U^* \rangle^{B,FL} = \Pi^{\text{RandFloat1}}$ .
- 2: Each party locally run  $\langle S \rangle^B = \Pi^{\text{RandBits}}(1)$ .
- 3: Parties run  $\langle f(D)_{\text{clamp}B} \rangle^{B,FL} = \Pi^{\text{Clamp}}(\langle f(D) \rangle^{B,FL}, B)$ .
- 4: Parties compute  $\langle Y_{\text{LapNoise}} \rangle^{B,FL} = \lambda * \text{LN}(\langle U^* \rangle^{B,FL})$ .
- 5: Parties compute  $\langle S'_{Y_{\text{LapNoise}}} \rangle^B = \langle S_{Y_{\text{LapNoise}}} \rangle^B \wedge \langle S \rangle^B$ , where  $\langle S_{Y_{\text{LapNoise}}} \rangle^B$  is the sign bit of  $\langle Y_{\text{LapNoise}} \rangle^{B,FL}$ .
- 6: Each party locally set  $\langle S'_{Y_{\text{LapNoise}}} \rangle^B$  as the sign bit of  $\langle Y_{\text{LapNoise}} \rangle^{B,FL}$ .
- 7: Parties compute  $\langle x \rangle^{B,FL} = \langle f(D)_{\text{clamp}B} \rangle^{B,FL} + \langle Y_{\text{LapNoise}} \rangle^{B,FL}$ .
- 8: Parties run  $\langle x' \rangle^{B,FL} = \Pi^{\text{XDivideLambda}}(\langle x \rangle^{B,FL}, n)$ .
- 9: Parties run  $\langle x'' \rangle^{B,FL} = \Pi^{\text{RoundDouble2Int}}(\langle d' \rangle^{B,UI}, \langle z' \rangle^{B,UI})$ .
- 10: Parties run  $\langle x''' \rangle^{B,FL} = \Pi^{\text{MulDoubleNPow2}}(\langle d'' \rangle^{B,UI}, \langle z'' \rangle^{B,UI}, n)$ .
- 11: Parties run  $\langle x_{SM} \rangle^{B,FL} = \Pi^{\text{Clamp}}(\langle x''' \rangle^{B,FL}, B)$  and output  $\langle x_{SM} \rangle^{B,FL}$ .

**Protocol 3.7:** MPC-DP protocol for snapping mechanism.

### 3.5 Secure Noise Generation

In this section, we first introduce the secure noise generation methods from [Tea20], then provide the MPC implementations.

The general secure noise generation framework is as follows:

1. Set a resolution parameter  $r = 2^k$ , where  $k \in [-1022 \dots 1023]$ .
2. Sample  $i$  from  $\text{Sampler}(\epsilon, \delta, r, \Delta_r)$ , which guarantees that  $\Pr[|i|] < \frac{1}{e^{(1000)}}$ .
3. Privacy mechanism is defined as  $M(D) = f_r(D) + ir$ , where  $ir$  is the noise term.

The basic idea is to operate on a discrete instead of a continuous number field. To achieve this, a resolution parameter  $r$  is chosen which controls the scale of the simulated continuous distribution.  $i$  is an integer correspond to certain distribution and generated by  $\text{Sampler}(\epsilon, \delta, r, \Delta_r)$ , where  $\epsilon$  and  $\delta$  are the parameters that controls the privacy protection levels. By limiting  $\mathbb{P}[|i| > 2^{52}] < \frac{1}{e^{(1000)}}$ , we guaranteed that under double precision floating-point implementations,  $i$  can be represented exactly as an integer and  $ir$  exactly as a floating-point number with very high probability. For the cases that  $\Pr[|i| > 2^{52}] > \frac{1}{e^{(1000)}}$ , they can either be ignored or counted into the  $\delta$  term (probability that privacy leakage happens) of mechanism  $M(D)$ .  $f_r(D)$  is obtained by rounding  $f(D)$  to the nearest multiple

of  $r$  and can be represented exactly as floating-point. Similar to the sensitivity definition of  $f(D)$ , the sensitivity  $\Delta_r$  of  $f_r(D)$  is defined as  $\Delta_r = \max_{D, D'} \|f_r(D) - f_r(D')\|$ .

### 3.5.1 Laplacian Mechanism

Instead of sampling the noise from a Laplacian distribution  $Lap(b)$  (PDF:  $Pr(x|b) = \frac{1}{2b}e^{-\frac{|x|}{b}}$  for  $x \in \mathbb{R}$  and  $b = \frac{\Delta}{\epsilon}$ ), we sample an integer  $i$  from a double-side geometric distribution  $DGeo(p)$  (PMF:  $Pr(x|p) = (1-p)^{|x|}p$  for  $x \in \mathbb{N}$  and  $p = 1 - e^{-\frac{r\epsilon}{\Delta_r}}$ ).  $r$  is the smallest power of 2 exceeding  $\frac{\Delta}{2^k\epsilon}$ , where  $k$  is a predefined integer that controls the degree of discretization and accuracy of  $M(D)$ .

**TODO: DP prove and properties.**

We first describe the algorithm for sampling the non-negative geometric random variable. The algorithm  $Alg_{GeometricExpBinarySearch}$  [Tea20] samples a positive integer  $x$  from geometric distribution with  $p = 1 - e^{-\lambda}$ , and we have

$$\begin{aligned} \text{PMF} : Pr(x | p = 1 - e^{-\lambda}) &= (1-p)^{x-1} p \\ \text{CDF} : 1 - (1-p)^x & \end{aligned}$$

The sampling interval is  $(0 \dots Int_{max64}]$ , where  $Int_{max64} = (0\bar{1}_{(63)})_2 = 9223372036854775807$  is the maximal signed 64-bit integer. For the cumulative distribution function, when  $x = Int_{max64}$  we have

$$\text{CDF} : 1 - e^{-\lambda Int_{max64}} = \begin{cases} 0.\bar{9}_{(27)}8396\dots & \text{for } \lambda = 2^{-57} \\ 0.\bar{9}_{(13)}8733\dots & \text{for } \lambda = 2^{-58} \\ 0.\bar{9}_{(6)}8874\dots & \text{for } \lambda = 2^{-59} \end{cases} \quad (3.9)$$

The support of geometric distribution is the set that contains all the positive integers. Since we can only sample integers in interval  $(0 \dots Int_{max64}]$ , we take  $\lambda \geq 2^{-58}$  such that the difference in total probability caused by  $Int_{max64}$  and  $+\infty$  can be ignored.

Function  $\text{Split}(\cdot)$  calculate the middle point  $M$  of interval  $(L \dots R]$  such that

$$Pr(L < x \leq M | L < x \leq R) = \frac{Pr(L < x \leq M)}{Pr(L < x \leq R)} = \frac{1}{2},$$

where

$$\begin{aligned}
 \Pr(L < x \leq M) &= \Pr(x \leq M) - \Pr(x \leq L) \\
 &= (1 - (1 - p)^M) - (1 - (1 - p)^L) \\
 &= (1 - e^{-\lambda M}) - (1 - e^{-\lambda L}) \\
 &= e^{-\lambda L} - e^{-\lambda M} \\
 &= \frac{1}{e^{\lambda L}} - \frac{1}{e^{\lambda M}} \\
 &= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda L + \lambda M}}
 \end{aligned} \tag{3.10}$$

We can also get  $\Pr(L < x \leq R) = \frac{e^{\lambda R} - e^{\lambda L}}{e^{\lambda L + \lambda R}}$  similarly. Finally, we have  $M = L - \frac{\text{LN}(0.5) + \text{LN}(1 + e^{-\lambda(R-L)})}{\lambda}$ . Note that the calculation of  $M$  should take into account about the underflow and overflow of  $\text{LN}(\cdot)$  and  $\text{Exp}(\cdot)$  under floating-point implementation.

**TODO: discuss about LN and Exp in double precision floating-point number**

Function  $\text{Proportion}(\cdot)$  calculates the proportion  $Q$  between  $\Pr(L < x \leq M)$  and  $\Pr(L < x \leq R)$ , which should be approximately 0.5, where

$$\begin{aligned}
 Q &= \Pr(L < x \leq M \mid L < x \leq R) \\
 &= \frac{\Pr(L < x \leq M)}{\Pr(L < x \leq R)} \\
 &= \frac{e^{-\lambda(M-L)} - 1}{e^{-\lambda(R-L)} - 1}
 \end{aligned} \tag{3.11}$$

Generally, in  $\text{Algo}^{\text{GeometricExpBinarySearch}}$ , we sample an integer  $x$  by splitting the sampling interval into two subintervals  $((L \dots M]$  and  $(M \dots R])$  that have approximately equal probability mass with middle point  $M$  as line 1 – 3 shows. Line 4 – 7 ensures the the middle point  $M$  lies between  $L$  and  $R$ . Then we calculate the probability mass proportion between those two subintervals and randomly choose one interval based on the generated uniform variable  $U^*$ . Next, we repeat this process until the interval only contains one number which is the integer  $x$  we are looking for.

Note that one implicit assumption of  $\text{Algo}^{\text{GeometricExpBinarySearch}}$  is that the uniform random variable  $U^* \leq 1 - e^{-\lambda \text{Int}_{\max 64}}$  or  $1 - e^{-\lambda \text{Int}_{\max 64}} \approx 1$ . For example, suppose  $\lambda$  is a very small value such than  $\lambda * \text{Int}_{\max 64} = 4$ , then we have CDF:  $1 - e^{-4} = 0.9816\dots$  which means the total probability of the sampling interval is  $0.9816\dots$  and certain samples (with probability  $1 - 0.9816\dots$ ) can't be generated by the algorithm.

**Algorithm:**  $Algo^{GeometricExpBinarySearch}$ 
**Input:** None

**Output:**  $x \sim Geo(p = 1 - e^{-\lambda})$ 

```

1:  $L \leftarrow 0, R \leftarrow Int_{max64}$ 
2: FOR  $L + 1 < R$ 
3:    $M \leftarrow Split(L, R, \lambda)$ 
4:   IF  $M \leq L$ 
5:      $M = L + 1$ 
6:   ELSE IF  $M \geq R$ 
7:      $M = R - 1$ 
8:    $Q = Proportion(L, R, M, \lambda)$ 
9:    $U^* \leftarrow Algo^{RandFloat1}$ 
10:  IF  $U^* \leq Q$ 
11:     $R \leftarrow M$ 
12:  ELSE
13:     $L \leftarrow M$ 
14: RETURN  $x \leftarrow R$ 

```

**Algorithm 3.4:** Algorithm for geometric distribution with  $p = 1 - e^{-\lambda}$ .

To convert  $Algo^{GeometricExpBinarySearch}$  into MPC protocol, we use similar techniques as ?? to transform the branching into MPC calculations. In the FOR loop, we assume that  $Algo^{GeometricExpBinarySearch}$  executes less than  $iter$  times and return the results. In line 1 – 4, each party locally do the calculation. After line 5, the parties execute the MPC computations and finish the first iteration in line 10. In line 6 – 9, the parties transform the branching into MPC computations. In line 10, the parties set the *flag* to record if the FOR loop condition is satisfied. In line 21, 22, the parties calculate the position  $p_0, \dots, p_{iter-1}$  where  $p_j = 1$  and  $j$  is the smallest number such that  $flag_j = 1$ , i.e., the FOR loop condition is not satisfied and algorithms return a result  $x = R_j$ .

**Protocol:**  $\Pi^{\text{GeometricExpBinarySearch}}$

**Input:** None

**Output:**  $\langle x \rangle^A$ , where  $x \sim \text{Geo}(p = 1 - e^{-\lambda})$

- 1 : Each party locally set  $L_0 = 0, R_0 = \text{Int}_{\max 64}$ .
- 2 : Each party locally calculate  $M_0 = \text{Split}(L_0, R_0, \lambda)$
- 3 : Each party locally set  $M_0 = L_0 + 1$  if  $M_0 \leq L_0$ , and set  $M_0 = R_0 - 1$  if  $M_0 > L_0 \wedge M_0 \geq R_0$ .
- 4 : Each party locally calculate  $Q_0 = \text{Proportion}(L_0, R_0, M_0, \lambda)$
- 5 : Parties execute  $\Pi^{\text{RandFloat1}}$  and obtain  $\langle U_0^* \rangle^B$
- 6 : Parties compute  $\langle \text{cond}_{U_0 \leq Q_0} \rangle^A = \text{B2A}(\langle U_0^* \rangle^B \leq Q_0)$
- 7 : Parties compute  $\langle \text{cond}_{U_0 > Q_0} \rangle^A = \text{B2A}(\langle U_0^* \rangle^B > Q_0)$
- 8 : Parties compute  $\langle R_0 \rangle^A = \langle \text{cond}_{U_0 \leq Q_0} \rangle^A * M + \langle \text{cond}_{U_0 > Q_0} \rangle^A * \langle R_0 \rangle^A$ .
- 9 : Parties compute  $\langle L_0 \rangle^A = \langle \text{cond}_{U_0 > Q_0} \rangle^A * M + \langle \text{cond}_{U_0 \leq Q_0} \rangle^A * \langle L_0 \rangle^A$ .
- 10 : Parties compute  $\langle \text{flag}_0 \rangle^B = \text{A2B}(\langle L_0 \rangle^A + 1 \geq \langle R_0 \rangle^A)$
- 11 : For  $j \in [1 \dots \text{iter}]$ 
  - 12 : Parties compute  $\langle M_j \rangle^A = \Pi^{\text{Split}}(\langle L_{j-1} \rangle^A, \langle R_{j-1} \rangle^A, \lambda)$
  - 13 : Parties com  $\langle M_j \rangle^A = (\langle M_j \rangle^A \leq \langle L_{j-1} \rangle^A) * (\langle L_{j-1} \rangle^A + 1) + (\langle M_j \rangle^A > \langle L_{j-1} \rangle^A \wedge \langle M_j \rangle^A \geq \langle R_{j-1} \rangle^A) * (\langle R_{j-1} \rangle^A - 1)$
  - 14 : Parties compute  $\langle Q_j \rangle^B = \Pi^{\text{Proportion}}(\langle L_{j-1} \rangle^A, \langle R_{j-1} \rangle^A, \langle M_j \rangle^A, \lambda)$
  - 15 : Partie execute  $\Pi^{\text{RandFloat1}}$  and obtain  $\langle U_j^* \rangle^B$
  - 16 : Parties copmute  $\langle \text{cond}_{U_j \leq Q_j} \rangle^A = \text{B2A}(\langle U_j^* \rangle^B \leq Q_j)$
  - 17 : Parties compute  $\langle \text{cond}_{U_j > Q_j} \rangle^A = \text{B2A}(\langle U_j^* \rangle^B > Q_j)$
  - 18 : Parties compute  $\langle R_j \rangle^A = \langle \text{cond}_{U_j \leq Q_j} \rangle^A * M + \langle \text{cond}_{U_j > Q_j} \rangle^A * \langle R_j \rangle^A$ .
  - 19 : Parties compute  $\langle L_j \rangle^A = \langle \text{cond}_{U_j > Q_j} \rangle^A * M + \langle \text{cond}_{U_j \leq Q_j} \rangle^A * \langle L_j \rangle^A$ .
  - 20 : Parties compute  $\langle \text{flag}_j \rangle^B = \text{A2B}(\langle L_j \rangle^A + 1 \geq \langle R_j \rangle^A)$
- 21 : Parties compute  $(\langle e_0 \rangle^B, \dots, \langle e_{\text{iter}-1} \rangle^B) = \Pi^{\text{PreOr}}(\langle \text{flag}_0 \rangle^B, \dots, \langle \text{flag}_{\text{iter}-1} \rangle^B)$
- 22 : For  $j \in [1 \dots \text{iter}]$ , parties compute  $\langle p_j \rangle^B = \langle e_j \rangle^B - \langle e_{j-1} \rangle^B$ , where  $\langle p_0 \rangle^B = \langle e_0 \rangle^B$
- 23 : Parties compute  $\langle x \rangle^A = \sum_{j=0}^{\text{iter}-1} \text{B2A}(\langle p_j \rangle^B) * \langle R_j \rangle^A$

**Algorithm 3.5:** MPC protocol for geometric distribution with  $p = 1 - e^{-\lambda}$ .

### 3.5.2 Gaussian Mechanism

For privacy mechanism  $M(D) = f_r(D) + ir$ , we sample  $i$  from a binomial distribution  $Bino(n, p)$  (PDF:  $Pr(x | n, p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$ , where  $x \in \mathbb{R}$ ) instead of a Gaussian distribution  $\mathcal{N}(x | \mu, \sigma)$  (PMF:  $Pr(x | \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$ , where  $x \in \mathbb{N}$ ).

**TODO: Prove why binomial can approximate gaussian distribution** The binomial distribution  $Bino(n, p)$  is the discrete probability distribution of the number of successes (with probability  $p$ ) in a sequence of  $n$  independent experiments. Further, suppose  $x \sim Bino(n_x, p)$  and  $y \sim Bino(n_y, p)$ , then we have  $x + y$  is a binomial random variable sampled from  $Bino(n_x + n_y, p)$  [Dev86, Lemma 4.3].  $x, y$  can be generated by flipping  $n$  unbiased coins independently and count the numbers of *head*. Then based on the above property, we can generate  $z \sim Bino(n, p)$  by aggregating the binomial variable  $z_i \sim Bino(n_i, p)$  with small  $n_i$ , where  $n = \sum n_i$ . However, for very large  $n$  (e.g.,  $2^{96}$ ), this method would be inefficient. A rejection based sampling algorithm [BKP<sup>+</sup>14] is deployed and modified.

**TODO: how to modify exactly**

After modification, we have  $Algo^{Binomial}(\sqrt{n})$  [BKP<sup>+</sup>14; Tea20] which generate  $i \sim Bino(n, p = 0.5)$  with input  $\sqrt{n}$ , where  $n \leq 2^{96}$ . One advantage of  $Algo^{Binomial}(\sqrt{n})$  is that it only operated on  $\sqrt{n}$  instead of  $n$ . If  $n \leq 2^{96}$ ,  $\sqrt{n} \leq 2^{48}$  can be represented exactly as interger under floating-point implementation.

$Algo^{RandInt}(m)$  generates a uniform random integer  $l \in [0 \dots m)$  and  $Algo^{Bernoulli}(p = \frac{\tilde{p}(i)}{f})$  generates a bernoulli random variable  $c \sim Bern(p = \frac{\tilde{p}(i)}{f})$ .

**Algorithm:**  $Alg^{Binomial}(\sqrt{n})$

```

Input: None
Output:  $i \sim Bino(n, p = 0.5)$ 
1:  $m \leftarrow \lfloor \sqrt{2} * \sqrt{n} + 1 \rfloor$ 
2: WHILE TRUE
3:    $s \leftarrow Alg^{Geometric}$ 
4:    $U \leftarrow Alg^{RandFloat1}$ 
5:   IF  $U < 0.5$ 
6:      $k \leftarrow s$ 
7:   ELSE
8:      $k \leftarrow -s - 1$ 
9:    $l \leftarrow Alg^{RandInt}(m)$ 
10:   $i \leftarrow km + l$ 
11:  IF  $-\frac{\sqrt{n \ln n}}{2} \leq i \leq \frac{\sqrt{n \ln n}}{2}$ 
12:     $\tilde{p}(i) = \sqrt{\frac{2}{\pi n}} * e^{-\frac{2i^2}{n}} * (1 - \nu_n)$ , where  $\nu_n = \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}$ 
13:    IF  $\tilde{p}(i) > 0$ 
14:       $f \leftarrow \frac{4}{m * 2^s}$ 
15:       $c \leftarrow Alg^{Bernoulli}\left(p = \frac{\tilde{p}(i)}{f}\right)$ 
16:      IF  $c == 0$ 
17:        RETURN  $i$ 

```

**Algorithm 3.6:** Algorithm for binomial distribution with large  $n$ .

Next, we convert  $Alg^{Binomial}(\sqrt{n})$  into MPC protocols.



**Algorithm:**  $\Pi^{Binomial}(\sqrt{n})$ 
**Input:** None

**Output:**  $\langle i \rangle^B \sim \text{Bino}(n, p = 0.5)$ 

- 1 : Each party locally calculate  $m = \lfloor \sqrt{2} * \sqrt{n} + 1 \rfloor$ ,  $i_{min} = -\frac{\sqrt{n \ln n}}{2}$ ,  $i_{max} = \frac{\sqrt{n \ln n}}{2}$ ,  
 $v_n = \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}$  and  $p_{coe} = \sqrt{\frac{2}{\pi n}} * (1 - v_n)$
- 2 : For  $j \in [0 \dots iter)$
- 3 : Parties run  $\langle s_j \rangle^B = \Pi^{Geometric}$
- 4 : Parties run  $\langle s'_j \rangle^B = \text{MUL}(\text{ADD}(\langle s_j \rangle^B, 1), -1)$
- 5 : Parties run  $\langle U_j^* \rangle^B = \Pi^{RandFloat1}$
- 6 : Parties compute  $\langle \text{cond}_{U_j^* < 0.5} \rangle^B = \text{CMP}(0.5, \langle U_j^* \rangle^B)$
- 7 : Parties compute  $\langle k_j \rangle^B = \langle s_j \rangle^B * \langle \text{cond}_{U_j^* < 0.5} \rangle^B + \langle s'_j \rangle^B * \text{NOT}(\langle \text{cond}_{U_j^* < 0.5} \rangle^B)$
- 8 : Parties run  $\langle l_j \rangle^B = \Pi^{RandInt}(m)$
- 9 : Parties calculate  $\langle i_j \rangle^B = \text{ADD}(\text{MUL}(\langle k_j \rangle^B, m), \langle l_j \rangle^B)$
- 10 : Parties calculate  $\langle \text{cond}_{i_{min} \leq i_j \leq i_{max}} \rangle^B = \text{CMP}(\langle i_j \rangle^B, i_{min}) * \text{CMP}(i_{max}, \langle i_j \rangle^B)$
- 11 : Parties calculate  $\langle t_{i_j^2} \rangle^B = \text{SQRT}(\langle i_j \rangle^B)$ .
- 12 : Parties calculate  $\langle t_{2*i_j^2} \rangle^B = \text{MUL}(2, \langle t_{i_j^2} \rangle^B)$ .
- 13 : Parties calculate  $\langle t_{2*i_j^2/n} \rangle^B = \text{DIV}(\langle t_{2*i_j^2} \rangle^B, n)$ .
- 14 : Parties calculate  $\langle t_{-2*i_j^2/n} \rangle^B = \text{MUL}(\langle t_{2*i_j^2/n} \rangle^B, -1)$ .
- 15 : Parties calculate  $\langle t_{\exp(-2*i_j^2/n)} \rangle^B = \text{EXP}(\langle t_{-2*i_j^2/n} \rangle^B)$ .
- 16 : Parties calculate  $\langle p_j \rangle^B = \text{MUL}(p_{coe}, \langle t_{\exp(-2*i_j^2/n)} \rangle^B)$ .
- 17 : Parties calculate  $\langle \text{cond}_{p_j > 0} \rangle^B = \text{CMP}(\langle p_j \rangle^B, 0)$
- 18 : Parties calculate  $\langle f_j \rangle^B = \text{DIV}(4, \text{MUL}(m, \text{EXP2}(\langle s_j \rangle^B)))$
- 19 : Parties calculate  $\langle p_{bernoulli} \rangle^B = \text{DIV}(\langle p_j \rangle^B, \langle f_j \rangle^B)$
- 20 : Parties run  $\langle c \rangle^B = \Pi^{Bernoulli}(\langle p_{bernoulli} \rangle^B)$
- 21 : Parties calculate  $\langle \text{cond}_{c=0} \rangle^B = \text{EQ}(\langle c \rangle^B, 0)$
- 22 : Parties calculate  $\langle \text{cond}_{final} \rangle^B = \langle \text{cond}_{i_{min} \leq i_j \leq i_{max}} \rangle^B * \langle \text{cond}_{p_j > 0} \rangle^B * \langle \text{cond}_{c=0} \rangle^B$
- 23 : Parties calculate  $\langle i_j \rangle^B = \text{ADD}(\langle \text{cond}_{final} \rangle^B * \langle i_j \rangle^B, \text{NOT}(\langle \text{cond}_{final} \rangle^B) * (-1))$

**Protocol 3.8:** MPC protocol for binomial distribution with large  $n$ .

### 3.6 Discrete Gaussian Noise

In this part, we introduce the Algorithm for discrete Gaussian mechanism [CKS20].

(deployed in the 2020 US Census [20220])

**TODO: discrete Gaussian mechanism proves, cnetral differetnial privacy in preliminary part, related works** A discrete Gaussian distribution  $\mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)$  has probability mass function

$$Pr(x | \mu, \sigma^2) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sum_{y \in \mathbb{Z}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}}, \text{ where } x \in \mathbb{Z}.$$

$Alg^{DiscreteGauss}$  [CKS20] is as follows:

**Algorithm:**  $Alg^{DiscreteGauss}$

**Input:** None

**Output:**  $x \sim \mathcal{N}_{\mathbb{Z}}(\mu = 0, \sigma^2)$

```

1:  $t \leftarrow \lfloor \sigma \rfloor + 1$ 
2: WHILE TRUE
3:    $L \leftarrow Alg^{DiscreteLap}(t)$ 
4:    $B \leftarrow Alg^{BernoulliEXP}\left(e^{-\frac{(\lfloor \mu \rfloor - \frac{\sigma^2}{t})^2}{2\sigma^2}}\right)$ 
5:   IF  $B == 0$ 
6:     CONTINUE
7:   ELSE
8:     RETURN  $x \leftarrow L$ 
```

**Algorithm 3.7:** Algorithm for discrete Gaussian.

**Protocol:**  $\Pi^{DiscreteGauss}(\sigma)$

**Input:**  $\sigma$

**Output:**

- 1 : Each party locally calculate  $t = \lfloor \sigma \rfloor + 1$
- 2 : For  $j \in [0, \dots, iter)$
- 3 : Parties run  $\langle sign_j \rangle^B, \langle m_j \rangle^{B, UI} = \Pi^{DiscreteLap}(t, 1)$
- 4 : Parties calculate  $\langle b_j \rangle^B = \Pi^{BernoulliEXP} \left( \frac{(\langle |m_j| \rangle^{B, F} - \frac{\sigma^2}{t})^2}{2\sigma^2} \right)$
- 5 : Parties run  $\langle B_j \rangle^B = \Pi^{BernoulliEXP}()$
- 6 : choose xxxxx

$Algo^{DiscreteLap}(b = \frac{numer}{denom})$  [CKS20] generate a random variable  $x \sim Lap_{\mathbb{Z}}(b = \frac{numer}{denom})$ .

**TODO:** prove correctness

**Algorithm:**  $Algo^{DiscreteLap}(b = \frac{numer}{denom})$

**Input:**  $b = \frac{numer}{denom}$

**Output:**  $x \sim Lap_{\mathbb{Z}}(b = \frac{t}{s})$

- 1 : **WHILE TRUE**
- 2 :  $s \leftarrow Algo^{Bernoulli}(0.5)$
- 3 :  $m \leftarrow Algo^{GeometricExp}(\frac{denom}{numer})$
- 4 : **IF**  $s == 1 \wedge m == 0$
- 5 : **CONTINUE**
- 6 : **RETURN**  $x \leftarrow m(1 - 2s)$

**Algorithm 3.8:** Algorithm for discrete Laplacian  $x \sim Lap_{\mathbb{Z}}(b = \frac{numer}{denom})$ .

## List of Figures

---

1.1	DP setting. . . . .	6
1.2	Deterministic algorithm (need reproduce). . . . .	9
1.3	Indeterminism algorithm with small noise ( $b = 0.005$ ) (need reproduce). .	10
1.4	Indeterminism algorithm with large noise ( $b = 0.05$ ) (need reproduce). . .	10
1.5	Centralized DP mode (need reproduce). . . . .	14
1.6	Local DP mode (need reproduce). . . . .	14

## List of Tables

---

1.1	Inpatient microdata [MKGV07]. . . . .	2
1.2	4 – <i>anonymous</i> inpatient microdata [MKGV07]. . . . .	3
1.3	Database example. . . . .	5

## List of Abbreviations

---

## Bibliography

---

- [20220] U. S. C. 2020. “**Understanding the April 2021 Demonstration Data**”. 2020.
- [ABZS12] M. ALIASGARI, M. BLANTON, Y. ZHANG, A. STEELE. “**Secure computation on floating point numbers**”. In: *Cryptology ePrint Archive* (2012).
- [BK15] E. BARKER, J. KELSEY. “**Recommendation for Random Number Generation Using Deterministic Random Bit Generators**”. Computer Security Division Information Technology Laboratory, 2015.
- [BKP<sup>+</sup>14] K. BRINGMANN, F. KUHN, K. PANAGIOTOU, U. PETER, H. THOMAS. “**Internal DLA: Efficient simulation of a physical growth model**”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2014, pp. 247–258.
- [CKS20] C. L. CANONNE, G. KAMATH, T. STEINKE. “**The discrete gaussian for differential privacy**”. In: *arXiv preprint arXiv:2004.00010* (2020).
- [Com19] M. S. COMMITTEE. “**IEEE Standard for Floating-Point Arithmetic**”. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84.
- [Cov19] C. COVINGTON. “**Snapping Mechanism Notes**”. [https://github.com/ctcovington/floating\\_point/blob/master/snapping\\_mechanism/notes/snapping\\_implementation\\_notes.pdf](https://github.com/ctcovington/floating_point/blob/master/snapping_mechanism/notes/snapping_implementation_notes.pdf). 2019.
- [Cov21] C. COVINGTON. “**Snapping Mechanism Implementation in Python**”. <https://github.com/danrr/Snapping-mechanism>. 2021.
- [Dev86] L. DEVROYE. “**Non-Uniform Random Variate Generation**”. Springer New York, 1986.
- [DKM<sup>+</sup>06] C. DWORK, K. KENTHAPADI, F. MCSHERRY, I. MIRONOV, M. NAOR. “**Our data, ourselves: Privacy via distributed noise generation**”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, pp. 486–503.
- [DMNS06] C. DWORK, F. MCSHERRY, K. NISSIM, A. SMITH. “**Calibrating noise to sensitivity in private data analysis**”. In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284.
- [DN03] I. DINUR, K. NISSIM. “**Revealing information while preserving privacy**”. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2003, pp. 202–210.
- [DR<sup>+</sup>14] C. DWORK, A. ROTH. “**The algorithmic foundations of differential privacy**”. In: *Found. Trends Theor. Comput. Sci.* 9.3-4 (2014), pp. 211–407.

- [Dwo06] C. DWORK. “**Differential privacy**”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 1–12.
- [GMP16] I. GAZEAU, D. MILLER, C. PALAMIDESSI. “**Preserving differential privacy under finite-precision semantics**”. In: *Theoretical Computer Science* 655 (2016), pp. 92–108.
- [GMW19] O. GOLDBREICH, S. MICALI, A. WIGDERSON. “How to play any mental game, or a completeness theorem for protocols with honest majority”. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 307–328.
- [GS17] J. GEORGE, M. SISCO. “**Snapping Mechanism Implementation in Rust**”. <https://github.com/lvoid/DP-Snapping-Mech>. 2017.
- [LLV07] N. LI, T. LI, S. VENKATASUBRAMANIAN. “**t-closeness: Privacy beyond k-anonymity and l-diversity**”. In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE. 2007, pp. 106–115.
- [LLV09] N. LI, T. LI, S. VENKATASUBRAMANIAN. “**Closeness: A new privacy measure for data publishing**”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.7 (2009), pp. 943–956.
- [Mir12] I. MIRONOV. “**On significance of the least significant bits for differential privacy**”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 650–661.
- [MKGv07] A. MACHANAVAJJHALA, D. KIFER, J. GEHRKE, M. VENKATASUBRAMANIAN. “**l-diversity: Privacy beyond k-anonymity**”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), 3–es.
- [PR96] J. K. PATEL, C. B. READ. “**Handbook of the normal distribution**”. Vol. 150. CRC Press, 1996, pp. 28–29.
- [SS98] P. SAMARATI, L. SWEENEY. “**Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression**”. In: (1998).
- [Swe97] L. SWEENEY. “**Weaving technology and policy together to maintain confidentiality**”. In: *The Journal of Law, Medicine & Ethics* 25.2-3 (1997), pp. 98–110.
- [Tea20] G. D. P. TEAM. “**Secure Noise Generation**”. [https://github.com/google/differential-privacy/blob/main/common\\_docs/Secure\\_Noise\\_Generation.pdf](https://github.com/google/differential-privacy/blob/main/common_docs/Secure_Noise_Generation.pdf). 2020.
- [Vad17] S. VADHAN. “The complexity of differential privacy”. In: *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 347–450.
- [Wal74] A. J. WALKER. “**Fast generation of uniformly distributed pseudorandom numbers with floating-point representation**”. In: *Electronics Letters* 10.25 (1974), pp. 533–534.



## *Bibliography*

---

- [Whi97] C. R. WHITNEY. “**Jeanne Calment, World’s Elder, Dies at 122**”. 1997.
- [Zum15] N. ZUMEL. “**A Simpler Explanation of Differential Privacy**”. 2015.

## A Appendix

---

### A.1 Algorithms

The geometric distribution  $Geo(p)$  (PMF:  $\Pr(x|p) = (1-p)^{x-1}p$ ) generate a random variable  $x$  by counting the number of Bernoulli trials up and including the first success (with success probability  $p$ ).

$Algo^{Geometric}$  [Wal74; Tea20] generate a geometric random variable  $x \sim Geo(p = 0.5)$  by first generating an 8-bit random string  $r \in \{0, 1\}^8$  (eight Bernoulli trials) and counting its leading zeros (number of trials before the first success) into  $x$ . If all the bits in  $r$  are zeros, a new 8-bit random string is generated and its leading zeros is counted into  $x$ . This process repeats until the random strings contain one (the first success trial).

<p><b>Algorithm:</b> <math>Algo^{Geometric}</math></p> <hr/> <p><b>Input:</b> None <b>Output:</b> <math>x \sim Geo(p = 0.5)</math></p> <pre>1: <math>x \leftarrow 1</math> 2: <b>WHILE</b> <math>r == 0</math> 3:   <math>r \leftarrow Random8bits()</math> 4:   <math>x \leftarrow x + LeadingZeros(r)</math> 5: <b>RETURN</b> <math>x</math></pre>
--

**Algorithm A.1:** Algorithm for geometric distribution  $x \sim Geo(p = 0.5)$ .

Bernoulli distribution  $Bern(p)$  is a discrete probability distribution of a random variable  $x$  which takes the value 1 with probability  $p$  and value 0 with probability  $1 - p$ .

$Algo^{Bernoulli}(p)$  generates a random variable  $x \sim Bern(p)$  by first generating a random variable  $u \sim Uniform(0, 1)$  and output  $x = 1$  if  $u < p$ ,  $x = 0$  otherwise.

**Algorithm:**  $AlgO^{Bernoulli}(p)$

**Input:**  $p$   
**Output:**  $x \sim Bern(p)$   
1:  $u \leftarrow Uniform(0, 1)$   
2: **IF**  $u < p$   
3:     **RETURN**  $x \leftarrow 1$   
4: **ELSE**  
5:     **RETURN**  $x \leftarrow 0$

**Algorithm A.2:** Algorithm for Bernoulli distribution  $x \sim Bern(p)$ .

$AlgO^{BernoulliEXP1}(\gamma)$  [CKS20] generates a random variable  $x \sim Bern(p = e^{-\gamma})$  for  $\gamma \in [0, 1]$ .  
MOD2( $j$ ) outputs 1 if  $j$  is an odd, and 0 otherwise.

**TODO: Prove correctness**

**Algorithm:**  $AlgO^{BernoulliEXP1}(\gamma)$

**Input:**  $\gamma$   
**Output:**  $x \sim Bern(p = e^{-\gamma})$ , where  $\gamma \in [0, 1]$   
1:  $j \leftarrow 1$   
2: **WHILE** TRUE  
3:      $b \leftarrow AlgO^{Bernoulli}\left(\frac{\gamma}{j}\right)$   
4:     **IF**  $b == 1$   
5:          $j \leftarrow j + 1$   
6:     **ELSE**  
7:         **BREAK**  
8: **RETURN**  $x \leftarrow MOD2(j)$

**Algorithm A.3:** Algorithm for Bernoulli distribution  $x \sim Bern(p = e^{-\gamma})$  for  $\gamma \in [0, 1]$ .

$AlgO^{BernoulliEXP}(\gamma)$  [CKS20] generates a random variable  $x \sim Bern(p = e^{-\gamma})$ .

**TODO: Prove correctness**

<p><b>Algorithm:</b> <math>Algo^{BernoulliEXP}(\gamma)</math></p> <hr style="border: 0.5px solid black; margin: 5px 0;"/> <p><b>Input:</b> None</p> <p><b>Output:</b> <math>x \sim Bernoulli(p = e^{-\gamma})</math></p> <pre> 1: IF <math>\gamma \in [0, 1]</math> 2:   <math>b_1 \leftarrow Algo^{BernoulliEXP1}(\gamma)</math> 3:   RETURN <math>x \leftarrow b_1</math> 4: ELSE 5:   FOR <math>j = 0</math> TO <math>\lfloor \gamma \rfloor - 1</math> 6:     <math>b_2 \leftarrow Algo^{BernoulliEXP1}(-1)</math> 7:     IF <math>b_2 == 0</math> 8:       RETURN <math>x \leftarrow 0</math> 9:     <math>b_3 \leftarrow Algo^{BernoulliEXP1}(\lfloor \gamma \rfloor - \gamma)</math> 10:  RETURN <math>x \leftarrow b_3</math> </pre>
--

**Algorithm A.4:** Algorithm for Bernoulli distribution  $x \sim Bern(p = e^{-\gamma})$ .

$Algo^{GeometricExp}(p = 1 - e^{\frac{numer}{denom}})$  [CKS20] generate a random variable  $x \sim Geo(p = 1 - e^{\frac{numer}{denom}})$ .

TODO: prove correctneses

**Algorithm:**  $Algo^{GeometricExp}(p = 1 - e^{\frac{numer}{denom}})$

**Input:** None

**Output:**  $x \sim Geo(p = 1 - e^{\frac{numer}{denom}})$

```

1: IF numer == 0
2:   RETURN 0
3: WHILE TRUE
4:    $u \leftarrow Algo^{RandInt}(denom)$ 
5:    $b_1 \leftarrow Algo^{BernoulliEXP1}(\frac{u}{denom})$ 
6:   IF  $b_1 == 1$ 
7:     BREAK
8:    $k \leftarrow 0$ 
9:   WHILE TRUE
10:     $b_2 \leftarrow Algo^{BernoulliEXP1}(1)$ 
11:    IF  $b_2 == 1$ 
12:       $k \leftarrow k + 1$ 
13:    ELSE
14:      BREAK
15: RETURN  $x \leftarrow \frac{k * denom + u}{numer}$ 

```

**Algorithm A.5:** Algorithm for geometric distribution  $x \sim Geo(p = 1 - e^{\frac{numer}{denom}})$ .

$Algo^{RandInt}(m)$  uses the Simple Modular Method [BK15] to generate random integer  $x$  s.t.  $0 \leq x \leq m - 1$ .  $l$  is the number of bits needed to represent value  $m - 1$  and  $\kappa \geq 64$  is the security parameter. **TODO: security analyse**

**Algorithm:**  $Algo^{RandInt}(m)$

**Input:**  $m$

**Output:**  $x$

```

1: Generate random bits  $b_0, \dots, b_{l+\kappa-1}$ 
2: Calculate  $r = \sum_{j=0}^{l+\kappa-1} 2^j b_j$ 
3: Calculate  $x = r \bmod m$ 

```

**Algorithm A.6:** Algorithm for generate random integer  $x \in [0, \dots, m)$ .

## A.2 MPC Protocols: Building Blocks

**TODO: ABY share conversion in preliminary part**

$Pow2(\langle x \rangle)$  calculates  $2^x$ , and the share type of  $x$  could be boolean or arithmetic sharings.

Protocol  $\Pi^{PreOr}(\langle x \rangle^B)$  calculate the prefix-OR of a  $l$ -bit string  $\langle x \rangle^B = (\langle x_0 \rangle^B, \dots, \langle x_l \rangle^B)$  and output  $\langle y \rangle^B = (\langle y_0 \rangle^B, \dots, \langle y_l \rangle^B)$  such that  $\langle y_j \rangle^B = \bigvee_{k=0}^j \langle x_k \rangle^B$  for  $j \in [0, l]$ , where  $\langle y_0 \rangle^B = \langle x_0 \rangle^B$ .

Protocol  $\Pi^{RandBits}(l)$  can generate a share of a  $l$ -bit public unknown random string  $\langle y \rangle^B = (\langle y_0 \rangle^B, \dots, \langle y_{l-1} \rangle^B)$  without communications between parties. To achieve this, each party locally generate a random  $l$ -bit string  $x$  and set  $\langle y \rangle^B = x$ .

Protocol  $\Pi^{Bits}(x, l)$  outputs the  $l$  least significant bits  $y_0, \dots, y_{l-1}$  of the binary representation of  $x$ , where  $y_0$  is the least significant bit.

Protocol  $\Pi^{Geometric}$  is constructed based on  $Algo^{Geometric}$ . we first unroll the **WHILE** loop by assuming that it iterates less than  $iter$  times.  $u_j$  denote the generated 8-bit string in the  $j$ -th iteration,  $z_j$  denotes the number of leading zeros in 8-bit string  $u_j$ , and  $o_j$  denotes if  $u_j$  contains one. Recall that the geometric distribution counts the number of Bernoulli trials until the first success. We count the total number of zeros in a sequence of 8-bit strings  $u_0, \dots, u_{iter-1}$  until the first appearance of the string whose bits contains 1 (first success trial). Note in line 4–6, we set the  $z_j^{ind} = 1$  for  $j \in [0 \dots k]$  if  $u_{k-1}$  is the first string that contains one and all its  $k-1$  predecessor strings contains only zeros, and set  $z_j^{ind} = 0$  for  $j \in [k \dots iter)$ .

### Protocol: $\Pi^{Geometric}$

**Input:** None

**Output:**  $\langle x \rangle^{B, UI}$ , where  $x \sim Geo(p = 0.5)$

- 1 : For  $j \in [0 \dots iter)$ , each party locally run  $\Pi^{UniformBits}(8)$  in parallel and obtain  $\langle u_j \rangle^B$ .
- 2 : For  $j \in [0 \dots iter)$ , parties run  $\Pi^{LeadingZeros}(\langle u_j \rangle^B, 8)$  in parallel and obtain  $\langle z_j \rangle^{B, UI}$ .
- 3 : For  $j \in [0 \dots iter)$ , parties run  $\Pi^{BitsContainOne}(\langle u_j \rangle^B)$  in parallel and obtain  $\langle o_j \rangle^B$ .
- 4 : Parties run  $(\langle e_0 \rangle^B, \dots, \langle e_{iter-1} \rangle^B) = \Pi^{PreOr}(\langle o_0 \rangle^B, \dots, \langle o_{iter-1} \rangle^B)$ .
- 5 : For  $j \in [0 \dots iter)$ , parties compute  $\langle d_j \rangle^B = \text{NOT}(\langle e_j \rangle^B)$  in parallel.
- 6 : Each party locally set bits  $(\langle z_0^{ind} \rangle^B, \dots, \langle z_{iter-1}^{ind} \rangle^B) = (1, \langle d_0 \rangle^B, \dots, \langle d_{iter-2} \rangle^B)$ .
- 7 : Parties compute  $\langle x \rangle^{B, UI} = \sum_{j=0}^{iter-1} \langle z_j \rangle^{B, UI} \wedge \langle z_j^{ind} \rangle^B$ .

**Protocol A.1:** MPC Protocol for geometric distribution  $x \sim Geo(p = 0.5)$ .

$\Pi^{\text{Bernoulli}}(\langle p \rangle^{B,FL})$  convert  $\text{Algo}^{\text{Bernoulli}}(p)$  into MPC protocols.

**Protocol:**  $\Pi^{\text{Bernoulli}}(\langle p \rangle^{B,FL})$

**Input:**  $\langle p \rangle^{B,FL}$

**Output:**  $\langle x \rangle^B$ , where  $x \sim \text{Bern}(p)$

- 1: Parties run  $\langle U^* \rangle^{B,FL} = \Pi^{\text{UniformFloat}}$
- 2: Parties calculate  $\langle x \rangle^B = \text{CMP}(\langle p \rangle^{B,FL}, \langle U^* \rangle^{B,FL})$

**Protocol A.2:** MPC Protocol for Bernoulli distribution  $x \sim \text{Bern}(p)$ .

$\Pi^{\text{BernoulliEXP1}}(\langle \gamma \rangle^{B,FL})$  convert  $\text{Algo}^{\text{BernoulliEXP1}}(\gamma)$  [CKS20] into MPC protocols. We assume that  $\text{Algo}^{\text{BernoulliEXP1}}(\gamma)$  executes less than  $iter - 1$  times and return the results. In line 3, the parties set the *flag* to record if the  $b_j = 1$  is satisfied. In line 5, 6, the parties calculate the position  $p_0, \dots, p_{iter-1}$  where  $p_j = 1$  and  $j$  is the smallest number such that  $flag_j = 1$ , i.e., the WHILE loop condition is not satisfied and algorithms return a result  $x = b_j$ .

**Protocol:**  $\Pi^{\text{BernoulliEXP1}}(\langle \gamma \rangle^{B,FL})$

**Input:**  $\langle \gamma \rangle^{B,FL}$

**Output:**  $\langle x \rangle^B$ , where  $x \sim \text{Bern}(p = e^{-\gamma})$  and  $\gamma \in [0, 1]$

- 1: For  $j \in [1 \dots iter)$
- 2: Parties calculate  $\langle p_j \rangle^{B,FL} = \text{DIV}(\langle \gamma \rangle^{B,FL}, j)$
- 3: Parties run  $\langle b_j \rangle^B = \Pi^{\text{Bernoulli}}(\langle p_j \rangle^{B,FL})$
- 4: Parties calculate  $\langle flag_j \rangle^B = \text{EQ}(\langle b_j \rangle^B, 1)$
- 5: Parties run  $(\langle e_0 \rangle^B, \dots, \langle e_{iter} \rangle^B) = \Pi^{\text{PreOr}}(\langle flag_0 \rangle^B, \dots, \langle flag_{iter} \rangle^B)$
- 6: For  $j \in [1 \dots iter)$ , parties compute  $\langle p_j \rangle^B = \langle e_j \rangle^B - \langle e_{j-1} \rangle^B$ , where  $\langle p_0 \rangle^B = \langle e_0 \rangle^B$
- 7: Parties compute  $\langle x \rangle^B = \sum_{j=0}^{iter-1} \langle p_j \rangle^B * \langle b_j \rangle^B$

**Protocol A.3:** MPC Protocol for Bernoulli distribution  $x \sim \text{Bern}(p = e^{-\gamma})$ , where  $\gamma \in [0, 1]$ .

$\Pi^{\text{BernoulliEXP}}(\langle \gamma \rangle^{B,FL})$  convert  $\text{Algo}^{\text{BernoulliEXP}}(\gamma)$  [CKS20] into MPC protocols. We assume that  $\gamma < iter$ . In line 1, 2, the parties calculate the condition and Bernoulli sampling for the first case. In line 3 – 13, the parties calculate the condition and Bernoulli sampling for the second case. In line 6 – 8, the parties extract  $b_2 == 0$  which terminates the for loop under  $iter$  iterations. In line 9 – 12, the parties calculate the index  $j$  where the for loops terminates and compare it with  $\lfloor \gamma \rfloor$  as the condition for case 2. In line 14, the parties calculate

the Bernoulli samples under third case. Finally, the parties calculate  $\langle x \rangle^B = \langle \text{cond}_{\gamma \in [0,1]} \rangle^B * \langle b_1 \rangle^B + \text{NOT}(\langle \text{cond}_{\gamma \in [0,1]} \rangle^B) * (\langle \text{cond}_{b_2} \rangle^B * \langle b_2 \rangle^B + \text{NOT}(\langle \text{cond}_{b_2} \rangle^B) * \langle b_3 \rangle^B)$ .

Note that the assumption  $\gamma < \text{iter}$  reveals information about  $\gamma$ , which implies weaker differential privacy strength. **TODO: analyse about differential privacy**  
 set the *flag* to record if the  $b_j = 1$  is satisfied. In line 5, 6, the parties calculate the position  $p_0, \dots, p_{\text{iter}-1}$  where  $p_j = 1$  and  $j$  is the smallest number such that  $\text{flag}_j = 1$ , i.e., the WHILE loop condition is not satisfied and algorithms return a result  $x = b_j$ .

**Protocol:**  $\Pi^{\text{BernoulliEXP}}(\langle \gamma \rangle^{B,FL})$

**Input:**  $\langle \gamma \rangle^{B,FL}$

**Output:**  $\langle x \rangle^B$ , where  $x \sim \text{Bern}(p = e^{-\gamma})$

- 1 : Parties calculate  $\langle \text{cond}_{\gamma \in [0,1]} \rangle^B = \text{CMP}(\langle \gamma \rangle^{B,FL}, 0) * \text{CMP}(1, \langle \gamma \rangle^{B,FL})$
- 2 : Parties run  $\langle b_1 \rangle^B = \Pi^{\text{BernoulliEXP1}}(\langle \gamma \rangle^{B,FL})$
- 3 : For  $j \in [0, \dots, \text{iter})$
- 4 :     Parties run  $\langle b_{j,2} \rangle^B = \Pi^{\text{BernoulliEXP1}}(-1)$
- 5 :     Parties calculate  $\langle \text{cond}_{b_{j,2}=0} \rangle^B = \text{EQ}(\langle b_{j,2} \rangle^B, 0)$
- 6 : Parties calculate  $(\langle e_0 \rangle, \dots, \langle e_{\text{iter}-1} \rangle) = \Pi^{\text{PreOr}}(\langle b_{0,2} \rangle^B, \dots, \langle b_{\text{iter}-1,2} \rangle^B)$
- 7 : For  $j \in [1 \dots \text{iter})$ , parties compute  $\langle p_j \rangle^B = \langle e_j \rangle^B - \langle e_{j-1} \rangle^B$ , where  $\langle p_0 \rangle^B = \langle e_0 \rangle^B$
- 8 : Parties calculate  $\langle b_2 \rangle^B = \sum_{j=0}^{\text{iter}-1} \langle p_j \rangle^B * \langle b_{j,2} \rangle^B$
- 9 : Parties calculate  $\langle j_{b_{j,2}=0} \rangle^{B,UI} = \Pi^{\text{LeadingZeros}}(\langle p_0 \rangle^B, \dots, \langle p_{\text{iter}-1} \rangle^B)$
- 10 : Parties calculate  $\langle \lfloor \gamma \rfloor \rangle^{B,FL} = \text{Floor}(\langle \gamma \rangle^{B,FL})$
- 11 : Parties calculate  $\langle \lfloor \gamma \rfloor \rangle^{B,UI} = \text{FL2UI}(\langle \lfloor \gamma \rfloor \rangle^{B,FL})$
- 12 : Parties calculate  $\langle \text{cond}_{b_2} \rangle^B = \text{CMP}(\langle \lfloor \gamma \rfloor \rangle^{B,UI}, \langle j_{b_{j,2}=0} \rangle^{B,UI})$
- 13 : Parties calculate  $\langle \lfloor \gamma \rfloor - \gamma \rangle^{B,FL} = \text{SUB}(\langle \lfloor \gamma \rfloor \rangle^{B,FL}, \langle \gamma \rangle^{B,FL})$
- 14 : Parties run  $\langle b_3 \rangle^B = \Pi^{\text{BernoulliEXP1}}(\langle \lfloor \gamma \rfloor - \gamma \rangle^{B,FL})$
- 15 : Parties calculate  $\langle x \rangle^B$

**Protocol A.4:** MPC Protocol for Bernoulli distribution  $x \sim \text{Bern}(p = e^{-\gamma})$ .

$\Pi^{\text{RandInt}}(m)$  convert  $\text{Algo}^{\text{RandInt}}(m)$  into MPC protocols.

**TODO:** Problem with unsigned integer because  $\kappa$  requires be greater than 64.



**Protocol:**  $\Pi^{RandInt}(m)$ 
**Input:**  $m$ 
**Output:**  $\langle x \rangle^{B,UI}$ , where  $x \in [0, \dots, m)$ 

- 1: Each party locally generate  $l + \kappa$ -bit string  $(b_0, \dots, b_{l+\kappa-1})$  and set the as the bits of  $\langle r \rangle^{B,UI} = (\langle b_0 \rangle^B, \dots, \langle b_{l+\kappa-1} \rangle^B)$
- 2: Parties calculate  $\langle x \rangle^{B,UI} = \text{MOD}(\langle r \rangle^{B,UI}, m)$

**Protocol A.5:** MPC Protocol for random integer  $x \leftarrow [0, \dots, m)$ .

**TODO:** explanation **TODO:** analyse about differential privacy,  $iter_2$  leak information of  $j$  in second while loop

$\Pi^{GeometricExp}(numer, denom)$  convert  $\text{Algo}^{GeometricExp}(numer, denom)$  into MPC protocols, where  $numer \neq 0$ . We assume two WHILE loop terminates in  $iter_1$  and  $iter_2$  loops.

**Protocol:**  $\Pi^{GeometricExp}(numer, denom)$ 
**Input:**  $numer, denom$ 
**Output:**  $\langle x \rangle^{B,UI}$ , where  $x \sim \text{Geo}(p = 1 - e^{-\frac{numer}{denom}})$ 

- 1: For  $j \in [0, iter_1)$  Parties run  $\langle u_j \rangle^{B,UI} = \Pi^{RandInt}(denom)$
- 2: Parties calculate  $\langle r_j^1 \rangle^{B,FL} = \text{DIV}(\langle u_j \rangle^{B,UI}, denom)$
- 3: Parties run  $\langle b_j^1 \rangle^B = \Pi^{BernoulliEXP1}(\langle r_j^1 \rangle^{B,FL})$
- 4: Parties calculate  $(\langle e_0^1 \rangle, \dots, \langle e_{iter_1-1}^1 \rangle) = \Pi^{PreOr}(\langle b_0^1 \rangle^B, \dots, \langle b_{iter_1-1}^1 \rangle^B)$
- 5: For  $j \in [1 \dots iter_1)$ , parties compute  $\langle p_j^1 \rangle^B = \langle e_j^1 \rangle^B - \langle e_{j-1}^1 \rangle^B$ , where  $\langle p_0^1 \rangle^B = \langle e_0^1 \rangle^B$
- 6: Parties calculate  $\langle u \rangle^{B,UI} = \sum_{j=0}^{iter_1-1} \langle p_j^1 \rangle^B * \langle u_j \rangle^{B,UI}$
- 7: For  $j \in [0, iter_2)$
- 8: Parties run  $\langle b_j^2 \rangle^B = \Pi^{BernoulliEXP1}(1)$
- 9: Parties calculate  $(\langle e_0^2 \rangle, \dots, \langle e_{iter_2-1}^2 \rangle) = \Pi^{PreOr}(\text{NOT}(\langle b_0^2 \rangle^B), \dots, \text{NOT}(\langle b_{iter_2-1}^2 \rangle^B))$
- 10: For  $j \in [1 \dots iter_2)$ , parties compute  $\langle p_j^2 \rangle^B = \langle e_j^2 \rangle^B - \langle e_{j-1}^2 \rangle^B$ , where  $\langle p_0^2 \rangle^B = \langle e_0^2 \rangle^B$
- 11: Parties run  $\langle k \rangle^{B,UI} = \Pi^{LeadingZeros}(\langle p_0^2 \rangle^B, \dots, \langle p_{iter_2}^2 \rangle^B)$
- 12: Parties calculate  $\langle x \rangle^{B,UI} = \text{FL2UI}(\text{Floor}(\text{DIV}(\text{ADD}(\text{MUL}(k, denom), \langle u \rangle^{B,UI}), numer)))$

**Protocol A.6:** MPC Protocol for geometric distribution  $x \sim \text{Geo}(p = 1 - e^{-\frac{numer}{denom}})$ .

**Protocol:**  $\Pi^{DiscreteLap}(numer, denom)$

**Input:**  $numer, denom$

**Output:**  $\langle sign \rangle^B$  and  $\langle m \rangle^{B,UI}$ , which are the sign and integer part of  $x \sim Lap_{\mathbb{Z}}\left(b = \frac{numer}{denom}\right)$

- 1: For  $j \in [0, \dots, iter)$  Parties run  $\langle s_j \rangle^B = \Pi^{Bernoulli}(0.5)$
- 2: Parties run  $\langle m_j \rangle^{B,UI} = \Pi^{GeometricEXP}(denom, numer)$
- 3: Parties calculate  $\langle flag_j \rangle^B = \text{NOT}\left(\text{EQ}\left(\langle s_j \rangle^B, 1\right) * \text{EQ}\left(\langle m_j \rangle^{B,UI}, 0\right)\right)$
- 4: Parties compute  $(\langle e_0 \rangle^B, \dots, \langle e_{iter-1} \rangle^B) = \Pi^{PreOr}(\langle flag_0 \rangle^B, \dots, \langle flag_{iter-1} \rangle^B)$
- 5: For  $j \in [1 \dots iter)$ , parties compute  $\langle p_j \rangle^B = \langle e_j \rangle^B - \langle e_{j-1} \rangle^B$ , where  $\langle p_0 \rangle^B = \langle e_0 \rangle^B$
- 6: Parties calculate  $\langle s \rangle^B = \sum_{j=0}^{iter-1} \langle s_j \rangle^B * \langle p_j \rangle^B$
- 7: Parties calculate  $\langle m \rangle^{B,UI} = \sum_{j=0}^{iter-1} \langle m_j \rangle^{B,UI} * \langle p_j \rangle^B$
- 8: Parties calculate  $\langle sign \rangle^B = 1 - 2 * \langle s \rangle^B$

**Protocol A.7:** MPC Protocol for discrete Lapalcian  $x \sim Lap_{\mathbb{Z}}\left(b = \frac{numer}{denom}\right)$ .

Protocol  $\Pi^{LeadingZeros}(\langle s \rangle^B, l)$  counts the number of leading zeros in a  $l$ -bit string  $s$ . We apply  $\Pi^{PreOr}$  to find the position  $e$  of the first appearance of one in string  $s$ . Finally, by inverting  $e$ , we get  $e_j = 1$  if  $s_j$  belongs to the leading zeros. In line 3, we convert boolean share to arithmetic share to count the total number of leading zeros (number of  $d_j \neq 0$ ).

**Protocol:**  $\Pi^{LeadingZeros}(\langle s \rangle^B, l)$

**Input:**  $l$ -bit string  $\langle s \rangle^B$

**Output:**  $\langle z \rangle^A$

- 1: Parties run  $\Pi^{PreOr}(\langle s \rangle^B)$  and obtain  $\langle e_j \rangle^B$ .
- 2: For  $j \in [0 \dots l)$ , parties compute  $\langle d_j \rangle^B = \text{NOT}(\langle e_j \rangle^B)$  in parallel.
- 3: Parties compute  $\langle z \rangle^{B,UI} = \sum_{j=0}^{l-1} \text{B2A}(\langle d_j \rangle^B)$ .

**Protocol A.8:** MPC Protocol for counting leading zeros.

Protocol  $\Pi^{BitsContainOne}(\langle s \rangle^B, l)$  outputs  $o = 1$  if the given  $l$ -bit string  $s$  contains one, and  $o = 0$  otherwise.

**Protocol:**  $\Pi^{BitsContainOne}$ 
**Input:**  $l$ -bit string  $\langle s \rangle^B$ 
**Output:**  $\langle o \rangle^B$ 

1 : Parties compute  $\langle o \rangle^B = \bigvee_{j=0}^l \langle s_j \rangle^B$ 

**Protocol A.9:** MPC Protocol for checking if  $l$ -bit string  $s$  contains one.

**TODO: Binary2Unary needs to be improved.**  $\Pi^{Binary2Unary}(\langle a \rangle^A, l)$  [ABZS12] converts integer  $a$  from binary to unary bitwise representation and outputs a  $l$ -bit string  $\mathbf{p} = (p_0, \dots, p_{l-1})$ , where the  $a$  least significant bits  $(p_0, \dots, p_{a-1})$  are set to 1 and others to 0. In line 1, 2, we calculate the  $2^a$  and convert it to boolean shares. Then in line 3 we generate  $l + k$  random bits and hide  $2_a$  by adding it with the  $l + k - 1$ -bit integer and reconstruct the addition result  $c$ . In line 5, the plaintext value  $c$  is decomposed into binary bits. In line 6, we compute XOR of  $c_j$  and correspond bit  $u_j$ . Then in line 7, by *PreOr* we get  $(g_{l-1}, \dots, g_j) = (\bar{1}_{(l-j+1)})$  and  $(g_{j-1}, \dots, g_0) = (\bar{0}_{(j)})$ , where  $j - 1 = a$ . Finally, we calculate  $(p_{l-1}, \dots, p_0) = \text{NOT}(g_{l-1}, \dots, g_0)$  and the number of non-zero bits in  $(p_{l-1}, \dots, p_0)$  equals to  $a$ . The share conversion in line 4 can be omitted if arithmetic of boolean shares is available.

**Protocol:**  $\Pi^{Binary2Unary}(\langle a \rangle^A, l)$ 
**Input:**  $\langle a \rangle^B, l$ 
**Output:**  $\langle p_0 \rangle^B, \dots, \langle p_{l-1} \rangle^B$ 

- 1 : Parties compute  $\langle 2^a \rangle^B = \text{Pow2}(\langle a \rangle^A, l)$
- 2 : Parties compute  $\langle 2^a \rangle^A = \text{B2A}(\langle 2^a \rangle^B)$
- 3 : Parties run  $\Pi^{RandBits}(l + k)$  and obtain  $\langle u_0 \rangle^B, \dots, \langle u_{l+k-1} \rangle^B$ .
- 4 : Parties reconstruct  $c \leftarrow \text{Rec}(\langle 2^a \rangle^A + \text{B2A}(\langle u_{l+k-1} \rangle^B, \dots, \langle u_0 \rangle^B))$
- 5 : Each party locally run  $\Pi^{Bits}(c, l)$  and obtain  $\langle c_0 \rangle^B, \dots, \langle c_{l-1} \rangle^B$
- 6 : For  $j \in [0 \dots l]$ , each party locally compute  $\langle t_j \rangle^B = \text{XOR}(c_j, \langle u_j \rangle^B)$
- 7 : For  $j \in [0 \dots l]$ , parties compute  $\Pi^{PreOr}(\langle t_0 \rangle^B, \dots, \langle t_l \rangle^B)$  and obtain  $\langle g_0 \rangle^B, \dots, \langle g_l \rangle^B$
- 8 : For  $j \in [0 \dots l]$ , parties compute  $\langle p_j \rangle^B = \text{NOT}(\langle g_j \rangle^B)$

**Protocol A.10:** MPC Protocol for binary to unary conversion.