TECHNISCHE
UNIVERSITÄT
DARMSTADT

Thesis Type

This is the Title

Student Name

December 15, 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 15, 2021

Student Name

Abstract

This will be the abstract.

Acknowledgments

If you like, you can add acknowledgments here.

Contents

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's 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

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 value to a broader value range, and the latter method removes specific attribute value. After generalization, the values of attribute *Age* of the first eight records are replaced by value ranges such as < 30 and $>= 40$. 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 [MKGV07].

	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 [LLV07]. Then in 2007, [LLV07] introduced the concept of $t-closeness$ as an enhancement of $l-diversity$. However, [LLV09] 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

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

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 $bino \sim \frac{1}{n} Binomial\left(n, \frac{1}{2}\right)$ 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 $\gamma \in \left[0, \frac{1}{2}\right]$. $\gamma = \frac{1}{2}$ 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 $\gamma \to 0$ or provide more accurate result by setting $\gamma \to \frac{1}{2}$. 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 $\mathbb{E}\left[\tilde{C}\right] = CheatFraction$. The variance of $\tilde{C}$ is

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

With Chebyshev's inequality: For any real random variable Z with expectation μ and variance σ^2,

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

we have

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

where the error term $\left|\tilde{C} - CheatFraction\right| \to 0$ as $n \to \infty$ with high probability. The conclusion is that the error increase as the privacy protection level increases $\gamma \to 0$. To maintain the accuracy, more data $n \to \infty$ 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+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+14]). *A privacy mechanism $M : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$ 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+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+14]). *A privacy mechanism $M : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$ is (ε, δ)-differential privacy if for any two neighboring databases D_1, $D_1 \in \mathcal{X}^n$, and for all $T \subseteq \mathcal{Y}$, we have $\Pr[M(D_0) \in T] \leq e^{\varepsilon} \cdot \Pr[M(D_1) \in T] + \delta$, 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 ε-DP (or pure DP) when $\delta = 0$, and (ε, δ)-DP (or approximate DP) when $\delta \neq 0$.

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

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

Definition 1.1.6 (ℓ_t-sensitivity [DR+14]). *The ℓ_t-sensitivity of a query $f : \mathcal{X}^n \to \mathbb{R}^k$ 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:*

$$Laplace(x|b) = \frac{1}{2b} e^{\left(-\frac{|x|}{b}\right)}. \tag{1.5}$$

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

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

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

$x \sim Gauss(x|b)$ 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 $M(D_1)$ looks *similar* to prevent from being distinguished by the adversary.
- M is called ε-differentially private iff: $\left|\frac{\Pr[M(D_0)\in Q]}{\Pr[M(D_1)\in Q]}\right| \leq e^{\varepsilon}$.

Challenger C		**Adversary** A
input: M		**input**: D_0, D_1, Q
	D_0, D_1 $\longleftarrow$	
$b \leftarrow\!\!\$ \{0,1\}$		
$M(D_b)$,$M(D_{1-b})$		
	$M(D_b)$, $M(D_{1-b})$ $\longrightarrow$	
		$b' = 0$, **if** $M(D_{1-b}) \in Q$
		$b' = 1$, otherwise
		if $b == b'$, A wins.

Protocol 1.1: A motivating example of differential privacy.

Suppose the adversary A has chosen two data sets:

- $D_0 = \{0, 0, 0, \ldots, 0\}$ (100 zeros)
- $D_1 = \{1, 0, 0, \ldots, 0\}$ (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 $M(D) = mean(D)$. 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.

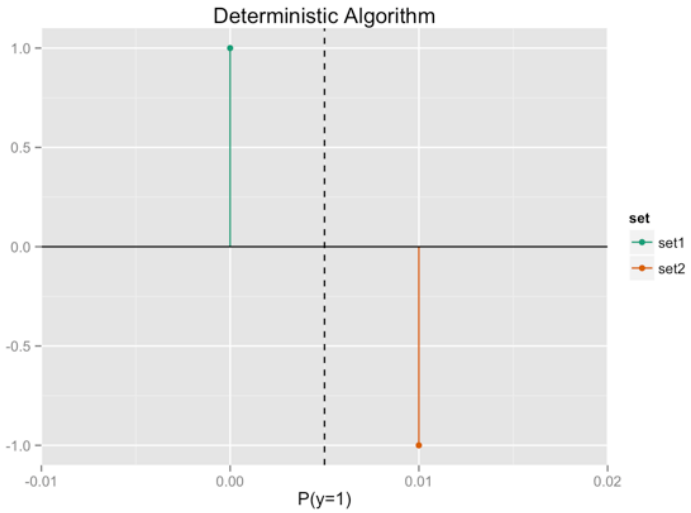


Figure 1.2: Deterministic algorithm (need reproduce).

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 $lap \sim Laplace(b = 0.05)$ 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 $\varepsilon = \log\left(\frac{\text{green area}}{\text{orange area}}\right)$, where ε express the degree of differential privacy and a smaller ε 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.

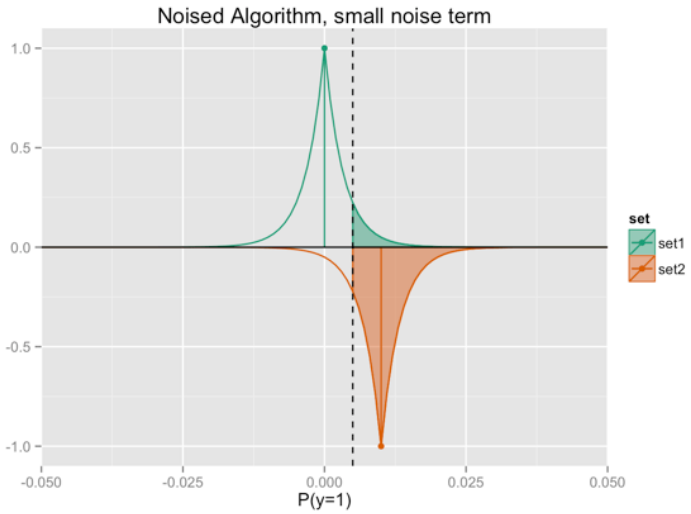


Figure 1.3: Indeterminism algorithm with small noise ($b = 0.005$) (need reproduce).

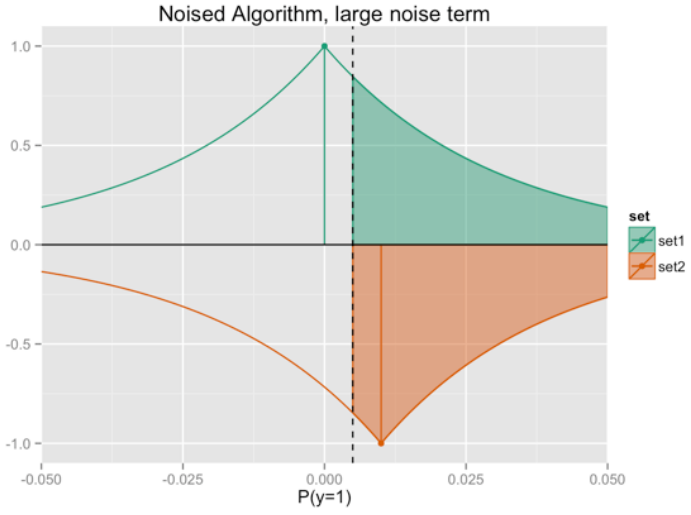


Figure 1.4: Indeterminism algorithm with large noise ($b = 0.05$) (need reproduce).

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.

ε-Differential Privacy

Definition 1.1.9 (Laplace Mechanism [DR+14]). *Let $f : \mathcal{X}^n \to \mathbb{R}^k$. The Laplace mechanism is defined as $M(X) = f(X) + (Y_1, \ldots, Y_k)$, where the Y_i are independent Laplace random variables drawn from distribution $Laplace(Y_i|b) = \frac{1}{2b} e^{\left(-\frac{|Y_i|}{b}\right)}$ with $b = \frac{\Delta f_1^{(f)}}{\varepsilon}$.*

Theorem 1. *Laplace mechanism preservers ε-DP [DR+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 ε, for any arbitrary z and neighboring D_0 and D_1 .

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

□

Definition 1.1.10 (Privacy Loss [DR+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 $\ln(\frac{\Pr[X=t]}{\Pr[Y=t]})$.*

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

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

(ε, δ)-Differential Privacy

ε-DP has high privacy requirement which leads to adding too much noise and affecting the accuracy of the queries. We introduce an relaxation of ε-DP, (ε, δ)-DP.

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

Gaussian mechanism is proved to satisfy (ε, δ)-DP [DR+14].

Similar to ε-DP, (ε, δ)-DP can also be interpreted regarding privacy loss: the absolute value of the privacy loss random variable is bounded by ε with probability $1 - \delta$ [DR+14, Lemma 3.17]. In other words, with probability δ, 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 $M : \mathcal{X}^n \to \mathcal{Y}$ be (ε, δ)-DP mechanism, and let $F : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary randomized mapping. Then $F \circ M$ is $(\varepsilon, \delta = 0)$-DP [DR+14].*

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

Group Privacy

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

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

Basic Composition

Theorem 4. *Suppose $M = (M_1 \ldots M_k)$ is a sequence of $\left(\varepsilon_[i], \delta_i\right)$-differential private mechanisms, where M_i is chosen sequentially and adaptively. Then M is $\left(\sum_{i=1}^n \varepsilon_i, \sum_{i=1}^n \delta_i\right)$-DP [DR+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+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.

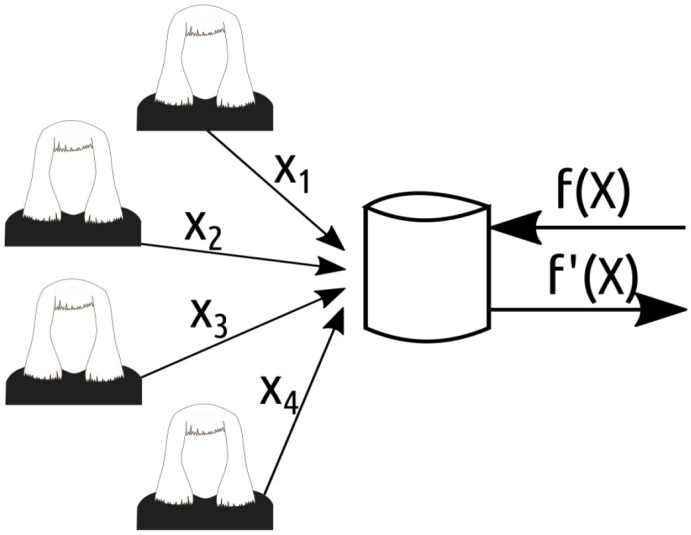


Figure 1.5: Centralized DP mode (need reproduce).

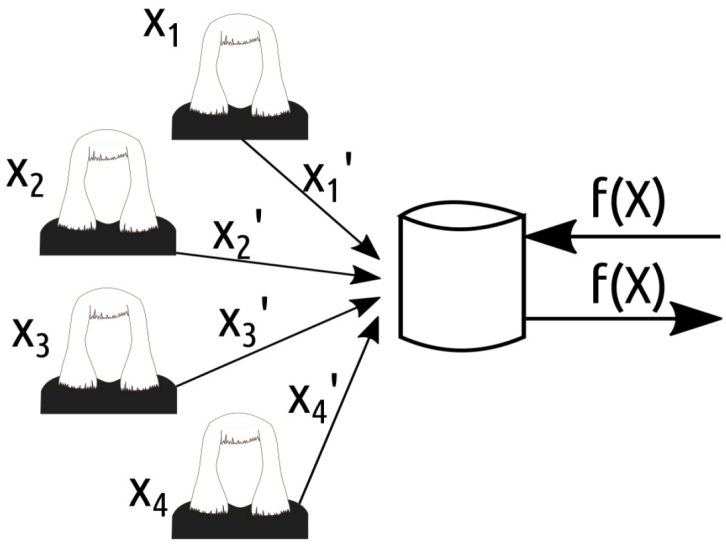


Figure 1.6: Local DP mode (need reproduce).

Advantages of Differential Privacy

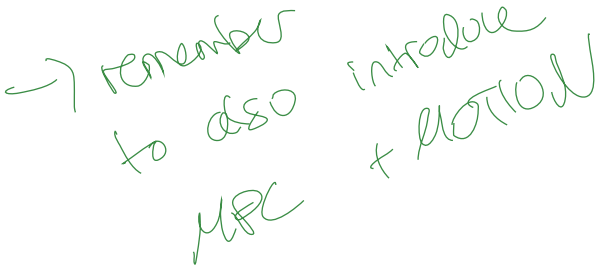
From the example § 1.1.4, we found that DP can still protect privacy even if the adversary knows the database. Generally speaking, DP ensures privacy protection by making no assumption about the adversary's auxiliary information (even when the adversary is the data provider) or computational strategy (regarding the complexity of modern cryptography) [Vad17]. In addition, DP provides a quantitive 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 $D = (D_1, \ldots, D_n)$ 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 $f(pre(D_1), \ldots, pre(D_n))$, 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 $f(pre(D_1), \ldots, pre(D_n))$ should be exactly the same as the result, where a trusted server has access to the whole dataset D and computes $f(pre(D))$ locally, i.e., $f(pre(D_1), \ldots, pre(D_n)) = f(pre(D))$.
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 Π^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 Π^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 $f(D_1, \ldots, D_n)$ since the second requirement can be satisfied with MPC protocols.

Before describing the MPC-DP protocol, let's consider a trival 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 $f(x_1, \ldots, x_n) = \sum_{j=1}^{n} x_j$ (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 (ε, δ)-DP), for $r_i \sim \mathcal{N}(0, \sigma^2)$, $\sigma^2 = \frac{2}{\varepsilon^2} \cdot \ln\left(\frac{1.25}{\delta}\right)$ and $i \in [n]$. Then, they run the MPC protocol Π^f with their private inputs

y_i to compute function f and after reconstruction they get:

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

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

In general, the users in above example can achieve $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-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 $f(y_1, \ldots, y_n)$, where i is the index of the corrupted users. In the worst case when $n-1$ users are corrupted, the differential privacy guarantee of $f(y_1, \ldots, y_n)$ degenerates from $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-DP to (ε, δ)-DP. If it is necessary for the users to have $f(y_1, \ldots, y_n)$ satisfying $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-DP, one solution is to ask each user to add stronger noise $r_i' \sim \mathcal{N}\left(0, n \cdot \sigma^2\right)$ locally before running Π^f. However, the excessive noise $\sum_{j=1}^{n} r_j'$ in the result can lead to severe deviation which can reduce or copmletely 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 copmute 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 $P_0, \ldots, P_N$. For $j \in [N]$, computing party P_j runs MPC protocols for function f with other computing parties using his secret-shared value $\langle D_1 \rangle_j, \ldots, \langle D_n \rangle_j$ (received from n users) and gets $\langle f(D) \rangle_j$. 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 perturbs $\langle f(D) \rangle_j$ by computing $\langle M(D) \rangle_j = \Pi^{Perturbation}\left(\langle f(D) \rangle_j + \langle r \rangle_j\right)$ 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 perturbed (noisy) result $M(D) = \text{Perturb}(f(D) + r)$ is revealed which computes 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: $\Pi^{MPC-DP}(\langle D_1 \rangle, \ldots, \langle D_n \rangle)$

Input: $\langle D_1 \rangle, \ldots, \langle D_n \rangle$

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

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

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

3 : Parties run $\langle M(D) \rangle = \Pi^{Perturb}(\langle f(D) \rangle + \langle r \rangle)$ and output $\langle M(D) \rangle$.

Protocol 2.1: General framework for MPC-DP protocol.

Π^{MPC-DP} describes the general framework of the MPC-DP protocol. Note in Π^{MPC-DP}, the parties first calculate $\langle f(D) \rangle = \Pi^{f}(\langle D_1 \rangle, \ldots, \langle D_n \rangle)$ 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 Π^{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 arithemtic) of MPC-DP protocols that combine secure multi-party computation protocols and differentially private mechanisms.

Recall that differentially private mechanisms [DMNS06; DKM+06] $M(D)$ guarantees differential privacy by adding appropriately chosen random noise to the query function $f(D) \in \mathbb{R}$ of database D, which can be expressed as:

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

where $Y \in \mathbb{R}$ 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 differentially private 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 catastrophical 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 querires to the Laplacian mechanism and exploits the facts that the inversion sampling methods [Dev86] leads to a porous distribution over double-precison 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

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

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. λ is a parameter that controls the privacy strength. Function $\lfloor x \rceil_\Lambda$ rounds x to the nearest multiple of Λ. Note that the snapping mechansim $M_S(f(D), \lambda, B)$ executes fully under floating-point implementations and can be proved to satisfy $\left(\frac{1}{\lambda} + \frac{2^{-49}B}{\lambda}\right)$-DP for $\lambda < B < 2^{46} \cdot \lambda$.

3.1.2 Secure Noise Generation

[Tea20] proposes alternative algorithms to realize both Laplacian and Gaussian mechanisms under floating-point arithmetic securely and yields lower error than the snapping mechanism. Since floating-point numbers cannot approximate continuous real numbers with infinite precision and the implementation of Laplacian mechanism (operating on real number field) under floating-point arithmetic leads to attack as [Mir12] shows. [Tea20] avoid this problem by developing a differential privacy mechanism that operates mostly on integers which is defined as:

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

where r is a resolution parameter that controls the scale of the simulated continuous distribution. i is an integer from a probability distribution (generated by the secure noise generation algorithms) 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. $M_{SNG}(f(D), r)$ can be proved to satisfy differential privacy and its operations can be realized exactly under floating-point arithmetic, then $M_{SNG}(f(D), r)$ can be securely implemented.

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 $f(D) \in \mathbb{Z}$. It is defined as follows:

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

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 $\{x \in \mathbb{Z} \mid 1 \le x \le a\}$, $[a \dots b)$ denotes $\{x \in \mathbb{Z} \mid a \le x < b\}$, (a, b) denotes $\{x \in \mathbb{R} \mid a < x < b\}$, and $[a, b]$ denotes $\{x \in \mathbb{R} \mid a \le x \le b\}$. 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 $\overline{b}_{(l)}$ means repeating the bit b for l times. In the binary representation of floating-point numbers, $a\overline{b}$ 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 $\langle \boldsymbol{x} \rangle^{B,D}$ represents a vector of ℓ Boolean sharing bits which is interpreted as an ℓ-bit number of data type D. The second superscript $D \in \{UI, SI, FP, FL\}$ denoting its data types: UI: unsigned integer, SI: signed integer, FP: fixed-point number, FL: floating-point number. Note that the superscript data type $\{\}^D$ and subscript party index $\langle\rangle_i$ 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 $\langle \boldsymbol{a} \rangle^{B,D} \odot \langle \boldsymbol{b} \rangle^{B,D}$ denotes the arithmetic operations of Boolean shared bit vectors $\langle \boldsymbol{a} \rangle^{B,D}$ and $\langle \boldsymbol{b} \rangle^{B,D}$, where $\odot \in \{+, -, \times, \div, >, ==\}$. $*$ and $\cdot$ are used interchangeable as $\times$.

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

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

For data type conversions, we denote $\langle \boldsymbol{a} \rangle^{B,FL} = \text{UI2FL}\left(\langle \boldsymbol{a} \rangle^{B,UI}\right)$ as the conversion from a Boolean sharing bits vector representing unsigned integer to a floating-point number. Other data type conversion operations such as $\langle \boldsymbol{a} \rangle^{B,UI} = \text{FL2UI}\left(\langle \boldsymbol{a} \rangle^{B,FL}\right)$ 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 $Algo^{Branching}$ Algorithm 3.1 into MPC computations as

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

Protocol: $Algo^{Branching}$

```
    ...
1 : IF ⟨a⟩^B
2 :     RETURN ⟨b⟩^B
3 : ELSE
4 :     RETURN ⟨c⟩^B
    ...
```

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

3.3.2 Loop

Suppose $Algo^{Lottery}$ 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. $Algo^{Toss}$ outputs either 1 (win) or 0 (lose) with the same probability $p = 0.5$.

Algorithm: $Algo^{Lottery}$

```
...
1 : trials ← 0
2 : WHILE TRUE
3 :     a ← Algo^Toss(b)
4 :     IF a
5 :         RETURN r − trials
6 :     trials ← 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 Π^{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 Π^{PreOr} in line 4 to calculate $e_j = \vee_{k=0}^{j} a_k$ 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 ⟨a_j⟩^B = Π^Toss(⟨b⟩^{B,UI}).
4 : Parties compute (⟨e_0⟩^B, ..., ⟨e_{iter−1}⟩^B) = Π^PreOr(⟨a_0⟩^B, ..., ⟨a_{iter−1}⟩^B).
5 : FOR j = 0 TO iter − 1
6 :     Each party locally compute ⟨p_j⟩^B = ⟨e_j⟩^B ⊕ ⟨e_{j−1}⟩^B, where ⟨p_0⟩^B = ⟨e_0⟩^B.
7 : Parties compute ⟨x⟩^{B,UI} = ⊕_{j=0}^{iter−1} ⟨p_j⟩^B ∧ (⟨r⟩^{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 mechnaism [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 noise $Y \sim Lap(b)$. 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, λ 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 calcualtion 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 mechansim is defined as followings:

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

$f(D) \in \mathbb{D}$ 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 palce* (ulp), i.e. spacing between two consecutive floating-point numbers. $\text{LN}(\cdot)$ is the natural logarithm implementation under floating-point with exact rounding. Λ is the smallest power of two greater than or equal to λ, and we have $\Lambda = 2^n$ such that $2^{n-1} < \lambda \leq 2^n$ for $n \in \mathbb{Z}$. $\lfloor \cdot \rceil_\Lambda$ rounds its input to the nearest multiple of Λ exactly by manipulating the binary floating-point representation of the input. Note that the snapping mechanism assumes that the *sensitivity* Δ_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 mechansim 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 Λ.
5. Calculation of $\lfloor \cdot \rceil_\Lambda$.

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:

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

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 $Algo^{RandFloat1}$ with the methods from [Wal74; Mir12], i.e., independently sampling a random variable $x \sim Geo(p = 0.5)$ from a geometric distribution with $Algo^{Geometric}$ 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: $Algo^{RandFloat1}$

Input: None

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

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

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

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 $[.5, 1)$ and 2^{52} reals in interval $[.25, .5)$. 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 Λ

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

We can represent λ 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 Λ 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 Λ by increasing the exponent of λ by 1 and setting all its significant bits $(d_i)_{i \in [52]}$ to 0.

In summary we have

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

Calculation of $\lfloor \cdot \rceil_\Lambda$

$\lfloor x \rceil_\Lambda$ can be done in three steps:

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

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 substracting 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:

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

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:

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

Note that $\left(1 d_1 \dots d_y\right)_2$ are the integer part and $\left(.d_{y+1} \dots d_{52}\right)_2$ is the fraction part. We have $\left(.d_{y+1}\right)_2 = 0.5$ when $d_{y+1} = 1$, and $\left(.d_{y+1}\right)_2 = 0$ when $d_{y+1} = 0$. Therefore, rounding x' to the nearest integer means carrying 1 to the integer part $(1 d_1 \dots d_y)_2$ 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 $\left(d'_1 \dots d'_y\right)_2 = \left(d_1 \dots d_y\right)_2 + 1$.

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

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

Case 4: $y = -1$
x' can be represent without exponent term as:

$$x' = (-1)^S (0.1 d_1 d_2 \dots d_{51})_2.$$

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:

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

Case 5: $y < -1$
x' can be represent without exponent term as:

$$x' = (-1)^S (0.01 d_1 d_2 \dots d_{50})_2.$$

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

$$+0 = (-1)^0 (1.\overline{0})_2 * 2^{(0000000000)_2 - 1023},$$
$$-0 = (-1)^1 (1.\overline{0})_2 * 2^{(0000000000)_2 - 1023}.$$

3. Multiply x'' by Λ 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 $x'' = (-1)^S (1.\overline{0})_2 \times 2^{(0000000000)_2 - 1023 + n}$ doesn't represent the values ± 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:

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

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

$$\langle M_S(f(D), \lambda, B)\rangle = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(\langle f(D)\rangle) \oplus \langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)\right\rceil_\Lambda\right). \tag{3.4}$$

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. λ controls the magnitude of the generated noise and is publicly known. Therefore the calculation of Λ can be done locally by each party or calculated by one party and broadcast to others. The generation of U^* and S, calculation of $\lfloor\cdot\rceil_\Lambda$ 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.

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

$$\begin{aligned} \left\langle \boldsymbol{x_{clampB}} \right\rangle^{B,FL} = \quad & \langle cond_{x<-B} \rangle^{B} \wedge -B \\ & \oplus \langle cond_{x>B} \rangle^{B} \wedge B \\ & \oplus \langle \boldsymbol{x} \rangle^{B,FL} \wedge \text{NOT}\left(\langle cond_{x<-B} \rangle^{B} \vee \langle cond_{x>B} \rangle^{B}\right) \end{aligned} \tag{3.5}$$

Protocol: $\Pi^{Clamp}\left(\langle \boldsymbol{x} \rangle^{B,FL}, B\right)$

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

Output: $\left\langle \boldsymbol{x_{clampB}} \right\rangle^{B,FL}$

1 : Parties compute $\langle cond_{x<-B} \rangle^{B} = \left(\langle \boldsymbol{x} \rangle^{B,FL} < -B\right)$.

2 : Parties compute $\langle cond_{x>B} \rangle^{B} = \left(\langle \boldsymbol{x} \rangle^{B,FL} > B\right)$.

3 : Parties compute $\left\langle \boldsymbol{x_{clampB}} \right\rangle^{B,FL}$.

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

Generation of U^* and S

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

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

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

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

3 : Parties compute $\langle \boldsymbol{e} \rangle^{B,UI} = 1023 - \langle \boldsymbol{x} \rangle^{B,UI}$

Protocol 3.3: MPC protocol for uniform random variable $U^* \in \mathbb{D} \cap (0,1)$.

Calculation of $\lfloor \cdot \rceil_\Lambda$

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

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

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: $\Pi^{XDivideLambda}\left(\langle \boldsymbol{e} \rangle^{B,UI}, n\right)$

Input: $\langle \boldsymbol{e} \rangle^{B,UI}$, n

Output: $\langle \boldsymbol{z} \rangle^{B,UI}$, **where** $z = e - n$

1 : Parties compute $\langle \boldsymbol{z} \rangle^{B,UI} = \langle \boldsymbol{e} \rangle^{B,UI} - n$

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 $\Pi^{RoundDouble2Int}\left(\langle \boldsymbol{d}\rangle^{B,UI}, \langle \boldsymbol{z}\rangle^{B,UI}\right)$ 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 $\Pi^{Binary2Unary}\left(\langle \boldsymbol{y}\rangle^{B,UI}, 52\right)$ 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 $\left(d'_1 \dots d'_y\right)_2 = \left(d_1 \dots d_y\right)_2 + 1$ 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 $z_{0,\overline{1}_{(10)}}$.

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 $z_{\overline{0}_{(11)}}$.

Finally, the parties compute the significant bits $\langle \boldsymbol{d_{round}}\rangle^{B,UI}$ and biased exponent $\langle \boldsymbol{z_{round}}\rangle^{B,UI}$ with

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

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

Protocol: $\Pi^{RoundDouble2Int}\left(\langle \boldsymbol{d}\rangle^{B,UI}, \langle \boldsymbol{z}\rangle^{B,UI}\right)$

Input: $\langle \boldsymbol{d}\rangle^{B,UI} = \left(\langle d_1\rangle^B, \ldots, \langle d_{52}\rangle^B\right)$, $\langle \boldsymbol{z}\rangle^{B,UI} = \left(\langle z_1\rangle^B, \ldots, \langle z_{11}\rangle^B\right)$

Output: $\langle \boldsymbol{d_{round}}\rangle^{B,UI}$, $\langle \boldsymbol{z_{round}}\rangle^{B,UI}$

1 : Parties compute $\langle \boldsymbol{y}\rangle^{B,UI} = \langle \boldsymbol{z}\rangle^{B,UI} - 1023$.

2 : Parties compute $\langle cond_{case1}\rangle^B = \left(\langle \boldsymbol{y}\rangle^{B,UI} \geq 52\right)$.

3 : Parties compute $\langle cond_{case2}\rangle^B = \left(\langle \boldsymbol{y}\rangle^{B,UI} == 0\right)$.

4 : Parties compute $\left\langle \boldsymbol{z_{d_1+1023}}\right\rangle^{B,UI} = \langle d_1\rangle^B \wedge 1 + 1023$.

5 : Each party locally set $\left\langle \boldsymbol{d_{\bar{0}_{(52)}}}\right\rangle^{B,UI} = \left(\bar{0}_{(52)}\right)$.

6 : Parties compute $\langle cond_{case3}\rangle^B = \left(\langle \boldsymbol{y}\rangle^{B,UI} \geq 1\right) \wedge \left(\langle \boldsymbol{y}\rangle^{B,UI} \leq 51\right)$.

7 : Parties run $\left(\langle p_1\rangle^B, \ldots, \langle p_{52}\rangle^B\right) = \Pi^{Binary2Unary}\left(\langle \boldsymbol{y}\rangle^{B,UI}, 52\right)$.

8 : Parties compute $\left\langle \boldsymbol{d_{1,\ldots,y,\bar{0}_{(52-y)}}}\right\rangle^{B,UI} = \left(\langle p_1\rangle^B \wedge \langle d_1\rangle^B, \ldots, \langle p_{52}\rangle^B \wedge \langle d_{52}\rangle^B\right)$.

9 : **FOR** $j = 1$ **TO** 51

10 : Each party locally compute $\left\langle c_{j+1}\right\rangle^B = \left\langle p_j\right\rangle^B \oplus \left\langle p_{j+1}\right\rangle^B$, where $\langle c_1\rangle^B = 0$.

11 : Parties compute $\left\langle d_{y+1}\right\rangle^B = \oplus_{j=1}^{52} \left\langle c_j\right\rangle^B \wedge \left\langle d_j\right\rangle^B$.

12 : Parties compute $\left\langle \boldsymbol{d_{sum(1,\ldots,y)}}\right\rangle^{B,UI} = \text{A2B}\left(\sum_{j=1}^{52} \text{B2A}\left(\left\langle p_j\right\rangle^B \wedge \left\langle y_j\right\rangle^B\right)\right)$.

13 : Parties compute $\left\langle \boldsymbol{d_{1',\ldots,y',\bar{0}_{(52-y)}}}\right\rangle^{B,UI} = \left\langle \boldsymbol{d_{1,\ldots,y,\bar{0}_{(52-y)}}}\right\rangle^{B,UI} + \text{POW2}\left(52 - \langle \boldsymbol{y}\rangle^{B,UI}\right)$.

14 : Parties compute $\langle cond_{case3a}\rangle^B = \left(\left\langle \boldsymbol{d_{sum(1,\ldots,y)}}\right\rangle^{B,UI} < \langle \boldsymbol{y}\rangle^{B,UI}\right)$.

15 : Parties compute $\langle cond_{case3b}\rangle^B = \left(\left\langle \boldsymbol{d_{sum(1,\ldots,y)}}\right\rangle^{B,UI} == \langle \boldsymbol{y}\rangle^{B,UI}\right)$.

16 : Parties compute $\left\langle \boldsymbol{z_{y+1+1023}}\right\rangle^{B,UI} = \langle \boldsymbol{z}\rangle^{B,UI} + 1 + 1023$.

17 : Parties compute $\langle cond_{case3c}\rangle^B = \text{NOT}\left(\left\langle d_{y+1}\right\rangle^B\right)$.

18 : Parties compute $\left\langle cond_{case4}\right\rangle^B = \left(\langle \boldsymbol{y}\rangle^{B,UI} == (-1)\right)$.

19 : Each party locally set $\left\langle \boldsymbol{z_{0,\bar{1}_{(10)}}}\right\rangle^{B,UI} = \left(0, \bar{1}_{(10)}\right)$.

20 : Parties compute $\langle cond_{case5}\rangle^B = \left(\langle \boldsymbol{y}\rangle^{B,UI} < (-1)\right)$.

21 : Each party locally set $\left\langle \boldsymbol{z_{\bar{0}_{(11)}}}\right\rangle^{B,UI} = \left(\bar{0}_{(11)}\right)$.

22 : Parties compute $\langle \boldsymbol{d_{round}}\rangle^{B,UI}$, $\langle \boldsymbol{z_{round}}\rangle^{B,UI}$.

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

3. Multiply x'' by Λ Suppose we have:

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

$\Pi^{MulDoubleNPow2}\left(\langle \boldsymbol{d}\rangle^{B,UI}, \langle \boldsymbol{z}\rangle^{B,UI}, n\right)$ 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.

Protocol: $\Pi^{MulDoubleNPow2}\left(\langle \boldsymbol{d}\rangle^{B,UI}, \langle \boldsymbol{z}\rangle^{B,UI}, n\right)$

Input: $\langle \boldsymbol{d}\rangle^{B,UI} = \left(\langle d_1\rangle^B, \ldots, \langle d_{52}\rangle^B\right)$, $\langle \boldsymbol{z}\rangle^{B,UI} = \left(\langle z_1\rangle^B, \ldots, \langle z_{11}\rangle^B\right)$, n

Output: $\left\langle \boldsymbol{z_{add(n)}}\right\rangle^B$

1 : Parties compute $\left\langle cond_{\neq \pm 0}\right\rangle^B = \left(\vee_{j=1}^{52} \langle d_j\rangle^B\right) \vee \left(\vee_{j=1}^{11} \langle z_j\rangle^B\right)$.

2 : Parties compute $\left\langle \boldsymbol{z_{add(n)}}\right\rangle^{B,UI} = \left\langle cond_{\neq \pm 0}\right\rangle^B \wedge \left(\langle \boldsymbol{z}\rangle^{B,UI} + n\right)$.

Protocol 3.6: MPC protocol for multiplying x'' by Λ.

MPC-DP Protocol for Snapping Mechanism

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

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

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

Note:

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

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

Input: $\langle \boldsymbol{f\left(D\right)} \rangle^{B,FL}$,λ, B, n

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

1 : Parties run $\langle \boldsymbol{U^*} \rangle^{B,FL} = \Pi^{RandFloat1}$.

2 : Each party locally run $\langle S \rangle^{B} = \Pi^{RandBits}\left(1\right)$.

3 : Parties run $\left\langle \boldsymbol{f\left(D\right)_{clampB}} \right\rangle^{B,FL} = \Pi^{Clamp}\left(\langle \boldsymbol{f\left(D\right)} \rangle^{B,FL}, B\right)$.

4 : Parties compute $\left\langle \boldsymbol{Y_{LapNoise}} \right\rangle^{B,FL} = \lambda * \mathrm{LN}\left(\langle \boldsymbol{U^*} \rangle^{B,FL}\right)$.

5 : Parties compute $\left\langle S'_{Y_{LapNoise}} \right\rangle^{B} = \left\langle S_{Y_{LapNoise}} \right\rangle^{B} \wedge \langle S \rangle^{B}$, where $\left\langle S_{Y_{LapNoise}} \right\rangle^{B}$ is the sign bit of $\left\langle \boldsymbol{Y_{LapNoise}} \right\rangle^{B,FL}$.

6 : Each party locally set $\left\langle S'_{Y_{LapNoise}} \right\rangle^{B}$ as the sign bit of $\left\langle \boldsymbol{Y_{LapNoise}} \right\rangle^{B,FL}$.

7 : Parties compute $\langle \boldsymbol{x} \rangle^{B,FL} = \left\langle \boldsymbol{f\left(D\right)_{clampB}} \right\rangle^{B,FL} + \left\langle \boldsymbol{Y_{LapNoise}} \right\rangle^{B,FL}$.

8 : Parties run $\left\langle \boldsymbol{x'} \right\rangle^{B,FL} = \Pi^{XDivideLambda}\left(\langle \boldsymbol{z} \rangle^{B,UI}, n\right)$.

9 : Parties run $\left\langle \boldsymbol{x''} \right\rangle^{B,FL} = \Pi^{RoundDouble2Int}\left(\left\langle \boldsymbol{d'} \right\rangle^{B,UI}, \left\langle \boldsymbol{z'} \right\rangle^{B,UI}\right)$.

10 : Parties run $\left\langle \boldsymbol{x'''} \right\rangle^{B,FL} = \Pi^{MulDoubleNPow2}\left(\left\langle \boldsymbol{d''} \right\rangle^{B,UI}, \left\langle \boldsymbol{z''} \right\rangle^{B,UI}, n\right)$.

11 : Parties run $\langle \boldsymbol{x_{SM}} \rangle^{B,FL} = \Pi^{Clamp}\left(\left\langle \boldsymbol{x'''} \right\rangle^{B,FL}, B\right)$ and output $\langle \boldsymbol{x_{SM}} \rangle^{B,FL}$.

Protocol 3.7: MPC-DP protocol for snapping mechanism.

3.5 Differential Privacy Mechanism based on scaled Integer

In this section, we adapt the differentially privacy algorithms from [Tea20] and provide the correspond MPC-DP implementations.

As [Mir12] shows that the implementation of Laplacian mechanism with textbook noise generation method under floating-point arithmetic can lead to privacy violation. [Tea20] prevents this issue by developing a differential privacy mechanism use (scaled) integers to simulate continuous noise, which is defined as:

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

where i is an integer sampled from a probability distribution and scaled to simulate continuous noise with resolution parameter r. $f_r\left(D\right) \in \mathbb{D}$ is calculated by rounding the output of query function $f\left(D\right) \in \mathbb{R}$ to the nearest multiple of r. Note that in original paper [Tea20], the sampling algorithms of integer i is called Secure Noise Generation (SNG) algorithms.

The calculation of $M_{SNG}\left(f\left(D\right), r\right)$ can be executed *exactly* under floating-point arithmetic. *exactly* refers to that certain real numbers (such as integer i) can be represented without precision loss as floating numbers, and the arithmetic operations of those real numbers yield

the same result as in the floating-point arithmetic operations (when those real numbers are represented as floating-point numbers and execute the floating-point arithmetic operations). In addition, $M_{SNG}(f(D), r)$ can be proved to achieve differential privacy (similar to Laplacian and Gaussian mechanisms) by sampling i from specific probability distributions. Therefore, the implmentation of $M_{SNG}(f(D), r)$ under floating-point arithmetic is immune to the attack [Mir12] that is caused by the porous distribution and rounding effect of floating-point arithmetic.

Next, we describe how to implement $M_{SNG}(f(D), r)$ and sample integer i exactly under floating-point arithmetic. Recall a floating-point number d can be represented in IEEE-754 double-precision binary floating-point format as:

$$d_{IEEE-754} = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023},$$

where $(d_1, \dots, d_{52}) \in \{0,1\}^{52}$ are the significant bits, $(e_1 \dots e_{11}) \in \{0,1\}^{11}$ are the biased exponent bits and $S \in \{0,1\}$ is the sign bit.

Floating-point number d can also be reformulated by right-shifting the radix point 52 times as:

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

We use above $d_{Rshift(52)}$ floating-point number format to represent the terms in $M_{SNG}(f(D), r) = f_r(D) + ir$.

Let $i, t \in \left\{z \in \mathbb{Z} \mid |z| \le 2^{52} - 1\right\}$, $r = 2^{\left(r_{e_1} \dots r_{e_{11}}\right)_2 - 1023 - 52}$ and $\left(r_{e_1}, \dots, r_{e_{11}}\right) \in \{0,1\}^{11}$. We have $f_r(D) = tr$, where tr is the nearest multiple of r to $f(D)$. Then, $M_{SNG}(f(D), r)$ can be expressed with $d_{Rshift(52)}$ floating-point number format as:

$$\begin{aligned} M_{SNG}(f(D), r) &= f_r(D) + ir \\ &= tr + ir \\ &= \left(2^{52} + t\right) \cdot r + \left(2^{52} + i\right) \cdot r - 2^{52} \cdot r - 2^{52} \cdot r \end{aligned} \tag{3.9}$$

Note that

$$\begin{aligned} \left(2^{52} + t\right) \cdot r &= \left(2^{52} + t\right) \cdot 2^{\left(r_{e_1} \dots r_{e_{11}}\right)_2 - 1023 - 52} \\ &= \left(2^{52} + (t_1 \dots t_{52})_2\right) \cdot 2^{\left(r_{e_1} \dots r_{e_{11}}\right)_2 - 1023 - 52} \\ &= (1.t_1 \dots t_{52})_2 \cdot 2^{\left(r_{e_1} \dots r_{e_{11}}\right)_2 - 1023} \end{aligned} \tag{3.10}$$

Therefore, $\left(2^{52} + t\right) \cdot r$, $\left(2^{52} + i\right) \cdot r$ and $2^{52} \cdot r = 1 \cdot 2^{\left(r_{e_1} \dots r_{e_{11}}\right)_2 - 1023}$ can be represented exactly as floating-point numbers, and $M_{SNG}(f(D), r)$ can be calculated exactly under floating-point arithmetic. In this way, we scale the integer $i \in \left\{z \in \mathbb{Z} \mid |z| < 2^{52}\right\}$ into a floating-point number ir exactly.

In general, the calculation of $M_{SNG}(f(D), r)$ consists of three steps:

1. Set a resolution parameter $r = 2^k$, where $k \in [-1074 \dots 972]$.
2. Sample i from $Sampler(\epsilon, \delta, r, \Delta_r)$, which guarantees that $\Pr\left[|i| > 2^{52}\right] < \frac{1}{e^{(1000)}}$.
3. Calculate $f_r(D) + ir$ in floating-point arithmetic.

TODO: prove about $\Pr\left[|i| > 2^{52}\right] < \frac{1}{e^{(1000)}}$

Step 1, 2 guarantee that i, r and $f_r(D)$ can be represented exactly as floating-point numbers with very high probability (fails with probability $p < \frac{1}{e^{(1000)}}$). i is an integer generated by $Sampler(\epsilon, \delta, r, \Delta_r)$ and correspond to a probability distribution, where ϵ, δ are the parameters that controls the privacy protection level, and Δ_r is the sensitivity of $f_r(D)$. Similar to the sensitivity definition of $f(D)$, the sensitivity of $f_r(D)$ is defined as $\Delta_r = \max_{D,D'} \left\| f_r(D) - f_r\left(D'\right) \right\|$, where D and D' are neighboring databases.

3.5.1 Approximating Laplacian Mechanism

In this section, we use $M_{SNGLap}(f(D), r) = f_r(D) + ir$ [Tea20] (with i is sampled from a double-side geometric distribution $DGeo(p)$) to simulate Laplacian mechanism ($M_{Lap}(f(D)) = f(D) + Y$, $Y \sim Lap(b)$). Since the double-side geometric distribution is a discrete variate of Laplacian distribution, it is natural to use a scale geometric random variable ir to simulate the Laplacian random variable Y.

Specifically, integer i is sampled from a double-side geometric distribution $DGeo\left(p = 1 - e^{-\lambda}\right)$, where $\lambda = \frac{r\epsilon}{\Delta_r}$. r is the smallest power of 2 exceeding $\frac{\Delta}{2^c \epsilon}$ and c is a predefined integer that controls the degree of discretization and accuracy of $M_{SNGLap}(f(D), r)$.

TODO: how to choose parameter in practice, ranges, ???

Theorem 6 ([Tea20]). *The mechanism $M_{SNGLap}(f(D), r)$ satisfies ε-DP for query function f when r and i are generated as above.*

TODO: DP prove, properties.

In following part, we first introduce the algorithm for sampling geometric random variables § 3.5.1, then the algorithms for double-side geometric random variables § 3.5.1.

Geometric Distribution Sampling

$Algo^{GeometricExpBinarySearch}(\lambda)$ [Tea20] samples a positive integer x from geometric distribution $Geo\left(p = 1 - e^{-\lambda}\right)$:

$$\text{PMF: } \Pr\left(x \mid p = 1 - e^{-\lambda}\right) = (1-p)^{x-1}\, p = e^{-\lambda(x-1)} \cdot \left(1 - e^{-\lambda}\right)$$
$$\text{CDF: } \Pr\left(x \leq X \mid p = 1 - e^{-\lambda}\right) = 1 - (1-p)^{X} = 1 - e^{-\lambda X}$$

The sampling interval of x is $(0 \ldots Int_{max52}]$, where $Int_{max52} = \left(\overline{0}_{(12)}\overline{1}_{(52)}\right)_2 = 4503599627370495 = 4.503599627370495 \times 10^{15}$ is the largest signed 64-bit integer that we can represent in floating-point exactly. For the cumulative distribution function (CDF) with $X = Int_{max52}$, we have

$$\text{CDF} : \Pr(x \leq Int_{max52}) = 1 - e^{-\lambda Int_{max52}} = \begin{cases} 0.\overline{9}_{(27)}8396\ldots & \text{for } \lambda = 2^{-46} \\ 0.\overline{9}_{(13)}8733\ldots & \text{for } \lambda = 2^{-47} \\ 0.\overline{9}_{(6)}8874\ldots & \text{for } \lambda = 2^{-48} \end{cases} \tag{3.11}$$

The support of geometric distribution $Geo(p)$ is $supp(Geo(p)) = \{z \in \mathbb{Z} \mid z > 0\}$ that covers all the positive integers. However, we can only sample integers from interval $(0 \ldots Int_{max52}]$ instead of $(0 \ldots +\infty)$, which requires that the difference regarding CDF between $\Pr(x \leq Int_{max52})$ and $\Pr(x \leq +\infty) = 1$ should be small enough (for $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$). Therefore, we set $\lambda \geq 2^{-48}$ to ensure $\Pr(x \leq Int_{max52}) \geq 0.\overline{9}_{(6)}8874\ldots$.

Generally, $Algo^{GeometricExpBinarySearch}(\lambda)$ samples an integer x by splitting the sampling interval $(L \ldots R]$ into two subintervals (i.e., $(L \ldots M]$ and $(M \ldots R]$), such that they have approximate equal probability mass ($\approx \frac{1}{2}$) with a middle point M as line 3 shows. Line $4-7$ ensures that the middle point M lies in interval $(L \ldots R]$, and reset M into $(L \ldots R]$ otherwise. Then we calculate the probability mass proportion Q between the and randomly choose one (either $(L \ldots M]$ or $(M \ldots R]$) based on the generated uniform variable U^* and Q. Next, we repeat this process until the remaining interval only contains one value as line 2, and that value R is the random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ we are looking for.

In $Algo^{GeometricExpBinarySearch}(\lambda)$, function $\text{Split}(L, R, \lambda) = L - \text{int}\left(\frac{\ln(0.5) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda}\right)$ calculates the middle point M of interval $(L \ldots R]$ such that

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

where

$$\begin{aligned}\Pr(L < x \leq M) &= \Pr(x \leq M) - \Pr(x \leq L) \\ &= \left(1-(1-p)^M\right) - \left(1-(1-p)^L\right) \\ &= \left(1-e^{-\lambda M}\right) - \left(1-e^{-\lambda L}\right) \\ &= e^{-\lambda L} - e^{-\lambda M} \\ &= \frac{1}{e^{\lambda L}} - \frac{1}{e^{\lambda M}} \\ &= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda L + \lambda M}},\end{aligned} \tag{3.12}$$

$$\begin{aligned}\Pr(L < x \leq R) &= \Pr(x \leq R) - \Pr(x \leq L) \\ &= \dots \\ &= \frac{e^{\lambda R} - e^{\lambda L}}{e^{\lambda L + \lambda R}}.\end{aligned} \tag{3.13}$$

Then we have

$$\begin{aligned}\Pr(L < x \leq M \mid L < x \leq R) &= \frac{\Pr(L < x \leq M)}{\Pr(L < x \leq R)} \\ &= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda L + \lambda M}} \cdot \frac{e^{\lambda L + \lambda R}}{e^{\lambda R} - e^{\lambda L}} \\ &= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda R} - e^{\lambda L}} \cdot e^{\lambda (R-M)} \\ &= \frac{e^{\lambda R} - e^{(L+R-M)}}{e^{\lambda R} - e^{\lambda L}}\end{aligned} \tag{3.14}$$

Next, we have

$$
\begin{aligned}
\Pr(L < x \le M \mid L < x \le R) &\approx \frac{1}{2} \\
\frac{e^{\lambda R} - e^{\lambda(L+R-M)}}{e^{\lambda R} - e^{\lambda L}} &\approx \frac{1}{2} \\
e^{\lambda R} - e^{\lambda(L+R-M)} &\approx \frac{1}{2}\left(e^{\lambda R} - e^{\lambda L}\right) \\
\frac{1}{2}\left(e^{\lambda R} + e^{\lambda L}\right) &\approx e^{\lambda(L+R-M)} \\
\ln\left(\frac{1}{2}\left(e^{\lambda R} + e^{\lambda L}\right)\right) &\approx \ln\left(e^{\lambda(L+R-M)}\right) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) &\approx \lambda(L+R-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) &\approx \ln\left(e^{\lambda R}\right) + \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) - \ln\left(e^{\lambda R}\right) &\approx \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(\frac{e^{\lambda R} + e^{\lambda L}}{e^{\lambda R}}\right) &\approx \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right) &\approx \lambda(L-M) \\
\frac{\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda} &\approx L - M \\
L - \frac{\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda} &\approx M
\end{aligned}
\tag{3.15}
$$

Finally, we have $M = L - \mathrm{int}\left(\frac{\ln\left(\frac{1}{2}\right)+\ln\left(1+e^{-\lambda(R-L)}\right)}{\lambda}\right)$, where $\mathrm{int}(\cdot)$ rounds the input to a 64-bit signed integer. Note that the calculation of M should take into account about the underflow and overflow effect of $\ln(\cdot)$ and $e^{(\cdot)}$ under floating-point implementation.

TODO: discuss about ln and exp in double precision floating-point number

Function $\mathrm{Proportion}(L, R, M, \lambda) = \frac{e^{-\lambda(M-L)}-1}{e^{-\lambda(R-L)}-1}$ calculates the proportion Q (Q should be approximate to $\frac{1}{2}$) of $\Pr(L < x \le M)$ and $\Pr(L < x \le R)$, where

$$
\begin{aligned}
Q &= \Pr(L < x \leq M \mid L < x \leq R) \\
&= \frac{\Pr(L < x \leq M)}{\Pr(L < x \leq R)} \\
&= \dots \\
&= \frac{e^{-\lambda(M-L)} - 1}{e^{-\lambda(R-L)} - 1}
\end{aligned}
\tag{3.16}
$$

Then, based on Q and the generated random floating-point number $U^* \in \mathbb{D} \cap (0,1)$, either interval $(L \dots M]$ or $(M \dots R]$ will be chosen. Specifically, suppose $\Pr(L < x \leq M) = l$ and $\Pr(M < x \leq R) = r$, we have $Q = \frac{l}{l+r}$ and choose $(L \dots M]$ if $U^* \leq Q$, $(M \dots R]$ otherwise. In other words, the probability that a interval is chosen is proportional to its probability mass.

Algorithm: $Algo^{GeometricExpBinarySearch}(\lambda)$

Input: λ

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

1 : $L \leftarrow 0, R \leftarrow Int_{max52}$
2 : **FOR** $L + 1 < R$
3 : $\quad M \leftarrow \text{Split}(L, R, \lambda)$
4 : $\quad$ **IF** $M \leq L$
5 : $\quad\quad M = L + 1$
6 : $\quad$ **ELSE IF** $M \geq R$
7 : $\quad\quad M = R - 1$
8 : $\quad Q = \text{Proportion}(L, R, M, \lambda)$
9 : $\quad U^* \leftarrow Algo^{RandFloat1}$
10 : $\quad$ **IF** $U^* \leq Q$
11 : $\quad\quad R \leftarrow M$
12 : $\quad$ **ELSE**
13 : $\quad\quad L \leftarrow M$
14 : **RETURN** $x \leftarrow R$

Algorithm 3.4: Algorithm for geometric distribution $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$.

To convert $Algo^{GeometricExpBinarySearch}(\lambda)$ into MPC protocol $\Pi^{GeometricExpBinarySearch}(\lambda)$, we use technique § 3.3 to transform the **IF** branchings into MPC computations. For the FOR loop, we assume that $Algo^{GeometricExpBinarySearch}(\lambda)$ iterates less than $iter$ times and return the result. In line 1−4, each party locally calculates M_0 and Q_0. In line 5, the parties generate U^* and finish the first iteration for binary search in line 10. In line 6−9, the parties transform the **IF** branchings into MPC computations and choose one subinterval (either $(L_0 \dots M_0]$ or $(M_0 \dots R_0]$). In line 10, the parties set the $flag_0$ to record if the FOR loop condition

is satisfied. In line $11-23$, the parties convert the binary search in MPC computations and run that for $iter-1$ times. In line 12, the parties compute the middle point M_j with $\Pi^{Split}\left(\langle \boldsymbol{L}\rangle^{B,UI}, \langle \boldsymbol{R}\rangle^{B,UI}, \lambda\right)$ Prot. A.12. In line 12, the parties calculate M_j In line $13-16$, the parties recalculate M_j it if $M_j \notin (L_{j-1} \dots R_{j-1})$, or keep it unchanged otherwise:

$$\begin{aligned} \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} = & \left\langle cond_{M_j \leq L_{j-1}} \right\rangle^{B} \wedge \left(\left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI} + 1 \right) \\ & \oplus \left\langle cond_{M_j \geq R_{j-1}} \right\rangle^{B} \wedge \left(\left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI} - 1 \right) \\ & \oplus \left\langle cond_{L_{j-1} < M_j < R_{j-1}} \right\rangle^{B} \wedge \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \end{aligned} \tag{3.17}$$

In line 17—22, the parties compute the proportion Q_j with $\Pi^{Proportion}\left(\left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI}, \left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI}, \left\langle \boldsymbol{M_j} \right\rangle^{B,UI}, \lambda\right)$ Prot. A and choose one subinterval (either $\left(L_j \dots M_j\right]$ or $\left(M_j \dots R_j\right]$) for further operations. In line 23, the parties compute $flag_j$ to record if the FOR loop condition is satisfied. In line 24, 26, the parties calculate the position $p_0, \dots, p_{iter-1}$ where $p_k = 1$ and $k \in [0 \dots iter-1]$ is the smallest number such that $flag_k = 1$, i.e., the FOR loop condition in $Algo^{GeometricExpBinarySearch}(\lambda)$ is not satisfied and the algorithm returns the result $x = R_k$. In line 27, the parties extract the correct result from $(R_0, \dots, R_{iter-1})$.

Protocol: $\Pi^{GeometricExpBinarySearch}(\lambda)$

Input: λ

Output: $\langle x \rangle^{B,UI}$, **where** $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$

Each party locally set $L_0 = 0, R_0 = Int_{max52}$.

Each party locally computes $M_0 = \text{Split}(L_0, R_0, \lambda)$.

Each party locally set $M_0 = L_0 + 1$ **if** $M_0 \leq L_0$, $M_0 = R_0 - 1$ **if** $M_0 \geq R_0$.

Each party locally computes $Q_0 = \text{Proportion}(L_0, R_0, M_0, \lambda)$.

Parties execute $\Pi^{RandFloat1}$ and obtain $\left\langle \boldsymbol{U_0^*} \right\rangle^{B,FL}$.

Parties compute $\left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} = \left(\left\langle \boldsymbol{U_0^*} \right\rangle^{B,FL} \leq Q_0 \right)$.

Parties compute $\left\langle cond_{U_0 > Q_0} \right\rangle^{B} = \text{NOT}\left(\left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \right)$.

Parties compute $\langle \boldsymbol{R_0} \rangle^{B,UI} = \left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \wedge M_0 \oplus \left\langle cond_{U_0 > Q_0} \right\rangle^{B} \wedge R_0$.

Parties compute $\langle \boldsymbol{L_0} \rangle^{B,UI} = \left\langle cond_{U_0 > Q_0} \right\rangle^{B} \wedge M_0 \oplus \left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \wedge L_0$.

Parties compute $\langle flag_0 \rangle^{B} = \left(\langle \boldsymbol{L_0} \rangle^{B,UI} + 1 \geq \langle \boldsymbol{R_0} \rangle^{B,UI} \right)$.

FOR $j = 1$ **TO** $iter - 1$

  Parties compute $\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} = \Pi^{Split}\left(\left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI}, \left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI}, \lambda \right)$.

  Parties compute $\left\langle cond_{M_j \leq L_{j-1}} \right\rangle^{B} = \left(\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \leq \left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI} \right)$.

  Parties compute $\left\langle cond_{M_j \geq R_{j-1}} \right\rangle^{B} = \left(\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \geq \left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI} \right)$.

  Parties compute $\left\langle cond_{L_{j-1} < M_j < R_{j-1}} \right\rangle^{B} = \text{NOT}\left(\left\langle cond_{M_j \leq L_{j-1}} \right\rangle^{B} \vee \left\langle cond_{M_j \geq R_{j-1}} \right\rangle^{B} \right)$.

  Parties compute $\left\langle \boldsymbol{M_j} \right\rangle^{B,UI}$ as Eq. (3.17).

  Parties run $\left\langle \boldsymbol{Q_j} \right\rangle^{B,FL} = \Pi^{Proportion}\left(\left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI}, \left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI}, \left\langle \boldsymbol{M_j} \right\rangle^{B,UI}, \lambda \right)$.

  Partie run $\left\langle \boldsymbol{U_j^*} \right\rangle^{B,FL} = \Pi^{RandFloat1}$.

  Parties copmute $\left\langle cond_{U_j \leq Q_j} \right\rangle^{B} = \left(\left\langle \boldsymbol{U_j^*} \right\rangle^{B,FL} \leq \left\langle \boldsymbol{Q_j} \right\rangle^{B,FL} \right)$.

  Parties compute $\left\langle cond_{U_j > Q_j} \right\rangle^{B} = \text{NOT}\left(\left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \right)$.

  Parties compute $\left\langle \boldsymbol{R_j} \right\rangle^{B,UI} = \left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \wedge \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \oplus \left\langle cond_{U_j > Q_j} \right\rangle^{B} \wedge \left\langle \boldsymbol{R_j} \right\rangle^{B,UI}$.

  Parties compute $\left\langle \boldsymbol{L_j} \right\rangle^{B,UI} = \left\langle cond_{U_j > Q_j} \right\rangle^{B} \wedge \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \oplus \left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \wedge \left\langle \boldsymbol{L_j} \right\rangle^{B,UI}$.

  Parties compute $\left\langle flag_j \right\rangle^{B} = \left(\left\langle \boldsymbol{L_j} \right\rangle^{B,UI} + 1 \geq \left\langle \boldsymbol{R_j} \right\rangle^{B,UI} \right)$.

Parties compute $\left(\langle e_0 \rangle^{B}, \ldots, \langle e_{iter-1} \rangle^{B} \right) = \Pi^{PreOr}\left(\langle flag_0 \rangle^{B}, \ldots, \langle flag_{iter-1} \rangle^{B} \right)$.

FOR $j = 1$ **TO** $iter - 1$

  Each party locally computes $\left\langle p_j \right\rangle^{B} = \left\langle e_j \right\rangle^{B} \oplus \left\langle e_{j-1} \right\rangle^{B}$, where $\langle p_0 \rangle^{B} = \langle e_0 \rangle^{B}$.

Parties compute $\langle x \rangle^{B,UI} = \oplus_{j=0}^{iter-1} \left\langle p_j \right\rangle^{B} \wedge \left\langle \boldsymbol{R_j} \right\rangle^{B,UI}$.

Protocol 3.8: MPC protocol for geometric distribution $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$.

Double-side Geometric Distribution Sampling

In this part we describe how to sample $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$ based on $Geo\left(p\right)$.

Recall that geometric distribution $Geo\left(p = 1 - e^{-\lambda}\right)$:

$$\begin{aligned} \text{PMF: } \Pr\left(x \,|\, p = 1 - e^{-\lambda}\right) &= (1-p)^{x-1}\, p = e^{-\lambda(x-1)} \cdot \left(1 - e^{-\lambda}\right), \\ \text{Support}: supp\left(Geo\left(p\right)\right) &= \{z \in \mathbb{Z} \,|\, z > 0\}. \end{aligned} \tag{3.18}$$

The double-side geometric distribution $DGeo\left(p = 1 - e^{-\lambda}\right)$:

$$\begin{aligned} \text{PMF: } \Pr\left(x \,|\, p = 1 - e^{-\lambda}\right) &= \frac{(1-p)^{|x|}\, p}{2-p} = \frac{e^{-\lambda|x|} \cdot \left(1 - e^{-\lambda}\right)}{1 + e^{\lambda}}, \\ \text{Support: } supp\left(DGeo\left(p\right)\right) &= \{z \in \mathbb{Z}\}. \end{aligned} \tag{3.19}$$

$DGeo\left(p\right)$ can be generated by left-shifting geometric distribution $Geo\left(p\right)$ along $-x$-axis by 1, mirroring along the y-axis and scaled such that for $x \sim DGeo\left(p\right)$, $\Pr\left(-\infty < x < +\infty \,|\, p\right) = 1$.

As $Algo^{DGeometric}\left(\lambda\right)$ shows, we generate a double-side geometric random variable $s \cdot g$ by first setting the sign $s = -1$ and number part $g = 0$. Then in the **WHILE** loop, we keep regenerating the sign s and random variable g until $s == -1 \wedge g == 0$. In line 3, $g \leftarrow Algo^{GeometricExpBinarySearch}\left(\lambda\right) - 1$ because we need a left-shifted geometric distribution $Geo_{left-shift}\left(p\right)$ with support $supp\left(Geo_{left-shift}\left(p\right)\right) = \{z \in \mathbb{N}\}$. Then when sign $s = -1$, we mirror the left-shifted geometric distribution along the y-axis. In line 2, for $g == 0$, we reject the random variable $s \cdot g = -1 \cdot 0$ and accept only $s \cdot g = +1 \cdot 0$, otherwise we would generate value $s \cdot g = 0$ double times.

Algorithm: $Algo^{DGeometric}\left(\lambda\right)$

Input: λ

Output: $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$

1: $s \leftarrow -1$, $g \leftarrow 0$
2: **WHILE** $s == -1$ **AND** $g == 0$
3: $\quad g \leftarrow Algo^{GeometricExpBinarySearch}\left(\lambda\right) - 1$
4: $\quad s \leftarrow 2 * Bern(0.5) - 1$
5: **RETURN** $x \leftarrow s \cdot g$

Algorithm 3.5: Algorithm for double-side geometric distribution $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$.

To convert $Algo^{DGeometric}\left(\lambda\right)$ into MPC protocol $\Pi^{DGeometric}\left(\lambda\right)$, we assume that the **WHILE** loop termintes in less than $iter$ iterations. In line 2, 3, the parties generate the sign s_j and

number part g_j of double-side geometric random variable $s_j \cdot g_j$. In line 4, the parties compute the $flag_j$ to record if the WHILE loop condition is satisfied.

In line 5, the parties calculate the position $p_0, \ldots, p_{iter-1}$ where $p_k = 1$ and $k \in [0 \ldots iter-1]$ is the smallest number such that $flag_k = 1$, i.e., the WHILE loop condition in $Algo^{DGeometric}(\lambda)$ is not satisfied and the algorithm returns the result s_k and g_k. In line 8, 9, the parties extract the correct sign s and number part g from $iter$ pair of values.

Protocol: $\Pi^{DGeometric}(\lambda)$

Input: None

Output: $\langle s \rangle^B$ **and** $\langle g \rangle^{B,UI}$, **where** $x = (-1)^s \cdot g$ **and** $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$

1 : **FOR** $j = 0$ **TO** $j = iter - 1$

2 : Parties compute $\left\langle b_j \right\rangle^B = \Pi^{Bern}(0.5)$.

3 : Parties compute $\left\langle g_j \right\rangle^{B,UI} = \Pi^{GeometricExpBinarySearch}(\lambda) - 1$.

4 : $\left\langle flag_j \right\rangle^B = \text{NOT}\left(\left(\left\langle s_j \right\rangle^B == -1\right) \wedge \left(\left\langle g_j \right\rangle^{B,UI} == 0\right)\right)$

5 : Parties compute $\left(\langle e_0 \rangle^B, \ldots, \langle e_{iter-1} \rangle^B\right) = \Pi^{PreOr}\left(\langle flag_0 \rangle^B, \ldots, \langle flag_{iter-1} \rangle^B\right)$.

6 : **FOR** $j = 1$ **TO** $iter - 1$

7 : Each party locally computes $\left\langle p_j \right\rangle^B = \left\langle e_j \right\rangle^B \oplus \left\langle e_{j-1} \right\rangle^B$, where $\langle p_0 \rangle^B = \langle e_0 \rangle^B$.

8 : Parties compute $\langle s \rangle^B = \oplus_{j=0}^{iter-1} \left\langle p_j \right\rangle^B \wedge \left\langle s_j \right\rangle^B$.

9 : Parties compute $\langle g \rangle^{B,UI} = \oplus_{j=0}^{iter-1} \left\langle p_j \right\rangle^B \wedge \left\langle g_j \right\rangle^{B,UI}$.

Protocol 3.9: MPC Protocol for double-side geometric distribution $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$.

MPC-DP Protocol for SNG Laplacian Mechanism

Recall mechanism $M_{SNGLap}(f_r(D), r)$ § 3.5.1 which approximates the Laplacian mechanism is defined as:

$$\begin{aligned} M_{SNGLap}(f_r(D), r) &= f_r(D) + ir \\ &= f_r(D) + \left(2^{52} + i\right) \cdot r - 2^{52} \cdot r, \end{aligned} \tag{3.20}$$

where $i \sim DGeo\left(p = 1 - e^{-\lambda}\right)$ and $\lambda = \frac{r\epsilon}{\Delta_r}$.

In our MPC-DP general framework Prot. 2.1, it can be reformulated as:

$$M_{SNGLap}(\langle f_r(D) \rangle) = \langle f_r(D) \rangle + \left(2^{52} + \langle i \rangle\right) \cdot r - 2^{52} \cdot r,$$

where r, ϵ, Δ_r, λ are public known.

We present the MPC-DP protocol $\Pi^{SNGLap}\left(\langle \boldsymbol{f_r}(\boldsymbol{D})\rangle^{B,FL}, \lambda\right)$ for $M_{SNGLap}\left(\langle f_r(D)\rangle, r\right)$. We assume the parties have already computed $\langle \boldsymbol{f_r}(\boldsymbol{D})\rangle^{B,FL}$ and focus on generating the integer share $\langle \boldsymbol{i}\rangle^{B,UI}$.

Note:

1. The resolution parameter $r = 2^k$, where $k \in [-1074 \dots 972]$.
2. $f_r(D)$ is a multiple of r and can be represent exactly as floating-point number.
3. In line 2, both the calculation $i + 2^{52}$ and converion UI2FL$\left(i + 2^{52}\right)$ (from unsigned integer to floating-point number) is exact when $i \leq 2^{52} - 1$.
4. i_{float} can be represented in floating-point format as: $i_{float} = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$, where $(d_1 \dots d_{52})_2 = i$.
5. In line 3, the calculation of $i_{float} * r$ can be done by adding k (because $r = 2^k$) to the the exponent of i_{float}, which is exact under floating-point arithmetic.

Protocol: $\Pi^{SNGLap}\left(\langle \boldsymbol{f_r}(\boldsymbol{D})\rangle, \lambda\right)$

Input: $\langle \boldsymbol{f_r}(\boldsymbol{D})\rangle^{B,FL}, \lambda$

Output: $\left\langle \boldsymbol{x_{SNGLap}}\right\rangle^{B,FL}$

1 : Parties run $\langle \boldsymbol{i}\rangle^{B,UI} = \Pi^{DGeometric}(\lambda)$.

2 : Parties compute $\left\langle \boldsymbol{i_{float}}\right\rangle^{B,FL} = \text{UI2FL}\left(\langle \boldsymbol{i}\rangle^{B,UI} + 2^{52}\right)$.

3 : Parties compute $\left\langle \boldsymbol{Y_{LapNoise}}\right\rangle^{B,FL} = \left\langle \boldsymbol{i_{float}}\right\rangle^{B,FL} * r - 2^{52} * r$.

4 : Parties compute $\left\langle \boldsymbol{x_{SNGLap}}\right\rangle^{B,FL} = \langle \boldsymbol{f_r}(\boldsymbol{D})\rangle^{B,FL} + \left\langle \boldsymbol{Y_{LapNoise}}\right\rangle^{B,FL}$.

Protocol 3.10: MPC-DP protocol for SNG Laplacian Mechanism.

3.5.2 Approximating Gaussian Mechanism

In this section, we use $M_{SNGGauss}\left(f(D), r\right) = f_r(D) + ir$ [Tea20] (with i is sampled from a binomial distribution $Bino(n, p)$) to simulate Gaussian mechanism ($M_{Gauss}\left(f(D)\right) = f(D) + Y$, $Y \sim \mathcal{N}\left(\mu, \sigma^2\right)$).

To be more specific, we sample i from a binomial distribution $Bino(n, p)$ (PDF:$\Pr(x \mid n, p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$, where $x \in \mathbb{R}$ and $n \approx 2^{96}$) instead of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ (PMF: $\Pr(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$, where $x \in \mathbb{N}$).

TODO: Prove why binomial can approximate gaussian distribution, DP guarantee

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. Suppose $x \sim$

$Bino(n_x, p)$ and $y \sim Bino(n_y, p)$, then we have that $x+y$ is also a binomial random variable sampled from $Bino(n_x + n_y, p)$ [Dev86, Lemma 4.3]. Note that x, y can be generated by flipping n_x and n_y same and biased coins independently and count the numbers of $head$. However, for very large n (e.g., $n \approx 2^{96}$), this combination method would be inefficient. A rejection based binomial sampling algorithm [BKP+14] is deployed and modified.

TODO: how to modify exactly, origin algorithms, prove

After modification, we have $Algo^{Binomial}(\sqrt{n})$ [BKP+14; Tea20] which samples $i \sim Bino(n, p = 0.5)$ with input $\sqrt{n}$ for $n \leq 2^{96}$. One advantage of $Algo^{Binomial}(\sqrt{n})$ is that it only operates on $\sqrt{n}$ instead of n. Since $n \leq 2^{96}$, $\sqrt{n} \leq 2^{48}$ can be represented exactly as 64-bit integer under floating-point implementation.

In $Algo^{Binomial}(\sqrt{n})$, $Algo^{RandInt}(m)$ [BK15] generates a uniform random integer $l \in [0 \ldots m)$ and $Algo^{Bernoulli}\left(\frac{\tilde{p}(i)}{f}\right)$ generates a Bernoulli random variable $c \sim Bern\left(p = \frac{\tilde{p}(i)}{f}\right)$.

Algorithm: $Algo^{Binomial}\left(\sqrt{n} \approx 2^{48}\right)$

Input: n

Output: $x \sim Bino\,(n, p = 0.5)$

$m \leftarrow \left\lfloor \sqrt{2} * \sqrt{n} + 1 \right\rfloor$

WHILE TRUE

  $s \leftarrow Algo^{Geometric}$

  $U \leftarrow Algo^{RandFloat1}$

  IF $U < 0.5$

    $k \leftarrow s$

  ELSE

    $k \leftarrow -s - 1$

  $l \leftarrow Algo^{RandInt}\,(m)$

  $x \leftarrow km + l$

  IF $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2}$

    $\tilde{p}\,(x) = \sqrt{\frac{2}{\pi n}} * e^{-\frac{2x^2}{n}} * (1 - \nu_n)$, where $\nu_n = \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}$

    IF $\tilde{p}\,(x) > 0$

      $f \leftarrow \frac{4}{m * 2^s}$

      $c \leftarrow Algo^{Bernoulli}\left(\frac{\tilde{p}\,(x)}{f}\right)$

      IF $c == 0$

        RETURN x

Algorithm 3.6: Algorithm for binomial distribution $x \sim Bino\left(n \approx 2^{96}, p = 0.5\right)$.

Next, we convert $Algo^{Binomial}\left(\sqrt{n}\right)$ into MPC protocol $\Pi^{Binomial}\left(\sqrt{n}, m, x_{min}, x_{max}, \nu_n, \tilde{p}_{coe}\right)$ [Tea20]. First, we assume the **WHILE** loop in $Algo^{Binomial}\left(\sqrt{n}\right)$ terminates after $iter$ iterations.

Before the MPC computations, each party first locally compute following parameters:

$$\begin{aligned} m &= \lfloor \sqrt{2} * \sqrt{n} + 1 \rfloor, \\ x_{min} &= -\frac{\sqrt{n \ln n}}{2}, \\ x_{max} &= \frac{\sqrt{n \ln n}}{2}, \\ \nu_n &= \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}, \\ \tilde{p}_{coe} &= \sqrt{\frac{2}{\pi n}} * (1 - \nu_n). \end{aligned} \tag{3.21}$$

In line $1-21$, the parties run the opeartions in **WHILE** loop for $iter$ times. In line $2-4$, the parties sample a geometric random variable $s \sim Geo(p = 0.5)$ and convert it into signed integer and floating-point number for further opeartions. In line 8, the parties compute k based on the random floating-point number U^*. In line 11, the parties convert x into floating-point number for further opeartions. In line 18, the parties compute the parameter $p_{Bernoulli}$ and sample a Bernoulli random variable $c_j \sim Bern(p = p_{Bernoulli})$. In line 21, the parties compute the condition $cond_{j,total}$ to record if the **WHILE** loop condition is satisfied.

In line $22-24$, the parties calculate the position $p_0, \ldots, p_{iter-1}$ where $p_k = 1$ and $k \in [0 \ldots iter-1]$ is the smallest number such that $cond_{k,total} = 1$, i.e., the **WHILE** loop condition in $Algo^{Binomial}\left(\sqrt{n}\right)$ is not satisfied and the algorithm returns the result x. In line 25, the parties extract the correct result from $(i_0, \ldots, i_{iter-1})$.

Protocol: $\Pi^{Binomial}\left(\sqrt{n}, m, x_{min}, x_{max}, \nu_n, \tilde{p}_{coe}\right)$

Input: $\sqrt{n}$, m, x_{min}, x_{max}, ν_n, $\tilde{p}_{coe}$

Output: $\langle \boldsymbol{x} \rangle^{B,UI} \sim Bino\,(n, p = 0.5)$

1 : **FOR** $j = 0$ TO $iter - 1$

2 : Parties run $\left\langle \boldsymbol{s_j} \right\rangle^{B,UI} = \Pi^{Geometric}$

3 : Parties run $\left\langle \boldsymbol{s_j} \right\rangle^{B,SI} = \mathrm{UI2SI}\left(\left\langle \boldsymbol{s_j} \right\rangle^{B,UI}\right)$

4 : Parties run $\left\langle \boldsymbol{s_j} \right\rangle^{B,FL} = \mathrm{SI2FL}\left(\left\langle \boldsymbol{s_j} \right\rangle^{B,SI}\right)$

5 : Parties compute $\left\langle \boldsymbol{s'_j} \right\rangle^{B,SI} = -1 * \left(\left\langle \boldsymbol{s_j} \right\rangle^{B,SI} + 1\right)$

6 : Parties run $\left\langle \boldsymbol{U_j^*} \right\rangle^{B,FL} = \Pi^{RandFloat1}$

7 : Parties compute $\left\langle cond_{U_j^*<0.5} \right\rangle^{B} = \left(0.5 < \left\langle \boldsymbol{U_j^*} \right\rangle^{B,FL}\right)$

8 : Parties compute $\left\langle \boldsymbol{k_j} \right\rangle^{B,SI} = \left\langle \boldsymbol{s_j} \right\rangle^{B,SI} \wedge \left\langle cond_{U_j^*<0.5} \right\rangle^{B} \oplus \left\langle \boldsymbol{s'_j} \right\rangle^{B,SI} \wedge \mathrm{NOT}\left(\left\langle cond_{U_j^*<0.5} \right\rangle^{B}\right)$

9 : Parties run $\left\langle \boldsymbol{l_j} \right\rangle^{B,SI} = \Pi^{RandInt}\,(m)$

10 : Parties compute $\left\langle \boldsymbol{x_j} \right\rangle^{B,SI} = \left\langle \boldsymbol{k_j} \right\rangle^{B,SI} * m + \left\langle \boldsymbol{l_j} \right\rangle^{B,SI}$

11 : Parties rin $\left\langle \boldsymbol{x_j} \right\rangle^{B,FL} = \mathrm{SI2FL}\left(\left\langle \boldsymbol{x_j} \right\rangle^{B,SI}\right)$

12 : Parties compute $\left\langle cond_{x_{min} \le x_j \le x_{max}} \right\rangle^{B} = \left(\left\langle \boldsymbol{x_j} \right\rangle^{B,SI} \ge x_{min}\right) \wedge \left(\left\langle \boldsymbol{x_j} \right\rangle^{B,SI} \le x_{max}\right)$

13 : Parties compute $\left\langle \boldsymbol{t_{-2x_j^2/n}} \right\rangle^{B,FL} = \left(-2 * \left(\frac{\left\langle \boldsymbol{x_j} \right\rangle^{B,FL}}{\sqrt{n}}\right)^2, -1\right).$

14 : Parties compute $\left\langle \boldsymbol{t_{\exp\left(-2x_j^2/n\right)}} \right\rangle^{B,FL} = \mathrm{EXP}\left(\left\langle \boldsymbol{t_{-2x_j^2/n}} \right\rangle^{B,FL}\right).$

15 : Parties compute $\left\langle \boldsymbol{\tilde{p}_j} \right\rangle^{B,FL} = \tilde{p}_{coe} * \left\langle \boldsymbol{t_{\exp\left(-2x_j^2/n\right)}} \right\rangle^{B,FL}.$

16 : Parties compute $\left\langle cond_{\tilde{p}_j>0} \right\rangle^{B} = \left(\left\langle \boldsymbol{\tilde{p}_j} \right\rangle^{B,FL} > 0\right)$

17 : Parties compute $\left\langle \boldsymbol{f_j} \right\rangle^{B,FL} = m * \mathrm{POW2}\left(\left\langle \boldsymbol{s_j} \right\rangle^{B,FL}\right)$

18 : Parties compute $\langle \boldsymbol{p_{Bernoulli}} \rangle^{B,FL} = \frac{\left\langle \boldsymbol{\tilde{p}_j} \right\rangle^{B,FL}}{\left\langle \boldsymbol{f_j} \right\rangle^{B,FL}}$

19 : Parties run $\left\langle c_j \right\rangle^{B} = \Pi^{Bernoulli}\left(\langle \boldsymbol{p_{Bernoulli}} \rangle^{B,FL}\right)$

20 : Parties compute $\left\langle cond_{c_j==0} \right\rangle^{B} = \left(\left\langle c_j \right\rangle^{B} == 0\right)$

21 : Parties compute $\left\langle cond_{j,total} \right\rangle^{B} = \left\langle cond_{x_{min} \le x_j \le x_{max}} \right\rangle^{B} \wedge \left\langle cond_{\tilde{p}_j>0} \right\rangle^{B} \wedge \left\langle cond_{c_j==0} \right\rangle^{B}$

22 : Parties compute $\left(\langle e_0 \rangle^{B}, \ldots, \langle e_{iter-1} \rangle^{B}\right) = \Pi^{PreOr}\left(\left\langle cond_{0,total} \right\rangle^{B}, \ldots, \left\langle cond_{iter-1,total} \right\rangle^{B}\right).$

23 : **FOR** $j = 0$ TO $iter - 1$

24 : Each party locally compute $\left\langle p_j \right\rangle^{B} = \left\langle e_j \right\rangle^{B} \oplus \left\langle e_{j-1} \right\rangle^{B}$, where $\langle p_0 \rangle^{B} = \langle e_0 \rangle^{B}$.

25 : Parties compute $\langle x \rangle^{B,SI} = \oplus_{j=0}^{iter-1} \left\langle p_j \right\rangle^{B} \wedge \left\langle \boldsymbol{x_j} \right\rangle^{B,SI}.$

Protocol 3.11: MPC protocol for binomial distribution $x \sim Bino\,(n, p = 0.5)$.

MPC-DP Protocol for SNG Gaussian Mechanism

Recall mechanism $M_{SNGGauss}(f_r(D), r)$ § 3.5.2 which approximates the Gaussian mechanism is defined as:

$$\begin{aligned} M_{SNGGauss}(f_r(D), r) &= f_r(D) + ir \\ &= f_r(D) + \left(2^{52} + i\right) \cdot r - 2^{52} \cdot r, \end{aligned} \tag{3.22}$$

where $i \sim Bino(n, p)$ and $\lambda = \frac{r\epsilon}{\Delta_r}$.

In our MPC-DP general framework Prot. 2.1, it can be reformulated as:

$$M_{SNGGauss}(\langle f_r(D)\rangle) = \langle f_r(D)\rangle + \left(2^{52} + \langle i\rangle\right) \cdot r - 2^{52} \cdot r,$$

where r, ϵ, Δ_r, λ are public known.

We present the MPC-DP protocol $\Pi^{SNGLap}\left(\langle \boldsymbol{f_r(D)}\rangle^{B,FL}, \lambda\right)$ for $M_{SNGLap}(\langle f_r(D)\rangle, r)$. We assume the parties have already computed $\langle \boldsymbol{f_r(D)}\rangle^{B,FL}$ and focus on generating the integer share $\langle \boldsymbol{i}\rangle^{B,UI}$.

Note:

1. The resolution parameter $r = 2^k$, where $k \in [-1074 \ldots 972]$.
2. $f_r(D)$ is a multiple of r and can be represent exactly as floating-point number.
3. In line 2, both the calculation $i + 2^{52}$ and converion UI2FL$\left(i + 2^{52}\right)$ (from unsigned integer to floating-point number) is exact when $i \leq 2^{52} - 1$.
4. i_{float} can be represented in floating-point format as: $i_{float} = (-1)^S (1.d_1 \ldots d_{52})_2 \times 2^{e-1023}$, where $(d_1 \ldots d_{52})_2 = i$.
5. In line 3, the calculation of $i_{float} * r$ can be done by adding k (because $r = 2^k$) to the the exponent of i_{float}, which is exact under floating-point arithmetic.

Protocol: $\Pi^{SNGLap}(\langle \boldsymbol{f_r(D)}\rangle, \lambda)$

Input: $\langle \boldsymbol{f_r(D)}\rangle^{B,FL}$,λ

Output: $\left\langle \boldsymbol{x_{SNGLap}}\right\rangle^{B,FL}$

1 : Parties run $\langle \boldsymbol{i}\rangle^{B,UI} = \Pi^{DGeometric}(\lambda)$.

2 : Parties compute $\left\langle \boldsymbol{i_{float}}\right\rangle^{B,FL} = \text{UI2FL}\left(\langle \boldsymbol{i}\rangle^{B,UI} + 2^{52}\right)$.

3 : Parties compute $\left\langle \boldsymbol{Y_{LapNoise}}\right\rangle^{B,FL} = \left\langle \boldsymbol{i_{float}}\right\rangle^{B,FL} * r - 2^{52} * r$.

4 : Parties compute $\left\langle \boldsymbol{x_{SNGLap}}\right\rangle^{B,FL} = \langle \boldsymbol{f_r(D)}\rangle^{B,FL} + \left\langle \boldsymbol{Y_{LapNoise}}\right\rangle^{B,FL}$.

Protocol 3.12: MPC-DP protocol for SNG Laplacian Mechanism.

3.6 Discrete Gausisan 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 $\mathcal{N}_{\mathbb{Z}}\left(\mu, \sigma^2\right)$ has probability mass function $\Pr\left(x \mid \mu, \sigma^2\right) = \frac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sum_{y \in \mathbb{Z}} e^{\frac{-(y-\mu)^2}{2\sigma^2}}}$, where $x \in \mathbb{Z}$.

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

Algorithm: $Algo^{DiscreteGauss}$

Input: None

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

$t \leftarrow \lfloor \sigma \rfloor + 1$

WHILE TRUE

  $L \leftarrow Algo^{DiscreteLap}(t)$

  $B \leftarrow Algo^{BernoulliEXP}\left(e^{\frac{-\left(|L| - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}}\right)$

  IF $B == 0$

    CONTINUE

  ELSE

    RETURN $x \leftarrow L$

Algorithm 3.7: Algorithm for discrete Gaussian.

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

Input: σ

Output:

1 : Each party locally calculate $t = \lfloor \sigma \rfloor + 1$

2 : For $j \in [0, \ldots, iter)$

3 : Parties run $\langle sign_j \rangle^B, \langle \boldsymbol{m}_j \rangle^{B,UI} = \Pi^{DiscreteLap}(t, 1)$

4 : Parties calculate $\langle b_j \rangle^B = \Pi^{BernoulliEXP}\left(\frac{\left(\langle |\boldsymbol{m}_j| \rangle^{B,F} - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}\right)$

5 : Parties run $\langle B_j \rangle^B = \Pi^{BernoulliEXP}()$

6 : choose xxxxx

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

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

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

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

1 : **WHILE TRUE**

2 : $s \leftarrow Algo^{Bernoulli}(0.5)$

3 : $m \leftarrow Algo^{GeometricExp}\left(\frac{denom}{numer}\right)$

4 : **IF** $s == 1 \wedge m == 0$

5 : **CONTINUE**

6 : **RETURN** $x \leftarrow m(1 - 2s)$

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

List of Figures

List of Tables

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+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. 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+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+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. GOLDREICH, 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. VENKITASUBRAMANIAM. **"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. 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.

[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: $\Pr(x \mid p) = (1-p)^{x-1} p$) 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, an 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).

Algorithm: $Algo^{Geometric}$

Input: None
Output: $x \sim Geo(p = 0.5)$

```
1 : x ← 1
2 : WHILE r == 0
3 :     r ← Random8bits()
4 :     x ← x + LeadingZeros(r)
5 : RETURN x
```

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: $x \sim Bern(p)$
$u \leftarrow Uniform(0,1)$
IF $u < p$
    RETURN $x \leftarrow 1$
ELSE
    RETURN $x \leftarrow 0$

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

$Algo^{BernoulliEXP1}(\gamma)$ [CKS20] generates a random variable $x \sim Bern\left(p = e^{-\gamma}\right)$ 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: γ
Output: $x \sim Bern\left(p = e^{-\gamma}\right)$, **where** $\gamma \in [0,1]$
$j \leftarrow 1$
WHILE TRUE
    $b \leftarrow Algo^{Bernoulli}\left(\frac{\gamma}{j}\right)$
    IF $b == 1$
        $j \leftarrow j + 1$
    ELSE
        BREAK
RETURN $x \leftarrow$ MOD2(j)

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

$Algo^{BernoulliEXP}(\gamma)$ [CKS20] generates a random variable $x \sim Bern\left(p = e^{-\gamma}\right)$.
TODO: Prove correctness

Algorithm: $Algo^{BernoulliEXP}(\gamma)$

Input: None

Output: $x \sim Bernoulli\left(p = e^{-\gamma}\right)$

IF $\gamma \in [0,1]$
  $b_1 \leftarrow Algo^{BernoulliEXP1}(\gamma)$
  RETURN $x \leftarrow b_1$
ELSE
  FOR $j = 0$ **TO** $\lfloor \gamma \rfloor - 1$
    $b_2 \leftarrow Algo^{BernoulliEXP1}(-1)$
    IF $b_2 == 0$
      RETURN $x \leftarrow 0$
  $b_3 \leftarrow Algo^{BernoulliEXP1}(\lfloor \gamma \rfloor - \gamma)$
RETURN $x \leftarrow b_3$

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

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

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

Input: None

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

1 : **IF** $numer == 0$
2 : **RETURN** 0
3 : **WHILE TRUE**
4 : $u \leftarrow Algo^{RandInt}(denom)$
5 : $b_1 \leftarrow Algo^{BernoulliEXP1}\left(\frac{u}{denom}\right)$
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 $x \sim Geo\left(p = 1 - e^{\frac{numer}{denom}}\right)$.

$Algo^{RandInt}(m)$ 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 $b_0, \ldots, b_{l+\kappa-1}$
2 : Calculate $r = \sum_{j=0}^{l+\kappa-1} 2^j b_j$
3 : Calculate $x = r \mod m$

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

A.2 MPC Protocols: Building Blocks

TODO: ABY, share conversion in preliminary part
$Pow2(\langle \boldsymbol{x} \rangle)$ calculates 2^x, and the share type of x could be boolean or arthmetic sharings.

Protocol $\Pi^{PreOr}\left(\langle \boldsymbol{x} \rangle^B\right)$ calculate the prefix-OR of a l-bit string $\langle \boldsymbol{x} \rangle^B = \left(\langle x_0 \rangle^B, \ldots, \langle x_l \rangle^B\right)$ and output $\langle \boldsymbol{y} \rangle^B = \left(\langle y_0 \rangle^B, \ldots, \langle y_l \rangle^B\right)$ such that $\left\langle y_j \right\rangle^B = \vee_{k=0}^{j} \langle x_k \rangle^B$ 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 $\langle \boldsymbol{y} \rangle^B = \left(\langle y_0 \rangle^B, \ldots, \langle y_{l-1} \rangle^B\right)$ without communications between parties. To achieve this, each party locally generate a random l-bit string $\boldsymbol{x}$ and set $\langle \boldsymbol{y} \rangle^B = \boldsymbol{x}$.

Protocol $\Pi^{Bits}(x, l)$ outputs the l least significand bits $y_0, \ldots, y_{l-1}$ of the binary representation of x, where y_0 is the least siginificant 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 $u_0, \ldots, u_{iter-1}$ 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 $j \in [0 \ldots k)$ 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 $j \in [k \ldots iter)$.

Protocol: $\Pi^{Geometric}$

Input: None

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

1 : For $j \in [0 \ldots iter)$, each party locally run $\Pi^{UniformBits}(8)$ **in** parallel and obtain $\left\langle \boldsymbol{u_j} \right\rangle^B$.

2 : For $j \in [0 \ldots iter)$, parties run $\Pi^{LeadingZeros}\left(\left\langle \boldsymbol{u_j} \right\rangle^B, 8\right)$ **in** parallel and obtain $\left\langle \boldsymbol{z_j} \right\rangle^{B,UI}$.

3 : For $j \in [0 \ldots iter)$, parties run $\Pi^{BitsContainOne}\left(\left\langle \boldsymbol{u_j} \right\rangle^B\right)$ **in** parallel and obtain $\left\langle o_j \right\rangle^B$.

4 : Parties run $\left(\langle e_0 \rangle^B, \ldots, \langle e_{iter-1} \rangle^B\right) = \Pi^{PreOr}\left(\langle o_0 \rangle^B, \ldots, \langle o_{iter-1} \rangle^B\right)$.

5 : For $j \in [0 \ldots iter)$, parties compute $\left\langle d_j \right\rangle^B = \mathrm{NOT}\left(\left\langle e_j \right\rangle^B\right)$ **in** parallel.

6 : Each party locally set bits $\left(\left\langle z_0^{ind} \right\rangle^B, \ldots, \left\langle z_{iter-1}^{ind} \right\rangle^B\right) = \left(1, \langle d_0 \rangle^B, \ldots, \langle d_{iter-2} \rangle^B\right)$.

7 : Parties compute $\langle \boldsymbol{x} \rangle^{B,UI} = \sum_{j=0}^{iter-1} \left\langle \boldsymbol{z_j} \right\rangle^{B,UI} \wedge \left\langle z_j^{ind} \right\rangle^B$.

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

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

Protocol: $\Pi^{Bernoullli}\left(\langle \boldsymbol{p} \rangle^{B,FL}\right)$

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

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

1 : Parties run $\langle \boldsymbol{U}^* \rangle^{B,FL} = \Pi^{UniformFloat}$

2 : Parties calculate $\langle x \rangle^{B} = \text{CMP}\left(\langle p \rangle^{B,FL}, \langle \boldsymbol{U}^* \rangle^{B,FL}\right)$

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

$\Pi^{BernoulliEXP1}\left(\langle \boldsymbol{\gamma} \rangle^{B,FL}\right)$ convert $Algo^{BernoulliEXP1}(\gamma)$ [CKS20] into MPC protocols. We assume that $Algo^{BernoulliEXP1}(\gamma)$ 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 $p_0, \ldots, p_{iter-1}$ 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: $\Pi^{BernoulliEXP1}\left(\langle \boldsymbol{\gamma} \rangle^{B,FL}\right)$

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

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

1 : For $j \in [1 \ldots iter)$

2 : Parties calculate $\left\langle \boldsymbol{p_j} \right\rangle^{B,FL} = \text{DIV}\left(\langle \boldsymbol{\gamma} \rangle^{B,FL}, j\right)$

3 : Parties run $\left\langle b_j \right\rangle^{B} = \Pi^{Bernoullli}\left(\left\langle \boldsymbol{p_j} \right\rangle^{B,FL}\right)$

4 : Parties calculate $\left\langle flag_j \right\rangle^{B} = \text{EQ}\left(\left\langle b_j \right\rangle^{B}, 1\right)$

5 : Parties run $\left(\langle e_0 \rangle^{B}, \ldots, \langle e_{iter} \rangle^{B}\right) = \Pi^{PreOr}\left(\langle flag_0 \rangle^{B}, \ldots, \langle flag_{iter} \rangle^{B}\right)$

6 : For $j \in [1 \ldots iter)$, parties compute $\left\langle p_j \right\rangle^{B} = \left\langle e_j \right\rangle^{B} - \left\langle e_{j-1} \right\rangle^{B}$, where $\langle p_0 \rangle^{B} = \langle e_0 \rangle^{B}$

7 : Parties compute $\langle x \rangle^{B} = \sum_{j=0}^{iter-1} \left\langle p_j \right\rangle^{B} * \left\langle b_j \right\rangle^{B}$

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

$\Pi^{BernoulliEXP}\left(\langle \boldsymbol{\gamma} \rangle^{B,FL}\right)$ convert $Algo^{BernoulliEXP}(\gamma)$ [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 calcualte the conditiaon 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 calcualte $\langle x\rangle^B = \left\langle cond_{\gamma\in[0,1]}\right\rangle^B * \langle b_1\rangle^B + \text{NOT}\left(\left\langle cond_{\gamma\in[0,1]}\right\rangle^B\right) * \left(\langle cond_{b2}\rangle^B * \langle b_2\rangle^B + \text{NOT}\left(\langle cond_{b2}\rangle^B\right) * \langle b_3\rangle^B\right)$.

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

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

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

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

1 : Parties calculate $\left\langle cond_{\gamma\in[0,1]}\right\rangle^B = \text{CMP}\left(\langle\gamma\rangle^{B,FL}, 0\right) * \text{CMP}\left(1, \langle\gamma\rangle^{B,FL}\right)$

2 : Parties run $\langle b_1\rangle^B = \Pi^{BernoullliEXP1}\left(\langle\gamma\rangle^{B,FL}\right)$

3 : For $j \in [0, \ldots, iter)$

4 : Parties run $\left\langle b_{j,2}\right\rangle^B = \Pi^{BernoullliEXP1}(-1)$

5 : Parties calculate $\left\langle cond_{b_{j,2}==0}\right\rangle^B = \text{EQ}\left(\left\langle b_{j,2}\right\rangle^B, 0\right)$

6 : Parties calcualate $(\langle e_0\rangle, \ldots, \langle e_{iter-1}\rangle) = \Pi^{PreOr}\left(\left\langle b_{0,2}\right\rangle^B, \ldots, \left\langle b_{iter-1,2}\right\rangle^B\right)$

7 : For $j \in [1 \ldots iter)$, parties compute $\left\langle p_j\right\rangle^B = \left\langle e_j\right\rangle^B - \left\langle e_{j-1}\right\rangle^B$, where $\langle p_0\rangle^B = \langle e_0\rangle^B$

8 : Parties calculate $\langle b_2\rangle^B = \sum_{j=0}^{iter-1} \left\langle p_j\right\rangle^B * \left\langle b_{j,2}\right\rangle^B$

9 : Parties calculate $\left\langle \boldsymbol{j}_{b_{j,2}==0}\right\rangle^{B,UI} = \Pi^{LeadingZeros}\left(\langle p_0\rangle^B, \ldots, \langle p_{iter-1}\rangle^B\right)$

10 : Parties calculate $\langle\lfloor\gamma\rfloor\rangle^{B,FL} = \text{Floor}\left(\langle\gamma\rangle^{B,FL}\right)$

11 : Parties calcualte $\langle\lfloor\gamma\rfloor\rangle^{B,UI} = \text{FL2UI}\left(\langle\lfloor\gamma\rfloor\rangle^{B,FL}\right)$

12 : Parties calcualte $\langle cond_{b2}\rangle^B = \text{CMP}\left(\langle\lfloor\gamma\rfloor\rangle^{B,UI}, \left\langle \boldsymbol{j}_{b_{j,2}==0}\right\rangle^{B,UI}\right)$

13 : Parties calculate $\langle\lfloor\gamma\rfloor - \gamma\rangle^{B,FL} = \text{SUB}\left(\langle\lfloor\gamma\rfloor\rangle^{B,FL}, \langle\gamma\rangle^{B,FL}\right)$

14 : Parties run $\langle b_3\rangle^B = \Pi^{BernoullliEXP1}\left(\langle\lfloor\gamma\rfloor - \gamma\rangle^{B,FL}\right)$

15 : Parties calculate $\langle x\rangle^B$

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

$\Pi^{RandInt}(m)$ convert $Algo^{RandInt}(m)$ into MPC protocols.
TODO: Problem with unsigned integer because κ is requires be greater than 64.

Protocol: $\Pi^{RandInt}(m)$

Input: m

Output: $\langle x \rangle^{B,UI}$, **where** $x \in [0,\ldots,m)$

1 : Each party locally generate $l+\kappa$-bit string $(b_0,\ldots,b_{l+\kappa-1})$ and set the as the bits of $\langle r \rangle^{B,UI} = \left(\langle b_0 \rangle^B,\ldots,\langle b_{l+\kappa-1} \rangle^B\right.$

2 : Parties calculate $\langle x \rangle^{B,UI} = \mathrm{MOD}\left(\langle r \rangle^{B,UI}, m\right)$

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

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

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

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

Input: $numer$, $denom$

Output: $\langle x \rangle^{B,UI}$, **where** $x \sim Geo\left(p = 1 - e^{\frac{numer}{denom}}\right)$

1 : For $j \in [0, iter_1)$ Parties run $\left\langle u_j \right\rangle^{B,UI} = \Pi^{RandInt}(denom)$

2 : Parties calculate $\left\langle r_j^1 \right\rangle^{B,FL} = \mathrm{DIV}\left(\left\langle u_j \right\rangle^{B,UI}, denom\right)$

3 : Parties run $\left\langle b_j^1 \right\rangle^B = \Pi^{BernoullliEXP1}\left(\left\langle r_j^1 \right\rangle^{B,FL}\right)$

4 : Parties calcualate $\left(\left\langle e_0^1 \right\rangle,\ldots,\left\langle e_{iter_1-1}^1 \right\rangle\right) = \Pi^{PreOr}\left(\left\langle b_0^1 \right\rangle^B,\ldots,\left\langle b_{iter-1}^1 \right\rangle^B\right)$

5 : For $j \in [1\ldots iter_1)$, parties compute $\left\langle p_j^1 \right\rangle^B = \left\langle e_j^1 \right\rangle^B - \left\langle e_{j-1}^1 \right\rangle^B$, where $\left\langle p_0^1 \right\rangle^B = \left\langle e_0^1 \right\rangle^B$

6 : Parties calculate $\langle u \rangle^{B,UI} = \sum_{j=0}^{iter_1-1} \left\langle p_j^1 \right\rangle^B * \left\langle u_j \right\rangle^{B,UI}$

7 : For $j \in [0, iter_2)$

8 : Parties run $\left\langle b_j^2 \right\rangle^B = \Pi^{BernoullliEXP1}(1)$

9 : Parties calcualate $\left(\left\langle e_0^2 \right\rangle,\ldots,\left\langle e_{iter_1-1}^2 \right\rangle\right) = \Pi^{PreOr}\left(\mathrm{NOT}\left(\left\langle b_0^2 \right\rangle^B\right),\ldots,\mathrm{NOT}\left(\left\langle b_{iter-1}^2 \right\rangle^B\right)\right)$

10 : For $j \in [1\ldots iter_2)$, parties compute $\left\langle p_j^2 \right\rangle^B = \left\langle e_j^2 \right\rangle^B - \left\langle e_{j-1}^2 \right\rangle^B$, where $\left\langle p_0^2 \right\rangle^B = \left\langle e_0^2 \right\rangle^B$

11 : Parties run $\langle k \rangle^{B,UI} = \Pi^{LeadingZeros}\left(\left\langle p_0^2 \right\rangle^B,\ldots,\left\langle p_{iter_2}^2 \right\rangle^B\right)$

12 : Parties calculate $\langle x \rangle^{B,UI} = \mathrm{FL2UI}\left(\mathrm{Floor}\left(\mathrm{DIV}\left(\mathrm{ADD}\left(\mathrm{MUL}(k, denom), \langle u \rangle^{B,UI}\right), numer\right)\right)\right)$

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

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

Input: $numer$, $denom$

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

1 : For $j \in [0, \ldots, iter)$ Parties run $\left\langle s_j \right\rangle^B = \Pi^{Bernoulli}(0.5)$

2 : Parties run $\left\langle \boldsymbol{m}_j \right\rangle^{B,UI} = \Pi^{GeometricEXP}(denom, numer)$

3 : Parties calculate $\left\langle flag_j \right\rangle^B = \text{NOT}\left(\text{EQ}\left(\left\langle s_j \right\rangle^B, 1\right) * \text{EQ}\left(\left\langle \boldsymbol{m}_j \right\rangle^{B,UI}, 0\right)\right)$

4 : Parties compute $\left(\langle e_0 \rangle^B, \ldots, \langle e_{iter-1} \rangle^B\right) = \Pi^{PreOr}\left(\langle flag_0 \rangle^B, \ldots, \langle flag_{iter-1} \rangle^B\right)$

5 : For $j \in [1 \ldots iter)$, parties compute $\left\langle p_j \right\rangle^B = \left\langle e_j \right\rangle^B - \left\langle e_{j-1} \right\rangle^B$, where $\langle p_0 \rangle^B = \langle e_0 \rangle^B$

6 : Parties calculate $\langle s \rangle^B = \sum_{j=0}^{iter-1} \left\langle s_j \right\rangle^B * \left\langle p_j \right\rangle^B$

7 : Parties calculate $\langle \boldsymbol{m} \rangle^{B,UI} = \sum_{j=0}^{iter-1} \left\langle \boldsymbol{m}_j \right\rangle^{B,UI} * \left\langle p_j \right\rangle^B$

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 $\Pi^{LeadingZeros}\left(\langle \boldsymbol{s} \rangle^B, l\right)$ counts the number of leading zeros in a l-bit string $\boldsymbol{s}$. We apply Π^{PreOr} to find the position e of the first appearance of one in string $\boldsymbol{s}$. 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: $\Pi^{LeadingZeros}\left(\langle \boldsymbol{s} \rangle^B, l\right)$

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

Output: $\langle z \rangle^A$

1 : Parties run $\Pi^{PreOr}\left(\langle \boldsymbol{s} \rangle^B\right)$ and obtain $\left\langle e_j \right\rangle^B$.

2 : For $j \in [0 \ldots l)$, parties compute $\left\langle d_j \right\rangle^B = \text{NOT}\left(\left\langle e_j \right\rangle^B\right)$ **in** $parallel$.

3 : Parties compute $\langle \boldsymbol{z} \rangle^{B,UI} = \sum_{j=0}^{l-1} \text{B2A}\left(\left\langle d_j \right\rangle^B\right)$.

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

Protocol $\Pi^{BitsContainOne}\left(\langle \boldsymbol{s} \rangle^B, l\right)$ outputs $o = 1$ if the given l-bit string $\boldsymbol{s}$ 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 $\langle o \rangle^B = \vee_{j=0}^{l} \langle s_j \rangle^B$

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

TODO: Binary2Unary needs to be improved. $\Pi^{Binary2Unary}\left(\langle a \rangle^A, l\right)$ [ABZS12] converts integer a from binary to unary bitwise representation and outputs a l-bit string $\boldsymbol{p} = (p_0, \ldots, p_{l-1})$, where the a least significant bits $(p_0, \ldots, p_{a-1})$ 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 $\left(g_{l-1}, \ldots, g_j\right) = \left(\overline{1}_{(l-j+1)}\right)$ and $\left(g_{j-1}, \ldots, g_0\right) = \left(\overline{0}_{(j)}\right)$, where $j - 1 = a$. Finally, we calculate $(p_{l-1}, \ldots, p_0) = \text{NOT}\left(g_{l-1}, \ldots, g_0\right)$ and the number of non-zero bits in $(p_{l-1}, \ldots, p_0)$ equals to a. The share conversion in line 4 can be omitted if arithmetic of boolean shares is available.

Protocol: $\Pi^{Binary2UnaryOld}\left(\langle a \rangle^A, l\right)$

Input: $\langle \boldsymbol{a} \rangle^B$, l

Output: $\langle p_0 \rangle^B, \ldots, \langle p_{l-1} \rangle^B$

1 : Parties compute $\langle \mathbf{2^a} \rangle^B = \text{Pow2}\left(\langle a \rangle^A, l\right)$

2 : Parties compute $\langle 2^a \rangle^A = \text{B2A}\left(\langle \mathbf{2^a} \rangle^B\right)$

3 : Parties run $\Pi^{RandBits}\left(l + k\right)$ and obtain $\langle u_0 \rangle^B, \ldots, \langle u_{l+k-1} \rangle^B$.

4 : Parties reconstruct $c \leftarrow \text{Rec}\left(\langle 2^a \rangle^A + \text{B2A}\left(\langle u_{l+k-1} \rangle^B, \ldots, \langle u_0 \rangle^B\right)\right)$

5 : Each party locally run $\Pi^{Bits}\left(c, l\right)$ and obtain $(c_0, \ldots, c_{l-1})$

6 : For $j \in [0 \ldots l)$, each party locally compute $\left\langle t_j \right\rangle^B = \text{XOR}\left(c_j, \left\langle u_j \right\rangle^B\right)$

7 : For $j \in [0 \ldots l)$, parties compute $\Pi^{PreOr}\left(\langle t_0 \rangle^B, \ldots, \langle t_l \rangle^B\right)$ and obtain $\langle g_0 \rangle^B, \ldots, \langle g_l \rangle^B$

8 : For $j \in [0 \ldots l)$, parties compute $\left\langle p_j \right\rangle^B = \text{NOT}\left(\left\langle g_j \right\rangle^B\right)$

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

Protocol: $\Pi^{Binary2UnaryNEW}\left(\langle \boldsymbol{a}\rangle^{B,UI}, l\right)$

Input: $\langle \boldsymbol{a}\rangle^{B,UI}$, l

Output: $\langle p_0\rangle^{B}, \ldots, \langle p_{l-1}\rangle^{B}$

1 : Parties compute $\langle \boldsymbol{t}\rangle^{B,UI} = \text{POW2}\left(\langle \boldsymbol{a}\rangle^{B,UI}\right)$.

2 : Parties compute $\left(\langle e_0\rangle^{B}, \ldots, \left\langle e_{64-1}\right\rangle^{B}\right) = \Pi^{PreOr}\left(\langle t_0\rangle^{B}, \ldots, \left\langle t_{64-1}\right\rangle^{B}\right)$.

3 : Each party locally set $\langle p_0\rangle^{B} = \text{NOT}\left(\langle e_0\rangle^{B}\right), \ldots, \langle p_{l-1}\rangle^{B} = \text{NOT}\left(\langle e_{l-1}\rangle^{B}\right)$.

Protocol A.11: MPC Protocol for binary to unary conversion.

Protocol: $\Pi^{Split}\left(\langle \boldsymbol{L}\rangle^{B,UI}, \langle \boldsymbol{R}\rangle^{B,UI}, \lambda\right)$

Input: $\langle \boldsymbol{L}\rangle^{B,UI}$, $\langle \boldsymbol{R}\rangle^{B,UI}$, λ

Output: $\langle \boldsymbol{M}\rangle^{B,UI}$, **where** $M = L - \textbf{int}\left(\dfrac{\ln(0.5) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda}\right)$

1 : Parties compute $\langle \boldsymbol{t}\rangle^{B,FL} = \dfrac{\text{LN}(0.5) + \text{LN}\left(1 + \text{EXP}\left(-\lambda \cdot \left(\langle \boldsymbol{R}\rangle^{B,UI} - \langle \boldsymbol{L}\rangle^{B,UI}\right)\right)\right)}{\lambda}$.

2 : Parties compute $\langle \boldsymbol{M}\rangle^{B,UI} = \langle \boldsymbol{L}\rangle^{B,UI} - \text{FL2UI}\left(\langle \boldsymbol{t}\rangle^{B,FL}\right)$.

Protocol A.12: MPC Protocol for $\text{Split}(L,R,\lambda) = L - \text{int}\left(\frac{\ln(0.5)+\ln\left(1+e^{-\lambda(R-L)}\right)}{\lambda}\right)$.

Protocol: $\Pi^{Proportion}\left(\left\langle \boldsymbol{L}_{j-1}\right\rangle^{B,UI}, \left\langle \boldsymbol{R}_{j-1}\right\rangle^{B,UI}, \left\langle \boldsymbol{M}_{j}\right\rangle^{B,UI}, \lambda\right)$

Input: $\langle \boldsymbol{L}\rangle^{B,UI}$, $\langle \boldsymbol{R}\rangle^{B,UI}$, $\langle \boldsymbol{M}\rangle^{B,UI}$, λ

Output: $\langle \boldsymbol{Q}\rangle^{B,FL}$, **where** $Q = \dfrac{e^{-\lambda(M-L)} - 1}{e^{-\lambda(R-L)} - 1}$

1 : Parties compute $\langle \boldsymbol{Q}\rangle^{B,FL} = \dfrac{e^{-\lambda \cdot \left(\langle \boldsymbol{M}\rangle^{B,FL} - \langle \boldsymbol{L}\rangle^{B,FL}\right)} - 1}{e^{-\lambda \cdot \left(\langle \boldsymbol{R}\rangle^{B,FL} - \langle \boldsymbol{L}\rangle^{B,FL}\right)} - 1}$.

Protocol A.13: MPC Protocol for $\text{Proportion}(L,R,M,\lambda) = \frac{e^{-\lambda(M-L)}-1}{e^{-\lambda(R-L)}-1}$.