TECHNISCHE
UNIVERSITÄT
DARMSTADT

Master Thesis

Securely Realizing Output Privacy in MPC

Liang Zhao

July 10, 2022

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

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

Supervisors: M.Sc. Helen Möllering
M.Sc. Oleksandr Tkachenko
Prof. Dr.-Ing. Thomas Schneider

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

Hiermit versichere ich, Liang Zhao, die vorliegende Master Thesis 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, Liang Zhao, have written the submitted Master Thesis 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, July 10, 2022

Liang Zhao

Abstract

Nowadays, the world has become an information-driven society where the distribution and processing of information is one important economic activity. However, the centralized database may contain sensitive data that would lead to privacy violations if the data or its aggregate statistics are disclosed. Secure Multi-Party Computation (SMPC) enables multiple parties to compute an arbitrary function on their private inputs and reveals no information beyond the computation result. Differential Privacy (DP) is a technique that can preserve the individual's privacy by perturbing the aggregate statistics with random noise. The hybrid approach combining SMPC and DP would provide a robust privacy guarantee and maintain the utility of the aggregate statistics. The theoretical definition of DP assumes precise noise sampling and arithmetic operations under real numbers. However, in the practical implementation of perturbation mechanisms, fixed-point or floating-point numbers are used to represent real numbers that lead to the violation of DP, as Mironov [Mir12] and Jin et al. [JMRO22] showed. This thesis explores the possibilities of *securely* generating distributed random noise in SMPC settings and builds a variety of perturbation mechanisms. Specifically, we evaluate the performance of fixed-point and floating-point arithmetic for noise generation in SMPC and choose the most efficient SMPC protocols to build the perturbation mechanisms.

Contents

1 Preliminaries

In this chapter, we start with the notations used in this thesis § 1.1. Afterwards, we describe the basic knowledge of secure multi-party computation in § 1.2. Finally, we introduce the background knowledge and theory of differential privacy in § 1.3.

1.1 Notations

For $a, b, c \in \mathbb{N}$, (a, b) denotes $\{x \in \mathbb{R} \,|\, a < x < b\}$, and $[a, b]$ denotes $\{x \in \mathbb{R} \,|\, a \leq x \leq b\}$. $\{a, b, c\}$ is a set containing the three numbers. $\mathbb{D}$ denotes the set of floating-point numbers, and $\mathbb{D} \cap (a, b)$ contains floating-point numbers in the interval (a, b).

Let $P_1, \ldots P_N$ denote N computation parties. The value x that is secret shared among N parties are denoted by $\langle x \rangle^{S,D} = \left(\langle x \rangle_1^{S,D}, \ldots, \langle x \rangle_N^{S,D}\right)$, where $\langle x \rangle_i^{S,D}$ is hold by party P_i. Superscript $S \in \{A, B, Y\}$ denotes the sharing type (cf. § 1.2.3): A for arithmetic sharing, B for Boolean sharing with GMW, Y for Yao sharing with BMR. Superscript $D \in \{UINT, INT, FX, FL\}$ indicates the data type: $UINT$ for unsigned integer, INT for signed integer, FX for fixed-point number, and FL for floating-point number. We omit subscript and subscript when it is clear from the context.

Bold symbol $\langle \boldsymbol{x} \rangle$ denotes a vector of ℓ shared bits. We use XOR ($\oplus$), AND ($\wedge$), and NOT ($\neg$) in the logical opeartions. Let $\langle a \rangle^{S,D} \odot \langle b \rangle^{S,D}$ be the arithmetic operations on two shared numbers, where $\odot \in \{+, -, \cdot, \div, >, ==\}$. $\langle a \rangle^B \cdot \langle \boldsymbol{b} \rangle^B$ represents the bitwise AND operations between $\langle a \rangle^B$ and every Boolean sharing bit $\langle b \rangle^B \in \langle \boldsymbol{b} \rangle^B$. Let $\langle a \rangle^{FL} = \Pi^{UINT2FL}\left(\langle a \rangle^{UINT}\right)$ be the conversion from an shared unsigned integer $\langle a \rangle^{UINT}$ to a shared floating-point number $\langle a \rangle^{FL}$. Other data type conversion operations are defined in a similar manner.

1.2 Secure Multi-Party Computation

Secure Multi-Party Computation enables multiply parties to jointly evaluate a function on their private inputs while revealing only the computation result. Yao [Yao82] introduced the concept of secure two-party computation with Yao's Millionaires' problem (i.e., two millionaires wish to know who is richer without revealing their actual wealth) and proposed the garbled circuit protocol [Yao86] as a solution. In the garbled circuit protocol, the target function is represented as a Boolean circuit consisting of connected gates and wires. One

party called gabler is responsible for garbling the circuit, and the other party called evaluator evaluates the garbled circuit and outputs the result.

Afterwards, Beaver, Micali and Rogaway (BMR) [BMR90] generalized Yao's Garbled Circuit protocol [Yao86] to multi-party settings. Goldreich, Micali and Wigderson (GMW) [GMW87] proposed a general solution to SMPC based on secret sharing, where each party splits his data into several shares and sends it to each of the parties. Secret sharing guarantees that any secret shares held by single party leak no information about the parties' private input.

Generally, the execution of MPC protocols is separated into two phases: an offline (or preprocessing) phase and an online phase. In the offline phase, the parties compute everything that does not depend on the private input. In the online phase, the parties compute the input-dependent part.

1.2.1 Security Model

The standard approach to prove the security of cryptographic protocols is to consider adversaries with different capabilities. We describe two types of adversaries: the *semi-honest* adversary and the *malicious* adversary. We refer to [EKR17, Chapter 2] for a formal and detailed description of the security model.

Semi-honest adversaries (also known as passive adversaries) try to infer additional information of other parties from the messages during the protocol execution without attempting to break the protocol. Therefore, it is a weak security model and only prevents the unintentional disclosure of information between parties. The semi-honest protocols are usually very efficient and the first step to design protocols with stronger security guarantees.

Malicious adversaries (also known as active adversaries) may cause corrupted parties to arbitrarily deviate from the protocol specification and attempt to learn information about the other parties' inputs. Protocols against malicious adversaries usually deploy cryptographic mechanisms to ensure that the parties cannot deviate from the protocol specification. Therefore, the protocol is often more expensive than the protocol against semi-honest adversaries.

1.2.2 Cryptographic Primitives

Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic primitive that enables two parties to obliviously transfer one value out of two values. Specifically, the sender has inputs (x_0, x_1), and the receiver has a choice bit c. Oblivious transfer protocol receives the inputs from the sender and receiver, and outputs x_c to the receiver. It guarantees that the sender does not learn anything about c and the receiver does not learn about x_{1-c}. Impagliazzo and Rudich [IR89] showed that a *black-box* reduction from OT to a one-way function [Isr06, Chapter 2] is as

hard as proving $P \neq NP$, which implies that OT requires relatively expensive (than symmetric cryptography) public-key cryptography [RSA78].

Nevertheless, Ishai et al. [IKNP03] proposed OT *extension* techniques that extend a small number of OTs based on public-key cryptography to a large number of OTs with efficient symmetric cryptography. Asharov et al. [ALSZ17] proposed specific OT functionalities for the optimization of SMPC protocols, such as Correlated Oblivious Transfer (C-OT) and Random Oblivious Transfer (R-OT). In C-OT, the sender inputs a correlation function f_Δ (e.g., $f_\Delta(x) = x \oplus \Delta$, where Δ is only known by the sender) and receives random values x_0 and $x_1 = f_\Delta(x_0)$. The receiver inputs a choice bit c and receives x_c. In R-OT, the sender has no inputs and receives random values (x_0, x_1), and the receiver inputs a choice bit c and receives x_c.

Multiplication Triples

Multiplication Triples (MTs) were proposed by Beaver [Bea91] that can be precomputed to reduce the online complexity of SMPC protocols by converting expensive operations (e.g., arithmetic multiplication and logical AND) to linear operations (e.g., arithmetic addition and logical XOR).

A multiplication triple has the form $\left(\langle a\rangle^S, \langle b\rangle^S, \langle c\rangle^S\right)$ with $S \in \{B, A\}$. In Boolean sharing with GMW (cf. § 1.2.3), we have $c = a \wedge b$ for Boolean sharing and $c = a \cdot b$ for arithmetic sharing (cf. § 1.2.3). Multiplication triples can be generated using C-OT (cf. § 1.2.2) in the two-party setting [DSZ15] or in the multi-party setting [BDST22].

1.2.3 MPC Protocols

We describe the SMPC protocols that are secure against $N-1$ semi-honest corruptions: Arithmetic sharing (cf. § 1.2.3), Boolean sharing with GMW (cf. § 1.2.3), and Yao sharing with BMR (cf. § 1.2.3). We refer to [DSZ15; BDST22] for a formal and detailed description.

Arithmetic Sharing (A)

The arithmetic sharing protocol enables parties to evaluate arithmetic circuits consisting of addition and multiplication gates. In arithmetic sharing, an ℓ-bit value x is shared additively among N parties as $\left(\langle x\rangle_1^A, \ldots, \langle x\rangle_N^A\right) \in \mathbb{Z}_{2^\ell}^N$, where $x = \sum_{i=1}^N \langle x\rangle_i^A \mod 2^\ell$ and party P_i holds $\langle x\rangle_i^A$. Value x can be reconstructed by letting each party P_i sends $\langle x\rangle_i^A$ to one specific party who computes $x = \sum_{i=1}^N \langle x\rangle_i^A \mod 2^\ell$. The addition of arithmetic shares can be computed without parties' interaction. Suppose the parties hold shares $\langle x\rangle_i^A$, $\langle y\rangle_i^A$, and wish to compute $\langle z\rangle = a \cdot \langle x\rangle + \langle y\rangle + b$ with public value a, $b \in \mathbb{Z}_{2^\ell}$. Then, one specific party P_1 computes $\langle z\rangle_1^A = a \cdot \langle x\rangle_1^A + \langle y\rangle_1^A + b$, and the other parties compute $\langle z\rangle_i^A = a \cdot \langle x\rangle_i^A + \langle y\rangle_i^A$ locally.

The multiplication of arithmetic shares can be performed using MTs (cf. § 1.2.2). Suppose $\left(\langle a\rangle^A, \langle b\rangle^A, \langle c\rangle^A\right)$ is an MTs in $\mathbb{Z}_{2^\ell}$, where $c = a \cdot b$. To compute $\langle z\rangle^A = \langle x\rangle^A \cdot \langle y\rangle^A$, the parties first locally compute $\langle d\rangle_i^A = \langle x\rangle_i^A - \langle a\rangle_i^A$ and $\langle e\rangle_i^A = \langle y\rangle_i^A - \langle b\rangle_i^A$, and reconstruct them to get d and e. Finally, each party P_i compute $\langle z\rangle_i^A = \langle c\rangle_i^A + e \cdot \langle x\rangle_i^A + d \cdot \langle y\rangle_i^A - d \cdot e$ locally.

Boolean Sharing with GMW

Boolean GMW protocol [GMW87] uses XOR-based secret sharing and enables multiple parties to evaluate the function represented as a Boolean circuit. A bit $x \in \{0,1\}$ is shared among N parties as $\left(\langle x\rangle_1^B, \ldots, \langle x\rangle_N^B\right) \in \{0,1\}^N$, where $x = \bigoplus_{i=1}^N \langle x\rangle_i^B$. Boolean GMW can be seen as a special case of arithmetic sharing protocol. Operation $\langle x\rangle_i^B \oplus \langle y\rangle_i^B$ and $\langle x\rangle_i^B \wedge \langle y\rangle_i^B$ are computed analogously as in the arithmetic sharing.

Yao Sharing with BMR

We first present Yao's Garbled Circuit protocol [Yao86] following the steps described in work [LP09] and then extend it to multi-party settings with BMR [BMR90] protocol. Yao's Garbled Circuit protocol [Yao86] enables two parties, the garbler, and the evaluator, to securely evaluate any functionality represented as a Boolean circuit.

1. Circuit Garbling. The garbler converts the jointly decided function f into a Boolean circuit C, and selects a pair of random κ-bit keys $\left(k_0^i, k_1^i\right) \in \{0,1\}^{2\kappa}$ to represent logical value 0 and 1 for each wire. For each gate g in the Boolean circuit C with input wire a and b, and output wire c, the gabler uses the generated random keys $\left(k_0^a, k_1^a\right)$, $\left(k_0^b, k_1^b\right)$, $\left(k_0^c, k_1^c\right)$ to create a garbled table $\tilde{g}$ based on the function table of g. For example, gate g is an AND gate and has a function table as Tab. 1.1 shows, the gabler encrypts the keys of wire c and permutes the entries to generate Tab. 1.2. Note that the symmetric encryption function Enc_k uses a secret-key k to encrypt the plaintext, and its decryption function Dec_k decrypts the ciphertext successfully only when the identical secret-key k is given. When all the gates in Boolean circuit C are garbled, the gabler sends the garbled circuit $\widetilde{C}$ that consists of garbled tables from all the gates to the evaluator for evaluation.

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.1: Function table of AND gate g.

2. Input Encoding. The gabler sends the wire keys corresponding to its input directly to the evaluator. To evaluate the garbled circuit $\widetilde{C}$, the evaluator needs the wire keys corresponding

$\tilde{c}$
$\mathrm{Enc}_{k_1^a,k_1^b}\left(k_1^c\right)$
$\mathrm{Enc}_{k_0^a,k_1^b}\left(k_0^c\right)$
$\mathrm{Enc}_{k_0^a,k_0^b}\left(k_0^c\right)$
$\mathrm{Enc}_{k_1^a,k_0^b}\left(k_0^c\right)$

Table 1.2: Garbled table of AND gate g with encrypted and permuted entries.

to his input. For each of the evaluator's input wire i with corresponding input bit c, the evaluator and the gabler run an 1-out-of-2-OT, where the gabler acts as a sender with inputs $\left(k_0^i, k_1^i\right)$, and the evaluator acts as a receiver with input c and receives k_c^i. Recall that OT (cf. § 1.2.2) guarantees that the gabler learns nothing about c and the evaluator learns only k_c^i.

3. Circuit Evaluation. After receiving the garbled circuit $\widetilde{C}$ and the keys of input wires, the evaluator begins to evaluate the garbled circuit $\widetilde{C}$. For each gate g with input wire a and b, output wire c, the evaluator uses the input wire keys $\left(k^a, k^b\right)$ to decrypt the output key k^c. When all the gates in the garbled circuit $\widetilde{C}$ are evaluated, the evaluator obtains the keys for the output wires. To reconstruct the output, either the gabler sends the mapping from output wire keys to plaintext bits to the evaluator, or the evaluator sends the decrypted output wire keys to the gabler.

Optimizations for Yao's Garbled Circuits. This part presents several prominent optimizations for Yao's Garbled Circuit protocol [Yao86]. Recall that in the evaluation step of Yao's garbled circuit $\widetilde{C}$ protocol, the evaluator needs to decrypt at most four entries to obtain the correct key of the output wire. Point and permute [BMR90] technique helps the evaluator to identify the entry that should be decrypted in garbled tables with an additional permutation bit for each wire. Garbled row reduction [NPS99] reduces the number of entries in the garbled table from four to three by fixing the first entry to a constant value. Free-XOR [KS08] allows the parties to evaluate XOR gates without interactions by choosing all the wire key pairs $\left(k_0^i, k_1^i\right)$ with the same fixed distance R (R is kept secret to the evaluator), e.g., k_0^i is chosen at random and k_1^i is set to $R \oplus k_0^i$. Fixed-Key AES garbling [BHKR13] reduces the encryption and decryption workload of Yao's Garbled Circuit protocol [Yao86] using a block cipher with a fixed key such that the AES key schedule is executed only once. Two-halves garbling [ZRE15] reduces the entry number of each AND gate from three to two by splitting each AND gate into two half-gates at the cost of one more decryption operation of the evaluator. Three-halves garbling [RR21] requires 25% less communication bits than the two-halves garbling at the cost of more computation.

BMR Protocol. BMR protocol [BMR90] extends Yao's Garbled Circuit protocol [Yao86] to the multi-party setting. Recall that in Yao's Garbled Circuit protocol, the circuit is garbled

by one party and evaluated by another party. At a high level, the BMR protocol enables the multi-party computation by having all parties jointly garbling the circuit in the offline phase. Then, each party sends the garbled labels that are associated with their private inputs to other parties. Next, each party plays the role of the evaluator and evaluates the garbled circuit locally. Finally, the parties use the received garbled label and the the result of local evaluation to compute the output.

1.2.4 SMPC Framework - MOTION

We build upon the SMPC framework MOTION [BDST22] that provides the following novel features:

1. Support for SMPC with N parties, full-threshold security (i.e., tolerating up to $N-1$ passive corruptions) and sharing conversions between *ABY*.
2. Implementation of primitive operations of SMPC protocols at the circuit's gate level and asynchronously evaluation, i.e., each gate is separately evaluated once their parent gates become ready.
3. Support for Single Instruction Multiple Data (SIMD), i.e., vectors of data are processed instead of single data, that can reduce memory footprint and communication.
4. Integration of HyCC compiler [BDK+18] that can generate efficient circuits for hybrid MPC protocols with functionality described in C programming language.

1.3 Differential Privacy

This section describes the concept of differential privacy in a formal mathematical view. We first introduce basic knowledge of probability distribution and random variable generation methods. Then, we describe traditional privacy preservation techniques and discuss their limitations. Next, we describe the motivation behind differential privacy and formalize its definition. Finally, we describe the differentially private mechanisms for realizing differential privacy.

1.3.1 Probability Distribution and Random Variable Generation

In this section, we introduce essential probability theory and random variable sampling methods as a preparation for differential privacy.

Continuous Probability Distribution

Definition 1.3.1 (Continuous Uniform Distribution)**.** *The continuous uniform distribution with two boundary parameters a and b, has the following probability density function:*

$$Uni(x \mid a, b) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

We use the probability density function to denote the corresponding probability distribution P or a random variable $x \sim P$. For example, $Uni(a, b)$ denotes the continuous uniform distribution with parameters a and b. We sometimes abuse notation and let $Uni(a, b)$ denote a random variable $x \sim Uni(a, b)$.

Definition 1.3.2 (Exponential Distribution)**.** *The exponential distribution with rate parameter $\lambda > 0$ has the following probability density function:*

$$Exp(x \mid \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{1.2}$$

The cumulative distribution function of an exponential distribution is defined as:

$$\Pr(x \mid \lambda) = \begin{cases} 1 - e^{-\lambda x} & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{1.3}$$

Definition 1.3.3 (Laplace Distribution [DR+14])**.** *The Laplace distribution (centered at 0) with scale parameter b, has the following probability density function:*

$$Lap(x \mid b) = \frac{1}{2b} e^{\left(-\frac{|x|}{b}\right)} \tag{1.4}$$

The Laplace distribution is a symmetric version of the exponential distribution. It is also called the double exponential distribution because it can be considered as an exponential distribution assigned with a randomly chosen sign.

The cumulative distribution function of the Laplace distribution is defined as:

$$\Pr(x \leq X \mid b) = \begin{cases} \frac{1}{2} e^{\frac{X}{b}} & \text{for } X \leq 0 \\ 1 - \frac{1}{2} e^{-\frac{X}{b}} & \text{for } X > 0 \end{cases} \tag{1.5}$$

Definition 1.3.4 (Gaussian Distribution)**.** *The univariate Gaussian (also known as the standard normal) distribution with mean μ and standard deviation σ, has the following probability density function:*

$$\mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{1.6}$$

Discrete Probability Distribution

Definition 1.3.5 (Bernoulli distribution)**.** *The Bernoulli distribution with parameter $p \in [0,1]$ has the following probability mass function:*

$$Bern(x \mid p) = \begin{cases} p & \text{for } x = 1 \\ 1-p & \text{for } x = 0 \end{cases} \tag{1.7}$$

Definition 1.3.6 (Binomial Distribution)**.** *The binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0,1]$ has the following probability mass function for $x \in \{0,1,2,\ldots,n\}$:*

$$Bino(x \mid n,p) = \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x} \tag{1.8}$$

Note that the distribution $Bino(n, p = 0.5) - \frac{n}{2}$ is symmetric about the y-axis, which we denote by $SymmBino(n, p = 0.5)$.

Definition 1.3.7 (Geometric Distribution)**.** *The geometric distribution with parameter $p \in [0,1]$ has the following probability mass function for $x \in \{0,1,2,\ldots\}$:*

$$Geo(x \mid p) = (1-p)^x\, p \tag{1.9}$$

Note that the geometric distribution models the number of trials until the first success (each trial has a success probability p). The cumulative distribution function of the geometric distribution is $\Pr(x \leq X \mid p) = 1-(1-p)^{X+1}$.

Definition 1.3.8 (Discrete Laplace Distribution [CKS20])**.** *The discrete Laplace distribution (also known as the two-side geometric distribution [GRS12]) with parameter $t > 0$ and $x \in \mathbb{Z}$ has the following probability mass function:*

$$DLap(x \mid t) = \frac{e^{\frac{1}{t}} - 1}{e^{\frac{1}{t}} + 1} \cdot e^{-\frac{|x|}{t}} \tag{1.10}$$

The discrete Laplace distribution can be generated by reflecting a geometric distribution across the y-axis and rescaling it such that its cumulative probability in the interval $(-\infty, \infty)$ equals to one.

Definition 1.3.9 (Discrete Gaussian Distribution [CKS20])**.** *The discrete Gaussian distribution with mean μ and standard deviation σ, has the following probability mass function for $x \in \mathbb{Z}$:*

$$DGau(x \mid \mu,\sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sum_{y\in\mathbb{Z}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}} \tag{1.11}$$

Probability Sampling Methods

Inverse Transform Sampling Method. The Inverse transform sampling method is a common method to generate random variables from a certain distribution f using its inverted cumulative distribution function F^{-1}.

Theorem 1 (Inverse Transform Sampling Method [Ste87, Theorem 2.1])**.** *Let F be a continuous distribution function on $\mathbb{R}$ with inverse F^{-1} defined as follows:*

$$F^{-1}(u) = \inf\{x : F(x) = u, 0 < u < 1\} \tag{1.12}$$

If $U \sim Uni(0,1)$ (cf. 1.3.1), then $F^{-1}(U)$ has cumulative distribution function F.

Inverse Sampling from a Bernoulli Distribution. Algorithm 1.1 samples a random variable $x \sim Bern(p)$ based on the comparison result of the generated uniform random variable $u \in (0,1)$ and parameter p.

Algorithm: $Algo^{Bern}(p)$

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

Algorithm 1.1: Inverse sampling from a Bernoulli distribution.

Inverse Sampling from a Laplace Distribution. A Laplace random variable $Y \sim Lap(b)$ can be sampled from an exponential distribuion with cumulative distribution function: $F(x \mid b) = 1 - e^{-\frac{x}{b}}$ as follows [Mir12]:

1. Sample random variables $U \sim Uni(0,1) \setminus 1$ and $Z \sim Bern(0.5)$
2. Generate a geometric random variable with $F^{-1}(U) = -b \cdot \ln(1-U)$.
3. Transform the geometric random variable $F^{-1}(U)$ to a Laplace random variable Y as: $Y = (2Z-1) \cdot b \ln(1-U)$

Sampling from a Geometric Distribution. Algorithm 1.2 [Wal74; Tea20b] generates a geometric random variable $x \sim Geo(0.5)$ by generating an ℓ-bit random string $r \in \{0,1\}^\ell$ (i.e., ℓ Bernoulli trials) and counting its leading zeros (i.e., the number of trials before the first success trial). If there is no 1 bit in the ℓ-bit r, the algorithm fails. However, we can decrease the failure probability (0.5^ℓ) by increasing the length of the random string r.

Algorithm: $Algo^{Geo}(0.5)$

Input: 0.5
Output: $x \sim Geo(0.5)$
1 : $x \leftarrow 0$
2 : $r \leftarrow\!\!\$\ \{0,1\}^\ell$
3 : $x \leftarrow LeadingZeros(r)$
4 : **RETURN** x

Algorithm 1.2: Sampling from a geometric distribution.

Sampling from a Discrete Laplace Distribution. Algorithm 1.3 [EKM+14] generates a discrete Laplace random variable $x \sim DLap(t)$ by transforming two independent uniform random variables $u1, u2 \in (0,1)$ as follows:

Algorithm: $Algo^{DLap\_EKMPP}(t)$

Input: t
Output: $x \sim DLap(t)$
1 : $u_1 \leftarrow Uni(0,1)$
2 : $u_2 \leftarrow Uni(0,1)$
3 : **RETURN** $x \leftarrow \lfloor -t \cdot \ln(u_1) \rfloor - \lfloor -t \cdot \ln(u_2) \rfloor$

Algorithm 1.3: Sampling from a discrete Laplace distribution.

1.3.2 Traditional Techniques for Privacy Preservation

Before differential privacy [Dwo06; DMNS06] became the leading method for controlling information disclosure, researchers had proposed approaches for privacy preservation. We briefly introduced several techniques with an adapted example from work [LLV07].

Suppose a fictitious hospital has collected massive data from thousands of patients and wants to make the data available to academic researchers. However, the data contains sensitive information of the patients, such as Zip Code, Age, Nationality, and Health Condition. Because the hospital has an obligation, e.g., due to the EU General Data Protection Regulation

(GDPR) [VV17], to preserve the privacy of the patients, it must carry specific privacy preservation measures before releasing the data to academic researchers. Let us assume that the released data is already anonymized by removing the identifying features such as the name and social security number of the patients as Tab. 1.3 shows. The remaining 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. However, the adversary can still identify the patient and discover his Condition by combining the records with other publicly available information.

	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 1.3: Inpatient microdata [MKGV07].

k-anonymity. A typical attack is the re-identification attack [Swe97] that combines the released anonymized data with publicly available information to re-identify individuals. One approach against such re-identification attacks is to deploy privacy preservation methods that satisfy the notion of *k-anonymity* [SS98]. 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. Samarati et al. [SS98] introduced two techniques to achieve *k-anonymity*: data generalization and data suppression. The former makes the data less informative by mapping the attribute values to a broader value range, and the latter removes specific attribute values. As Tab. 1.4 shows, the values of Age in the first eight records are replaced by the value ranges such as < 30 and ≥ 40 after generalization. The values of Nationality are suppressed by being replaced with $*$. Finally, the records in Tab. 1.4 satisfy the *4-anonymity* requirement. For example, given one patient's non-sensitive attribute values (e.g., Zip Code: $130**$ and Age: < 30), there are at least three other patients with the same non-sensitive attribute values.

	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 1.4: $4-anonymous$ inpatient microdata [MKGV07].

l-Diversity. Although *k-anonymity* alleviates re-identification attacks, it is still vulnerable to the so-called homogeneity attacks and background knowledge attacks [MKGV07]. Suppose we know one patient about thirty years old has visited the hospital, and his record appears in Tab. 1.4, then we could conclude that he has cancer. Afterward, the notion *l-Diversity* [MKGV07] was proposed to overcome the shortcoming of *k-anonymity* by preventing the homogeneity of sensitive attributes in the *equivalent* classes. Specifically, *l-Diversity* requires that there exist at least l different values for the sensitive attribute in each equivalent class. As Tab. 1.5 shows, an adversary cannot infer if a man (with Zip Code: $1305*$ and Age 31) has cancer or viral infection, even though it is very unlikely for a man who is less than forty years old to have heart disease.

t-closeness. However, the definition of *l-Diversity* was later proved to suffer from attacks by Li et al. [LLV07], who introduced the concept of *t-closeness* as an enhancement of *l-Diversity*. *t-closeness* requires that the distance between the distribution of the sensitive attributes in each equivalent class and the distribution of the sensitive attributes in the whole table to be less than a given threshold. The distance can be measured by the Kullback-Leibler distance [KL51] or Earth Mover's distance [RTG00]. However, Li et al. [LLV09] showed that *t-closeness* significantly affects the quantity of the valuable information in the released data.

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	1305∗	≤ 40	*	Heart Disease
4	1305∗	≤ 40	*	Viral Infection
9	1305∗	≤ 40	*	Cancer
10	1305∗	≤ 40	*	Cancer
5	1485∗	> 40	*	Cancer
6	1485∗	> 40	*	Heart Disease
7	1485∗	> 40	*	Viral Infection
8	1485∗	> 40	*	Viral Infection
2	1306∗	≤ 40	*	Heart Disease
3	1306∗	≤ 40	*	Viral Infection
11	1306∗	≤ 40	*	Cancer
12	1306∗	≤ 40	*	Cancer

Table 1.5: $3-diverse$ inpatient microdata [MKGV07].

Instead of releasing anonymized data, a more promising method is to limit the data analyst's access by deploying a 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, it is non-trivial to build such a privacy-preserving system. 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 query result, releasing approximate statistics may protect privacy to some extend. However, Dinur and Nissim [DN03] demonstrated that the adversary could still reconstruct the entire database when sufficient queries were allowed, and the approximate statistics error was bound to a certain level. Therefore, there are fundamental limits between what privacy protection can achieve and what useful statistical information the queries can provide. Differential privacy [Dwo06; DMNS06] is a robust definition that supports quantitative analysis of the amount of useful statistical information to be released while preserving a desired level of privacy.

1.3.3 Differential Privacy Formalization

Randomized Response

In this part, we introduce differential privacy with a very early differentially private algorithm, the Randomized Response [War65] using an example adapted from [Kam20]. Suppose a psychologist wishes to study the psychological impact of cheating on high school students. The psychologist needs to find out the number of students who have cheated. Undoubtedly, most students would not admit if they had cheated in exams. Suppose 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 *cheated*. Every student want to keep their sensitive information X_i secret, but they still need to answer whether they have cheated. Then, each student sends the psychologist an answer Y_i that equals to X_i or a random bit. Finally, the psychologist collects all the answers and estimates the fraction of cheating students $\overline{C} = \frac{1}{n}\sum_{i=1}^{n} X_i$.
The students can apply following strategy:

$$Y_i = \begin{cases} X_i & \textit{with probability } p \\ 1 - X_i & \textit{with probability } 1-p \end{cases} \tag{1.13}$$

where p is the probability that a student honestly answers the question.

Suppose all students use the same strategy and answer 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. Hence, students have to take another strategy by setting $p = \frac{1}{2}$, i.e., each student answers honestly or lies with the same probability. Note that the answer Y_i does not depend on X_i any more and the psychologist could not infer anything about X_i. Further, $\frac{1}{n}\sum_{i=1}^{n} Y_i$ is a binomial random variable $\frac{1}{n}\sum_{i=1}^{n} Y_i \sim \frac{1}{n} \cdot Binomial\left(n, \frac{1}{2}\right)$ and completely independent of $\overline{C}$.

So far, we have explored two strategies: the first strategy ($p = \{0,1\}$) leads to an accurate answer but is not privacy-preserving, while the second strategy ($p = \frac{1}{2}$) is perfectly private but delivers useless information. 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. The students can increase their privacy protection level by setting $\gamma \to 0$ or provide more accurate result by increasing $\gamma \to \frac{1}{2}$.

To measure the accuracy of the strategy, we start with the Y_i's expectation $\mathbb{E}[Y_i] = 2\gamma X_i + \frac{1}{2} - \gamma$, then we obtain $\mathbb{E}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right] = X_i$. Next, we compute the sample mean $\tilde{C} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right]$, where $\mathbb{E}\left[\tilde{C}\right] = \overline{C}$. The variance of $\tilde{C}$ is computed as follows:

$$Var\left[\tilde{C}\right] = Var\left[\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right]\right] = \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i]. \tag{1.14}$$

As $Y_i \sim Bern(p)$, we have $Var[Y_i] = p(1-p) \leq \frac{1}{4}$ and

$$\begin{aligned} \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i] &= \frac{1}{4\gamma^2 n} Var[Y_i] \\ &\leq \frac{1}{16\gamma^2 n}. \end{aligned} \tag{1.15}$$

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

$$\Pr(|X-\mu| \geq t) \leq \frac{\sigma^2}{t^2}, \tag{1.16}$$

For $t = O\left(\frac{1}{\gamma\sqrt{n}}\right)$, we have

$$\begin{aligned} \Pr\left(\left|\tilde{C}-\overline{C}\right| \geq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\leq O(1), \\ \Pr\left(\left|\tilde{C}-\overline{C}\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\geq O(1), \end{aligned} \tag{1.17}$$

and $\left|\tilde{C}-\overline{C}\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)$ with high probability. Note that the error $\left|\tilde{C}-\overline{C}\right| \to 0$ as $n \to \infty$ with high probability. Therefore, to reduce the error, we need to either decrease the privacy protection level $\gamma \to 0.5$ or increase the number of students n.

Terms and Definitions

We adapted the terms and definitions from [DR+14] to formalize the definition of differential privacy.

Database. The database D consists of n entries of data from a data universe $\mathcal{X}$ and is denoted by $D \in \mathcal{X}^n$. In the following, we will use the words database and dataset interchangeably. The database in Tab. 1.6 contains five students' names and exam scores. The database is represented by its rows, and 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 1.6: Database of five students.

Data Curator. A data curator is trusted to manage and organize the database. Its primary goal is to ensure that the database can provide useful information and preserve privacy after multiply queries. The curator can be simulated with cryptographic protocols such as SMPC [GMW87].

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 1.3.10 (Mechanism [DR+14]). *A 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 mechanism for the data analyst/adversary to query. After the querying, the data analyst/adversary receives an output.

Figure 1.1: Query process of a database.

Definition 1.3.11 (Neighboring Databases [DR+14]). *Two databases D_0, $D_1 \in \mathcal{X}^n$ are called neighboring databases if they differ in exact one entry, which is expressed as $D_0 \sim D_1$.*

Definition 1.3.12 (Differential Privacy [DR+14]). *A mechanism $M : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$ is said to satisfy (ε, δ)-differential privacy if for any two neighboring databases D_1, $D_1 \in \mathcal{X}^n$, and for all $T \subseteq \mathcal{Y}$,*

$$\Pr[M(D_0) \in T] \leq e^{\varepsilon} \cdot \Pr[M(D_1) \in T] + \delta,$$

where the randomness is over the choices made by M.

Intuitively, differential privacy implies that the output distribution of mechanism M is similar for all neighboring databases. M is called ε-DP (or pure DP) for $\delta = 0$, and (ε, δ)-DP (or approximate DP) for $\delta \neq 0$.

Definition 1.3.13 (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 $\vec{X}$ and is denoted by $\left\|\vec{X}\right\|_1 = \sum_{i=1}^{n} |x_i|$.*

Definition 1.3.14 (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 is denoted by $\left\|\vec{X}\right\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$.*

Definition 1.3.15 (ℓ_t-sensitivity [DR+14]). *The ℓ_t-sensitivity of a query function $f : \mathcal{X}^n \to \mathbb{R}^k$ is defined as $\Delta_t^{(f)} = \max_{D_0, D_1} \|f(D_0) - f(D_1)\|_t$ for $t \in \{1, 2\}$, where D_0, D_1 are neighboring databases.*

Generally, the sensitivity calculates the upper bound of how much a query function f can change when modifying a single entry using the notion of neighboring databases.

Motivating Example of Differential Privacy

The previous example about randomized response § 1.3.3 indicates that we can use DP to solve the trade-off problem between learning useful statistics and preserving the individuals' privacy. To illustrate how DP solves such problems, we adapt an example [1] shown in Prot. 1.1:

- An adversary proposes two datasets D_0 and D_1 that differ by exactly one entry and a test set Q. A challenger implements a mechanism M that can compute useful statistical information with databases D_0 and D_1.
- The challenger outputs $M(D_0)$, $M(D_1)$ to the adversary in a random order. The adversary aims to differentiate D_0 and D_1. The challenger's goal is to build a mechanism M to prevent $M(D_0)$ and $M(D_1)$ from being distinguished by the adversary.
- Mechanism M is called ε-DP 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
	$\xleftarrow{D_0, D_1}$	
$b \leftarrow\$ \{0,1\}$		
$M(D_b), M(D_{1-b})$		
	$\xrightarrow{M(D_b), M(D_{1-b})}$	
		$b' \leftarrow 0$, **if** $M(D_{1-b}) \in Q$
		$b' \leftarrow 1$, **otherwise**
		if $b == b'$, A **wins**.

Protocol 1.1: A example of differentially privace mechanism.

Suppose the adversary A has chosen the following databases:

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

The test set Q is an interval $[T, 1]$ with a threshold T. After choosing the threshold T, the adversary output $b' \leftarrow 0$ when $T < M(D_{1-b}) < 1$, or $b' \leftarrow 1$ when $0 < M(D_{1-b}) \leq T$. The

[1] https://win-vector.com/2015/10/02/a-simpler-explanation-of-differential-privacy/

adversary's goal is to find a *good* threshold T that helps him output a b' that equals b with high probability and win the game.

The Deterministic Case. Suppose mechanism M compute the mean value of the given databases D_0 and D_1, where $M(D_0) = 0$ and $M(D_1) = 0.01$. The adversary can set the test set $Q = [0.005, 1]$ win every game. In Fig. 1.2, the blue line represents the value of $M(D_0)$, whereas the orange line represents the value of $M(D_1)$ (plotted upside down for clarity). The vertical dotted line is the threshold $T = 0.005$ that separates D_0 and D_1 correctly.

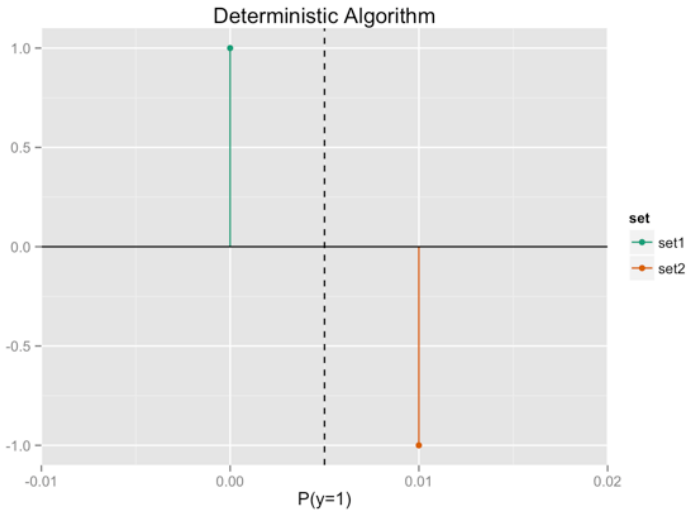

Figure 1.2: Deterministic algorithm.

The Indeterministic Case. The challenger takes some measures to *blur* the outputs of $M(D_0)$ and $M(D_1)$ and make them hard to differentiate. Suppose the challenger decides to add the Laplace noise $lap \sim Laplace(b = 0.05)$ to the result of $M(D)$ as Fig. 1.3 shows. The shaded blue region is the probability that $M(D_0)$ returns a value greater than the threshold T, and the adversary mistakes D_0 for D_1. In contrast, the shaded orange area is the probability that the adversary identify D_1 successfully. The challenger can decrease the adversary's success probability by adding stronger noise as Fig. 1.4 shows, where the shaded blue and orange areas are almost of the same size. In fact, we have $\varepsilon = \left|\ln\left(\frac{blue\ area}{orange\ area}\right)\right|$, where ε describes the degree of differential privacy. A smaller ε indicates a stronger privacy protection. Note that the challenger can add more noise to decrease the adversary's success probability, but the accuracy of the mean estimation decreases.

TODO: need reproduce following figures

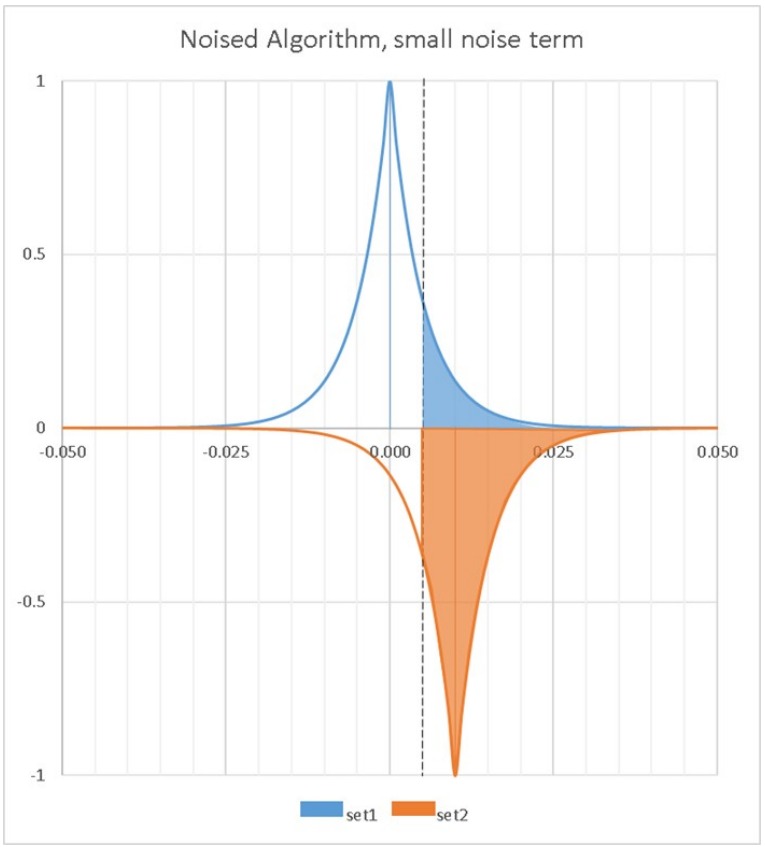

Figure 1.3: Indeterministic algorithm with small noise ($b = 0.005$).

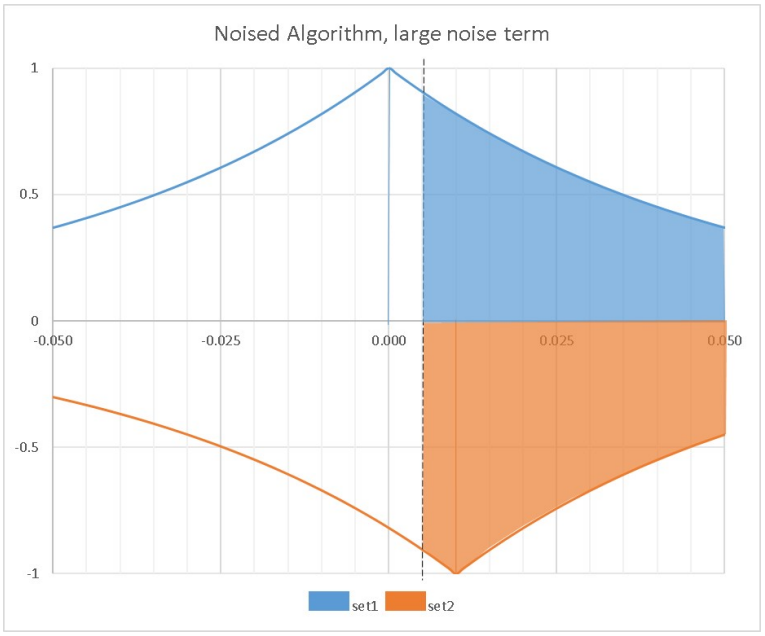

Figure 1.4: Indeterministic algorithm with large noise ($b = 0.05$).

Differentially Private 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 two common differentially private mechanisms.

Definition 1.3.16 (Laplace Mechanism [DR$^+$14]). *Let $f : \mathcal{X}^n \to \mathbb{R}^k$. The Laplace mechanism is defined as $M_{Lap}(X) = f(X) + (Y_1, \ldots, Y_k)$, where the Y_i are independent Laplace random variables drawn from a Laplace distribution $Lap(Y_i \mid b) = \frac{1}{2b} e^{\left(-\frac{|Y_i|}{b}\right)}$ with $b = \frac{\Delta_1^{(f)}}{\varepsilon}$.*

Theorem 2. *The Laplace Mechanism satisfies ε-DP [DR$^+$14].*

Definition 1.3.17 (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 a Gaussian distribution $\mathcal{N}\left(Y_i \mid \mu, \sigma^2\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{Y_i - \mu}{\sigma}\right)^2}$ with $\mu = 0$, $\sigma^2 = 2 \cdot \ln\left(\frac{1.25}{\delta} \cdot \left(\frac{\Delta_2^{(f)})^2}{\varepsilon^2}\right)\right)$.*

Theorem 3. *The Gaussian mechanism satisfies (ε, δ)-DP [DR$^+$14].*

Properties of Differential Privacy

We introduce three fundamental properties of DP [Dwo06; DMNS06] based on work [DR+14].

Proposition 1 (**Post-Processing**). *Let $M : \mathcal{X}^n \to \mathcal{Y}$ be a (ε, δ)-DP mechanism, and let $F : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary randomized mapping. Then $F \circ M$ is (ε, δ)-DP.*

The post-processing property implies that once a database is privatized, it is also differentially private after further processing.

Proposition 2 (**Group Privacy**). *Let $M : \mathcal{X}^n \to \mathcal{Y}$ be a (ε, δ)-DP mechanism. For all $T \subseteq \mathcal{Y}$,*

$$\Pr[M(D_0) + T] \leq e^{k\varepsilon} \cdot \Pr[M(D_1) \in T] + \delta,$$

where D_0, $D_1 \in \mathcal{X}^n$ are databases that differ in exactly k entries.

The group privacy property indicates that DP can also be extended to the case when two databases have more than one entry difference. However, as k increases, the privacy decay rate $e^{k\varepsilon}$ also increases, which implies a weaker level of privacy protection.

Proposition 3 (**Basic Composition**). *Suppose $M = (M_1 \dots M_k)$ is a sequence of $(\varepsilon_i, \delta_i)$-DP 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.*

The basic composition property provides a way to evaluate the overall privacy guarantee of the released result when k DP mechanisms are applied to the same dataset.

Challenges of Differential Privacy

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

Sensitivity Calculation. As discussed in [**subsubsec:DPMechanisms**], we first need to compute or estimate the sensitivity of the query function before applying DP mechanism. Take a database with human ages as an example, the ages should be bounded in the interval $[0, 150]$ (the 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 solution is to roughly estimate a *reasonable* value range and limit the data within that range. If the value range is chosen too wide (i.e., a larger sensitivity estimation), the DP mechanism applies a too strong noise to perturb the data, and the utility of the result might be destroyed. Nevertheless, a narrow value range estimation may also decrease the utility as too much data beyond the value range is discarded.

Security Issue in the Practical Implementation. Generally, the security analysis of differentially private mechanisms is based on two implicit assumptions: (i) Computations are performed on real numbers and require machines to have infinite precision, (ii) The noise is sampled from a probability distribution that is very close to the theoretically correct probability distribution. However, the practical implementation of differentially private mechanisms is typically based on floating-point or fixed-point arithmetic that only provides finite order of accuracy. Mironov [Mir12] demonstrated that the Laplace random noise sampled with textbook algorithm paragraph 1.3.1 under floating-point implementation could lead to violation of differential privacy. Specifically, for database D and a Laplace Mechanism M that samples the Laplace noise with paragraph 1.3.1, $M(D)$ fails to output certain floating-point numbers that can be related to database D. By comparing the missed out floating-point numbers, an adversary can differentiate between database D and its neighboring database D' even extract the entire content of database D. Based on the similar vulnerability of floating-point numbers, Jin et al. [JMRO22] presented a series of floating-point attacks against the Gaussian noise sampling algorithms. In addition, they also constructed a timing attack against the discrete noise sampling algorithms, i.e., the magnitude of the discrete noise can be predicted by measuring the sampling time. Gazeau et al. [GMP16] proved that any differentially private mechanisms perturb data by adding noise with a finite precision could lead to secret disclosure regardless of the actual implementation. This work aims to realize DP mechanisms with secure noise generation methods in SMPC that are free of the above attacks.

2 Secure Differentially Private Mechanisms On Finite Computer

In this chapter, we describe five existing differentially private mechanisms and implementations that (or after modification) are free from the attacks [Mir12; JMRO22] discussed in (cf. paragraph 1.3.3). These differentially private mechanisms are:

1. Snapping Mechanism [Mir12]
2. Integer-Scaling Laplace Mechanism [Tea20b]
3. Integer-Scaling Gaussian Mechanism [Tea20b]
4. Discrete Laplace Mechanism [GRS12; CKS20]
5. Discrete Gaussian Mechanism [CKS20]

We modify the secure noise generation algorithms of these mechanisms such that they can be adapted into SMPC protocols (cf. § 3).

2.1 Snapping Mechanism

Recall that the Laplace mechanism (cf. 1.3.16) satisties ε-DP by adding a Laplace random variable $Y \sim Lap(\lambda)$ to the query function $f(D)$ for database D and $\lambda = \frac{\Delta_1^{(f)}}{\varepsilon}$:

$$M(D, \lambda) = f(D) + Y. \tag{2.18}$$

As discussed in paragraph 1.3.1, we can generated a Laplace random variable Y by transforming a random chosen sign $S \in \{-1, 1\}$ and a uniform random variable $U \in (0, 1]$ (or $U \in (0, 1)$, if we ignore the very *small* probability of generating exact 1) as follows:

$$Y \leftarrow S \cdot \lambda \ln(U) \tag{2.19}$$

Mironov [Mir12] showed that the Laplace random variable Y generated in such method under floating-point arithmetic could lead to severe differential privacy breaching and proposed the snapping mechanism that avoided such security issues by rounding and smoothing the output $f(D) + Y$ in a specific approach.

The snapping mechanism [Mir12] is defined as follows:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)\right\rceil_\Lambda\right). \tag{2.20}$$

Let $\mathbb{D}$ denote the set of floating-point numbers, and $\mathbb{D} \cap (a, b)$ denote all the floating-point numbers in the interval (a, b). $f(D) \in \mathbb{D}$ is a query function with database D, and $S \otimes \lambda \otimes \text{LN}(U^*)$ is the noise term. S is the sign of the noise and uniformly distributed over $\{-1, 1\}$. U^* is a *uniform* distribution over $\mathbb{D} \cap (0, 1)$, and generates floating-point numbers with a probability proportional to its *unit in the last place* (ulp), i.e., spacing between two consecutive floating-point numbers. $\text{LN}(x)$ is the natural logarithm operation under floating-point implementation with exact rounding, i.e., $\text{LN}(x)$ rounds input x to the closest floating-point number with probability $p = 1$. $\oplus$ and $\otimes$ are the floating-point implementations of addition and multiplication. Function $\text{clamp}_B(x)$ limits the output to the interval $[-B, B]$ by outputting B if $x > B$, $-B$ if $x < -B$, and x otherwise. Parameter 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 $n \in \mathbb{Z}$. Function $\lfloor x \rceil_\Lambda$ rounds the floating-point input x exactly to the nearest multiple of Λ by manipulating its binary representation.

The snapping mechanism assumes that the *sensitivity* $\Delta_1^{(f)}$ (cf. 1.3.15) of query function f is 1, which can be extended to an arbitrary query function f' with *sensitivity* $\Delta_1^{(f')} \neq 1$ by scaling the output of $f'(D)$ with $f(D) = \frac{f'(D)}{\Delta_1^{(f')}}$.

Theorem 4 ([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 $\lambda < B < 2^{46} \cdot \lambda$.*

The computation of the snapping mechanism consists of the following steps:

1. Computation of function $\text{clamp}_B(\cdot)$ and function $\lfloor \cdot \rceil_\Lambda$.
2. Generation of a random uniform random variable U^* and a random sign S.
3. Execution of floating-point arithmetic operations, such as $\text{LN}(\cdot)$, addition, and multiplication.

We discuss and describe how to generate uniform random variable U^*. The implementation details of other steps can be found in the Covington's work [Cov19].

Generation of Uniform Random Variable. U^* is a *uniform* distribution over $\mathbb{D} \cap (0, 1)$ and can be represented in IEEE 754 floating-point as:

$$U^* = (1.d_1 \ldots d_{52})_2 \times 2^{e-1023}. \tag{2.21}$$

As discussed above, floating-point U^* is sampled with a probability proportional to its ulp. We sample such floating-point numbers using Algorithm 2.1 [Wal74; Mir12]. Specifically, we independently sample a geometric random variable $x \sim Geo(0.5)$ with Algorithm 1.2 and significant bits $(d_1, \ldots, d_{52}) \in \{0,1\}^{52}$, and compute U^*'s biased exponent $e = 1023 - (x+1)$.

Algorithm: $Algo^{RandFloat1}$

Input: None

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

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

2 : $x \leftarrow Geo(0.5)$

3 : $e \leftarrow 1023 - (x+1)$

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

Algorithm 2.1: Sampling uniform random floating-point number $U^* \in \mathbb{D} \cap (0,1)$.

Intuitively, *uniformly* sampling a floating-point number $U^* \in \mathbb{D} \cap (0,1)$ can be considered as randomly drawing a real number in the interval $(0,1)$ 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} floating-point numbers in the interval $[.5,1)$ and 2^{52} floating-point numbers in the interval $[.25,.5)$. If we only sample the floating-point numbers with equal distance to each other, a large amount of floating-point numbers would be ignored. As discussed in works [Wal74; Mir12], a better sampling approach is to sample floating-point numbers with probability proportional to its ulp (i.e., spacing to its consecutive neighbor). Since U^*'s significant bits are sampled randomly from $\{0,1\}^{52}$, the floating-point numbers with identical biased exponent $e-1023$ are distributed uniformly in the interval $[0,1)$. Using a geometric random variable $x \sim Geo(0.5)$ as the unbiased exponent $e-1023 = -(x+1)$ guarantees that the probability of sampling a floating-point number from $[0,1)$ is proportional to its ulp. Then we discuss the correctness of the sampling approach. The total sampling probability for the floating-point numbers in the interval $(0,1)$ with exponent $e-1023 = -(x+1) = -1$ is $Pr(x=0 \mid 0.5) = \frac{1}{2}$. The total sampling probability for the floating-point numbers in the interval $(0,0.5)$ with exponent $e-1023 = -2$ is $Pr(x=1 \mid 0.5) = \frac{1}{2^2}$, etc. Therefore, the total sampling probability for the floating-point numbers with arbitrary exponent in the interval $[0,1)$ is $\sum_{i=1}^{\infty} \frac{1}{2^i} \approx 1$.

2.2 Integer-Scaling Mechanism

Google Differential Privacy Team [Tea20b] proposed practical implementation framework (this work names it as Integer-Scaling mechanisms) for Laplace mechanism (cf. 1.3.16) and Gaussian mechanism (cf. 1.3.17) without suffering from the floating-point attacks [Mir12; JMRO22]. The basic idea of Integer-Scaling mechanisms is to re-scale a discrete random

variable to simulate the continuous random variable without precision loss. The framework is defined as follows:

$$M_{IS}\left(f\left(D\right), r, \varepsilon, \delta\right) = f_r\left(D\right) + ir, \tag{2.22}$$

where the discrete random variable i is re-scaled by the resolution parameter $r = 2^k$ (for $k \in [-1022, 970]$) to simulate a continuous random variable. Function $f_r\left(D\right) \in \mathbb{D}$ rounds the output of query function $f\left(D\right) \in \mathbb{R}$ to the nearest multiple of r. Resolution r determines the scale of the simulated continuous random variable ir and is predefined based on the requirement. ε, δ are the parameters that define the level of the desired differential privacy protection.

Explanation of Integer-Scaling Mechanisms. First, let us assume $f_r\left(D\right) + ir$ satisfy (ε, δ)-DP under real number arithmetic. Let $+$ denote the addtion under real number and $\oplus$ denote the addtion under floating-point number. If $f_r\left(D\right) \oplus ir$ can be computed *precisely* under floating-point arithmetic, then we can conclude that $f_r\left(D\right) \oplus ir$ also satisfies (ε, δ)-DP. *Precisely* indicates that the real numbers are represented as floating numbers without precision loss, and the arithmetic operations of these real numbers yield exactly the same result as when these real numbers are represented as floating-point numbers. For integer $|i| \leq 2^{52}$ (or $\Pr\left(\left|i > 2^{52}\right|\right)$ is small enough to ignore) and resolution parameter $r = 2^k$ (with $k \in [-1022, 970]$), we have $|ir| \leq 2^{1022}$. Since the exponent of a 64-bit floating-point number ranges from -1022 to 1023, ir can be represented precisely as a floating-point number by adding k to the exponent of integer i. In other words, integer i can be re-scaled by resolution parameter r under floating-point arithmetic without precision loss. Further, by choosing r and limiting $|f\left(D\right)| < 2^{52} \cdot r$, $f_r\left(D\right)$ can be represented precisely as a floating-point number. When real number addition result $f_r\left(D\right) + ir$ is rounded with the IEEE-754 standard, it equals to $f_r\left(D\right) \oplus ir$. According to the post-processing property of DP (cf. 1), the rounding result of $f_r\left(D\right) + ir$ and $f_r\left(D\right) \oplus ir$ both satisfy (ε, δ)-DP. Next, we describe to use the Integer-Scaling mechanisms to realize the Laplace and Gaussian mechanisms.

2.2.1 Integer-Scaling Laplace Mechanism

In this section, we describe the Integer-Scaling Laplace mechanism [Tea20b] and modify the corresponding sampling algorithms.

The Integer-Scaling Laplace mechanism [Tea20b] is defined as:

$$M_{ISLap}\left(f\left(D\right), r, \Delta_r, \varepsilon\right) = f_r\left(D\right) + ir, \tag{2.23}$$

where r is the resolution parameter that controls the scale of the simulated continuous Laplace random variable ir, discrete Laplace random variable $i \sim DLap\left(t = \frac{\Delta_r}{r\varepsilon}\right)$ (cf. 1.3.8), $\Delta_r = r + \Delta_1^{(f)}$, and $\Delta_1^{(f)}$ is the ℓ_1-sensitivity (cf. 1.3.15) of $f\left(D\right)$.

Theorem 5 ([Tea20b]). *The Integer-Scaling Laplace mechanism* $M_{ISLap}(f(D), r, \Delta_r, \varepsilon)$ *satisfies* ε*-DP for query function* f.

The generation of the discrete Laplace random variable i consists of two steps: (i) generation of a geometric random variable x with a binary search-based geometric sampling algorithm, and (ii) transformation from x to the discrete Laplace random variable i.

Binary Search Based Geometric Sampling Algorithm.

Algorithm 2.2 is a modification of the binary search based geometric sampling algorithm from the work [Tea20a] that samples a positive integer x from a geometric distribution $Geo\left(x \mid p = 1 - e^{-\lambda}\right)$ (cf. 1.3.7).

Sampling Interval of x**.** Let us first define the sampling interval of the geometric random variable x. In the original work [Tea20a], the sampling interval for x is $\left[0, 2^{63} - 2\right]$. We limit the sampling interval of x to $\left[0, 2^{52}\right]$ because each integer in this interval can be represented exactly as a 64-bit floating-point number.

Choice of λ**.** Then, we set a reasonable value range for parameter λ (the original work [Tea20b] requires $\lambda > 2^{-59}$). For the geometric distribution's cumulative distribution function: $\Pr(x \leq X \mid x \sim Geo(p)) = 1 - (1-p)^X$ with $X = 2^{52}$, we have

$$\Pr\left(x \leq 2^{52} \mid x \sim Geo(p)\right) = 1 - e^{-\lambda \cdot 2^{52}+1} = \begin{cases} 0.\overline{9}_{(27)}8396\ldots & \text{for } \lambda = 2^{-48} \\ 0.\overline{9}_{(13)}8734\ldots & \text{for } \lambda = 2^{-49} \end{cases} \tag{2.24}$$

where $0.\overline{9}_{(n)}$ denotes a number with n consecutive digits of value 9.

We require $\lambda \geq 2^{-48}$ which means that Algorithm 2.2 fails (when required to generate $x > 2^{52}$) with a probability $p = 1 - 0.\overline{9}_{(27)}8396 \approx 2^{-92}$.

Algorithm Description. Algorithm 2.2 first splits the sampling interval into two subintervals that have an almost equal cumulative probability. Then, one subinterval is chosen randomly, and the other is discarded. This splitting and choosing process is repeated until the remaining sampling interval only contains one value (i.e., the geometric random variable x we are looking for). In the original work [Tea20b], the sampling algorithm additional checks if the generated random variable exceeds the sampling interval $\left[0, 2^{63} - 2\right]$, which is not necessary in our case because we set $\lambda > 2^{-48}$ such that the exceeding happens with a very low probability $p \approx 2^{-92}$.

In Line 1, we set the initial sampling interval to $\left[0, 2^{52}\right]$. In Line 3, the sampling interval is splitted with function $\text{Split}(L, R, \lambda) = L - \frac{\ln(0.5)+\ln\left(1+e^{-\lambda(R-L)}\right)}{\lambda}$ into subintervals $[L \ldots M]$ and $(M \ldots R]$, such that they have approximately equal cumulative probability (i.e., $Pr(L \leq x \leq M \mid x \sim Geo) \approx Pr(M < x \leq R \mid x \sim Geo)$). Lines $4-7$ ensure that the split point M lies in the interval $[L \ldots R]$ (or relocates M to the interval $[L \ldots R]$ otherwise).

Line 8 calculates the cumulative probability proportion Q between interval $[L \dots M]$ and $(M \dots R]$ with function $\text{Proportion}(L,R,M,\lambda) = \frac{e^{-\lambda(M-L)}-1}{e^{-\lambda(R-L)}-1}$. In line $9-13$, we randomly choose one interval based on the comparison result of a uniform variable U (generated with Algorithm 2.1) and Q. If $U \leq Q$, we choose interval $[L \dots M]$ as the next sampling interval, $(M \dots R]$ otherwise. In other words, the probability that an interval is chosen is proportional to its cumulative probability. The whole process is repeated (at most 52 times) until the remaining interval contains only one value (condition in Line 2 is not satisfied). Finally, we set $x \leftarrow R-1$, where $x \sim Geo\left(p = 1-e^{-\lambda}\right)$.

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

Input: λ

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

1: $L \leftarrow 0, R \leftarrow 2^{52}$
2: **WHILE** $L+1 < R$
3: $\quad M \leftarrow \text{Split}(L,R,\lambda)$
4: $\quad$ **IF** $M \leq L$
5: $\quad\quad M \leftarrow L+1$
6: $\quad$ **ELSE IF** $M \geq R$
7: $\quad\quad M \leftarrow R-1$
8: $\quad Q \leftarrow \text{Proportion}(L,R,M,\lambda)$
9: $\quad U \leftarrow Uni(0,1)$
10: $\quad$ **IF** $U \leq Q$
11: $\quad\quad R \leftarrow M$
12: $\quad$ **ELSE**
13: $\quad\quad L \leftarrow M$
14: **RETURN** $x \leftarrow R-1$

Algorithm 2.2: Sampling from a geometric distribution $Geo\left(p = 1-e^{-\lambda}\right)$ using binary search.

Two-Side Geometric Sampling Algorithm.

In this part, we describe how to transform a geometric random variable $x \sim Geo\left(p = 1-e^{-\lambda}\right)$ into a Laplace random variable $i \sim DLap\left(t = \frac{1}{\lambda}\right)$.

Algorithm Description. Recall that two-side geometric distribution (also known as discrete Laplace distribution) can be generated by reflecting a geometric distribution across the y-axis (cf. 1.3.8). Algorithm 2.3 generates a discrete Laplace random variable $i = s \cdot g$ with this method. Since SMPC needs to determine the number of loops ahead of time, we use the FOR loop instead of the $WHILE$ loop in Line 1. We use Algorithm 2.2 to generate the geometric

random variable in line 2. In Line 4, the case $s \cdot g = (-1) \cdot 0 = -0$ is discarded. Otherwise, value 0 would be returned with twice the probability as in the discrete Laplace distribution. In Line 5, we output the correctly sampled discrete Laplace random variable i. In Line 6, Algorithm 2.3 fails to a generate discrete random variable within $ITER$ loops and output 0.

Algorithm: $Algo^{TwoSideGeo}(t)$

Input: t
Output: $i \sim DLap(t)$
1 : **FOR** $i \leftarrow 1$ **TO** $ITER$
2 : $g \leftarrow Geo\left(\frac{1}{t}\right)$
3 : $s \leftarrow\$ \{-1, 1\}$
4 : **IF** $\neg(s == -1 \wedge g == 0)$
5 : **RETURN** $i \leftarrow s \cdot g$ // success
6 : **RETURN** $i \leftarrow 0$ // failure

Algorithm 2.3: Sampling from a discrete Laplace distribution $DLap(t)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that Algorithm 2.3 fails for $ITER = i$. Since each loop is independent, we estimate $\Pr(A_{ITER})$ as follows:

$$\begin{aligned}
\Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}(\Pr(A_1)) \\
&= \Pi_{i=1}^{ITER}(\Pr(s == -1) \cdot \Pr(g == 0)) \\
&= \Pi_{i=1}^{ITER}\left(\frac{1}{2} \cdot \Pr\left(0 \leftarrow Geo\left(1 - e^{-\lambda}\right)\right)\right) \\
&= \Pi_{i=1}^{ITER}\frac{1}{2}\left(1 - e^{-\lambda}\right) \\
&= \frac{1}{2^{ITER}}\left(1 - e^{-\lambda}\right)^{ITER}.
\end{aligned} \tag{2.25}$$

To guarantee that $\Pr(A_{ITER}) < 2^{-40}$, we have $ITER = 6$ for $\lambda = 0.01$.

2.2.2 Integer-Scaling Gaussian Mechanism

In this section, we describe the Integer-Scaling Gaussian mechanism [Tea20b]:

$$M_{ISGauss}(f(D), r, \varepsilon, \delta) = f_r(D) + ir, \tag{2.26}$$

where r is the resolution parameter that controls the scale of the simulated Gaussian random variable ir. Parameter r and n are estimated with ε and δ as in the works [BW18; Tea20b], where the value range of n is around 2^{128} [Tea20b]. Since 2^{128} is too large to be represented as a 64-bit integer, $\sqrt{n}$ is used in the sampling algorithm.

Generally, $M_{ISGauss}(f(D), r, \varepsilon, \delta)$ uses a symmetrical binomial random variable i (cf. 1.3.6) to simulate a continuous Gaussian random variable $i_{Gau} \sim \mathcal{N}$. The closeness between the ir and i_{Gau} depends on n, i.e., a larger n indicates a better approximation effect.

Theorem 6 ([Tea20b]). *The Integer-Scaling Gaussian mechanism $M_{ISGauss}(f(D), r, \varepsilon, \delta)$ satisfies (ε, δ)-DP for query function f.*

Symmetrical Binomial Sampling Algorithm Algorithm 2.4 is a modification of the symmetrical binomial sampling algorithm [Tea20b], that samples $i \sim SymmBino(n, p = 0.5)$ with input $\sqrt{n}$. We modify the original sampling algorithm by replacing the $WHILE$ loop with a FOR loop (line 2), such that it has fixed round of iterations. If Algorithm 2.4 fails to generate a symmetrical binomial random variable within $ITER$ iterations, it outputs 0 as line 17 shows. We found that when $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2}$, $\tilde{p}(x)$ is greater than 0. Therefore, we replace the IF condition for checking $\tilde{p}(x) > 0$ (in the orignal work) to the condition for checking $\left(-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2} \wedge c == 1\right)$ in line 14. This modification simplifies the construction of SMPC protocols. In line 9, l is a uniform random integer between 0 and $m-1$.

Algorithm: $Algo^{SymmetricBinomial}\left(\sqrt{n}\right)$

Input: $\sqrt{n}$

Output: $i \sim SymmBino\,(n, p = 0.5)$

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

FOR $j \leftarrow 1$ **TO** $ITER$

  $s \leftarrow Geo\,(0.5)$

  $b \leftarrow\$ \{0, 1\}$

  IF $b == 0$

    $k \leftarrow s$

  ELSE

    $k \leftarrow -s - 1$

  $l \leftarrow\$ \{0, \ldots, m-1\}$

  $x \leftarrow km + l$

  $\tilde{p}\,(x) = \sqrt{\frac{2}{\pi n}} \cdot e^{-\frac{2x^2}{n}} \cdot \left(1 - \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}\right)$

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

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

  IF $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2} \wedge c == 1$

    RETURN $i \leftarrow x$ // success

  ELSE

    RETURN $i \leftarrow 0$ // failure

Algorithm 2.4: Sampling from a symmetric binomial distribution $SymmBino\left(\sqrt{n}, p = 0.5\right)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that Algorithm 2.4 fails when $ITER = i$. Bringmann et al. [BKP$^+$14] showed that each iteration has a probability $p = \frac{1}{16}$ to terminate, i.e., $\Pr(A_1) = \frac{15}{16}$. Since each iteration is independent, we compute $\Pr(A_{ITER})$ as follows:

$$\begin{aligned} \Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}\left(\Pr(A_1)\right) \\ &= \left(\frac{15}{16}\right)^{ITER} \end{aligned} \tag{2.27}$$

To guarantee that $Pr\,(A_{ITER}) < 2^{-40}$, we need at least $ITER \geq \log_{\frac{15}{16}}\left(2^{-40}\right) \approx 430$ iterations in the FOR loop.

2.3 Discrete Laplace Mechanism

In this section, we describe the discrete Laplace mechanism [CSS12; GRS12; EKM$^+$14] and present a modified sampling algorithm Algorithm 2.6 for generating discrete Laplace random variable.

The discrete Laplace mechanism is define as:

$$M_{DLap}\left(f\left(D\right), r, \varepsilon\right) = f\left(D\right) + Y, \tag{2.28}$$

where query function $f\left(D\right) \in \mathbb{Z}$ and $Y \sim DLap\left(t = \frac{\Delta_1^{(f)}}{\varepsilon}\right)$.

Theorem 7 ([CSS12; GRS12; EKM$^+$14])**.** *The discrete Laplace mechanism $M_{DLap}\left(f\left(D\right), r, \varepsilon\right)$ satisfies ε-DP for query function $f\left(D\right) \in \mathbb{Z}$.*

Except the previous introduced discrete Laplace sampling algorithm Algorithm 1.3, Canonne et al. [CKS20] proposed a discrete Laplace sampling algorithm that is based on rejection sampling method [CRW04]. The algorithm first generates a random geometric random variable and then converting it to a discrete Laplace random variable.

Rejection Sampling Based Geometric Sampling Algorithm. Algorithm 2.5 is a modificaiton of the geometric sampling algorithm [CKS20], that samples an integer $x \sim Geo\left(p = 1 - e^{-\frac{n}{d}}\right)$, where n, d are positive integers.

Algorithm Description. We replace the two $WHILE$ loops in the original work with two FOR loops (line $4-8$ and line $9-14$) and add a IF condition in line 15 to detect if the algorithm fails. Algorithm 2.5 consists of two FOR loops and check if these FOR loops terminate within $ITER_1$ and $ITER_2$ iterations. If they both terminate, the algorithm succeeds, and fails otherwise. One special case is when $d = 1$, the 1th FOR loop terminates without execution. We use Algorithm 1.1 to sample the Bernoulli random variable in line 6 and 10.

Algorithm: $Algo^{GeoExp}(n,d)$

Input: n, d

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

1: $k \leftarrow -1$,
2: **IF** $d == 1$
3: **GOTO** Line 9// 1th loop terminates
4: **FOR** $j \leftarrow 1$ **TO** $ITER_1$
5: $u \leftarrow_{\$} \{0, \ldots, d-1\}$
6: $b_1 \leftarrow Bern\left(e^{-\frac{u}{d}}\right)$
7: **IF** $b_1 == 1$
8: $k \leftarrow 0$, **BREAK** // 1th loop terminates
9: **FOR** $j \leftarrow 1$ **TO** $ITER_2$
10: $b_2 \leftarrow Bern\left(e^{-1}\right)$
11: **IF** $b_2 == 1$
12: $k \leftarrow k + 1$
13: **ELSE**
14: **BREAK** // 2nd loop terminates
15: **IF** $b_1 == 0 \wedge b_2 == 1$
16: **RETURN** 0 // failure
17: **ELSE**
18: **RETURN** $x \leftarrow \left\lfloor \frac{k \cdot d + u}{n} \right\rfloor$ // success

Algorithm 2.5: Rejection sampling from a geometric distribution $Geo\left(p = 1 - e^{-\frac{n}{d}}\right)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that the first FOR loop (line 4-8) not terminates when $ITER_1 = i$ iterations, and B_i is an event that the second FOR loop (line 9-14) not terminates for $ITER_2 = i$ iterations. To guarantee that Algorithm 2.5 fails with a probability less than 2^{-40}, we first compute $\Pr(A_1)$ and $\Pr(B_1)$ as follows:

$$\begin{aligned}\Pr(A_1) &= \sum_{i=0}^{d-1} \Pr(u=i)\cdot\Pr(b_1=0)\\ &= \sum_{i=0}^{d-1} \frac{1}{d}\cdot\left(1-e^{-\frac{i}{d}}\right)\\ &= 1-\frac{1}{d}\sum_{i=0}^{d-1} e^{-\frac{i}{d}}\\ &= 1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\end{aligned} \tag{2.29}$$

$$\begin{aligned}\Pr\left(A_{ITER_1}\right) &= \prod_{i=1}^{ITER_2} \Pr(A_1)\\ &= \left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}\end{aligned} \tag{2.30}$$

$$\begin{aligned}\Pr\left(B_{ITER_2}\right) &= \prod_{i=1}^{ITER_2} \Pr(B_1)\\ &= \prod_{i=1}^{ITER_2} \Pr(b_2=1)\\ &= e^{-ITER_2}\end{aligned} \tag{2.31}$$

As event A_{ITER_1} and B_{ITER_2} are independent,

$$\begin{aligned}\Pr(Algorithm\ 2.5) &= \Pr\left(A_{ITER_1}\vee B_{ITER_2}\right)\\ &= \Pr\left(A_{ITER_1}\right)+\Pr\left(B_{ITER_2}\right)-\Pr\left(A_{ITER_1}\right)\cdot\Pr\left(B_{ITER_2}\right)\\ &= \left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}+e^{-ITER_2}-\left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}\cdot e^{-ITER_2}\end{aligned} \tag{2.32}$$

To guarantee $\Pr(Algorithm\ 2.5) < 2^{-40}$, we have:

(i) $ITER_1 = 27$, $ITER_2 = 28$, when $n = 3$ and $d = 2$, (ii) $ITER_1 = 0$, $ITER_2 = 28$, when $n = 3$ and $d = 1$.

Discrete Laplace Sampling Algorithm. Algorithm 2.6 is a modificaiton of the discrete Laplace sampling algorithm from the work [CKS20] that samples $Y \sim DLap\left(t = \frac{d}{n}\right)$ (cf. 1.3.8), where d and n are positive integers.

Algorithm Description. Algorithm 2.6 is based on the same idea as Algorithm 2.3 but uses Algorithm 2.5 to generate the geometric random variable in line 3.

Algorithm: $Algo^{DLap}\left(t = \frac{d}{n}\right)$

Input: $t = \dfrac{d}{n}$

Output: $Y \sim DLap\left(t = \dfrac{d}{n}\right)$

1 : **FOR** $j = 1$ **TO** $ITER$
2 : $s \leftarrow\$ \{0,1\}$
3 : $m \leftarrow Geo\left(p = 1 - e^{-\frac{n}{d}}\right)$
4 : **IF** $\neg(s == 1 \wedge m == 0)$
5 : **RETURN** $Y \leftarrow (1-2s)\cdot m$ // success
6 : **RETURN** $Y \leftarrow 0$ // failure

Algorithm 2.6: Sampling from a discrete Laplace distribution $DLap\left(t = \frac{n}{d}\right)$.

Algorithm Fail Probability Estimation. Note that we need to guarantee that both Algorithm 2.6 and Algorithm 2.5 fail with a probability less than 2^{40}. Suppose A_i is an event that Algorithm 2.6 fails in $ITER_1 = i$ iterations. As $\Pr(s == 1)$ and $\Pr(m == 0)$ are independent of each other, we have:

$$\begin{aligned}\Pr(A_1) &= \Pr(s == 1)\cdot\Pr(m == 0 \wedge Algorithm\ 2.5\ \text{successes}) \\ &\quad + \Pr(m == 0 \wedge Algorithm\ 2.5\ \text{fails}) \\ &= \frac{1}{2}\cdot\Pr(m == 0 \wedge Algorithm\ 2.5\ \text{successes}) + \Pr(Algorithm\ 2.5\ \text{fails}) \\ &= \frac{1}{2}\cdot\left(1 - e^{-\frac{n}{d}}\right) + \Pr(Algorithm\ 2.5\ \text{fails})\end{aligned} \tag{2.33}$$

Finally, we have:

$$\begin{aligned}\Pr(Algorithm\ 2.6\ \text{fails}) &= \Pr(A_{ITER}) \\ &= \prod_{i=1}^{ITER}\Pr(A_1) \\ &= \left(\frac{1}{2}\cdot\left(1 - e^{-\frac{n}{d}}\right) + \Pr(Algorithm\ 2.5\ \text{fails})\right)^{ITER}\end{aligned} \tag{2.34}$$

To guarantee Algorithm 2.6 and Algorithm 2.5 fail with a probability $p < 2^{-40}$, it requires that $ITER = 30$ for $n = 3$ and $d = 2$.

2.4 Discrete Gaussian Mechanism

In this section, we describe the discrete Gaussian mechanism [CKS20] and the modified sampling algorithm Algorithm 2.7. The discrete Gaussian mechanism is defined as:

$$M_{DGauss}(D) = f(D) + Y, \tag{2.35}$$

where $f(D) \in \mathbb{Z}$ and $Y \sim DGau(\mu = 0, \sigma)$ (cf. 1.3.9).

Theorem 8 ([CKS20]). *The discrete Gaussian mechanism $M_{DGauss}(D) = f(D) + Y$ satisfies (ε, δ)-DP for query function $f(D) \in \mathbb{Z}$.*

Canonne et al. [CKS20] proposed a discrete Gaussian sampling algorithm that is based on rejection sampling [CRW04] technique. The algorithm first generates a random discrete Laplace random variable and then converts it to a discrete Gaussian random variable.

Discrete Gaussian Sampling Algorithm. Algorithm 2.7 is a modificaiton of the discrete Gaussian sampling algorithm from the work [CKS20] that samples $Y \sim DGau(\mu = 0, \sigma)$ (cf. 1.3.9). We replace the $WHILE$ loop in the original work with FOR loop (line 2). We use Algorithm 2.6 to generate the discrete Laplace random variable in line 3 and Algorithm 1.1 to generate the Bernoulli random variable B in line 4.

Algorithm: $Algo^{DGau}(\sigma)$

Input: σ
Output: $Y \sim DGau(\mu = 0, \sigma)$

1 : $t \leftarrow \lfloor \sigma \rfloor + 1$
2 : **FOR** $j \leftarrow 1$ **TO** $ITER$
3 : $\quad L \leftarrow DLap(t)$
4 : $\quad B \leftarrow Bern\left(e^{\left(|L| - \sigma^2/t\right)^2 / 2\sigma^2}\right)$
5 : $\quad$ **IF** $B == 1$
6 : $\quad\quad$ **RETURN** $Y \leftarrow L$ // success
7 : **RETURN** $Y \leftarrow 0$ // failure

Algorithm 2.7: Sampling from a discrete Gaussian distribution $DGau(\mu = 0, \sigma)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that Algorithm 2.7 fails in $ITER_1 = i$ iterations. We first compute $\Pr(A_1)$ as follows:

$$\begin{aligned}
\Pr(A_1) &= \sum_{i=0}^{\infty} \Pr(B = 0 \wedge L = i \wedge Algorithm\ 2.6\ \text{successes}) + \Pr(Algorithm\ 2.6\ \text{fails}) \\
&= \sum_{i=0}^{\infty} (\Pr(B = 0) \cdot \Pr(L = i) \cdot \Pr(Algorithm\ 2.6\ \text{successes})) \\
&\quad + \Pr(Algorithm\ 2.6\ \text{fails}) \\
&= \sum_{i=0}^{\infty} \left(1 - e^{-\frac{\left(|i| - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}} \right) \cdot \frac{\left(e^{\frac{1}{t}} - 1\right) \cdot e^{-\frac{|i|}{d}}}{e^{\frac{1}{t}} + 1} \cdot \Pr(Algorithm\ 2.6\ \text{successes}) \\
&\quad + \Pr(Algorithm\ 2.6\ \text{fails})
\end{aligned} \tag{2.36}$$

To guarantee that $\Pr(Algorithm\ 2.7\ \text{fails}) < 2^{-40}$ and $\Pr(Algorithm\ 2.6\ \text{fails}) < 2^{-40}$, it requires that $ITER = 23$ for $\sigma = 1.5$.s

3 SMPC Protocols for Differentially Private Mechanisms

In this chapter, we first present a SMPC-DP procedure that combines SMPC protocols and differentially private mechanisms. Then, we construct the SMPC protocols of the previously introduced secure differentially private mechanisms and noise sampling algorithms (cf. § 2).

Generally, we face two technical challenges when transforming the differentially private mechanisms into SMPC protocols. The first is to identify the most efficient SMPC protocols for arithmetic operations. The second is determining the most efficient sampling algorithms for differentially private mechanisms. For the first challenge, we implement SMPC protocols that support (signed) integer arithmetic, fixed-point arithmetic, floating-point arithmetic, and data type conversions in MOTION [BDST22]. For the second challenge, we construct the SMPC protocols for all the introduced sampling algorithms and compare the performance in **??**.

Setting. In our SMPC-DP procedure, we consider m users, n computation parties and one reconstruction party. Each user U_i has private data D_i and wish to compute $f(D_1, \ldots, D_m)$ and perturb it with the help of computation parties. The reconstruction party is responsible for reconstructing the computation result in plaintext. We assume that the communication channels between users and computation parties are secure and authenticated, and the communication channels between computation parties are pair-wise secure and authenticated. The users can go offline after sending their secret shared private data to the computation parties. Then, the computation parties execute the interactive SMPC protocols.

Privacy Goals. For users, computation parties, and the reconstruction party, the reconstructed perturbed result should satisfy differential privacy, and the whole computation process should leak no information about the individual users' private data.

Attacker Model. The users, computation parties, and the reconstruction party may be corrupted and collude with each other. For the computation parties, we assume that the adversaries are semi-honest (i.e., they follow the protocol specifications but try to infer information), and there is at least one computation party that is not corrupted (i.e., full-threshold security).

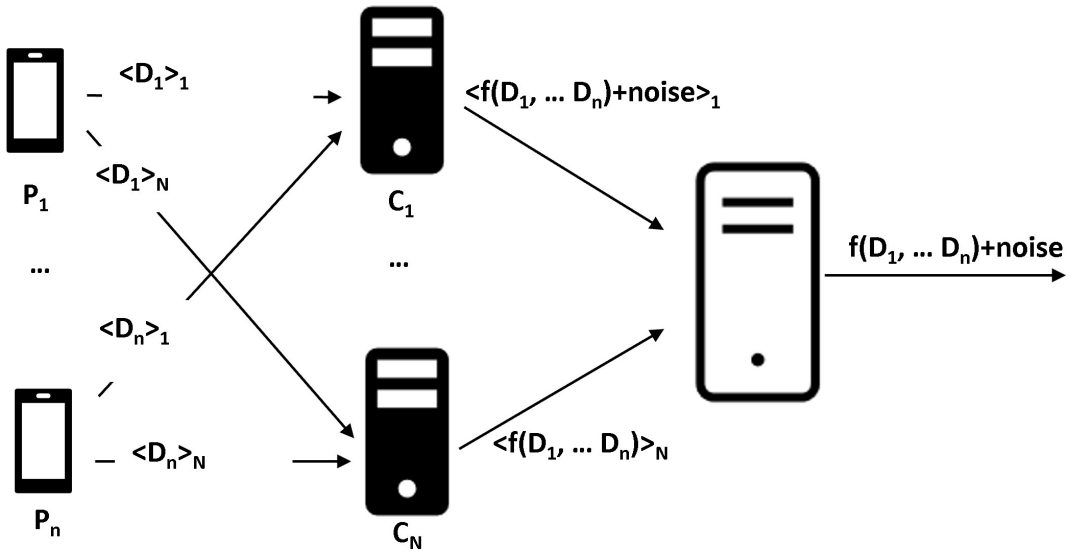

Figure 3.1: SMPC-DP Procedure

SMPC-DP Procedure Overview. SMPC-DP procedure consists of three steps as Fig. 3.1 shows: (i) The users send the secret shared data $\langle D_i \rangle$ to the computation parties. (ii) The computation parties execute the SMPC protocols to compute $\langle f(D_1, \ldots, D_n) \rangle$ (we assume this step is finished), generate secret shared noise $\langle Y \rangle$, perturb the result with differentially privacy mechanisms. (iii) The computation parties send their share of the perturbed result to the reconstruction party, that reconstructs the perturbed result.

3.1 Building Blocks

We construct SMPC protocols based on the following building blocks (details can be found in § A.1):

1. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{RandBits}(\ell)$ allows parties to generate shares of a publicly unknown random ℓ-bit string $\langle \boldsymbol{y} \rangle^B = \left(\langle y_0 \rangle^B, \ldots, \langle y_{\ell-1} \rangle^B \right)$ without interactive operation.
2. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{PreOr}\left(\langle \boldsymbol{x} \rangle^B \right)$ computes the prefix-OR of a ℓ-bit string $\langle \boldsymbol{x} \rangle^B = \left(\langle x_0 \rangle^B, \ldots, \langle x_{\ell-1} \rangle^B \right)$ and output $\langle \boldsymbol{y} \rangle^B = \left(\langle y_0 \rangle^B, \ldots, \langle y_{\ell-1} \rangle^B \right)$ such that $\left\langle y_j \right\rangle^B = \vee_{k=0}^{j} \langle x_k \rangle^B$ for $j \in [0, \ell - 1]$, where $\langle y_0 \rangle^B = \langle x_0 \rangle^B$.
3. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{Geometric}$ generates shares of a geometric random variable $y \sim Geo(p = 0.5)$.
4. $\left(\langle y_1 \rangle^B, \ldots, \langle y_\ell \rangle^B \right) \leftarrow \Pi^{Binary2Unary}\left(\langle \boldsymbol{x} \rangle^B, \ell \right)$ [ABZS12] converts an unsigned integer x from binary to unary bitwise representation and outputs a ℓ-bit string $\boldsymbol{y} = (y_0, \ldots, y_{\ell-1})$,

where the x least significant bits $y_0, \ldots, y_{x-1}$ are set to 1 and the rest bits $y_x, \ldots, y_{\ell-1}$ are set to 0.

5. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{HW}(x_0, \ldots, x_\ell)$ [BP08] computes the Hamming weight (i.e., the number of 1 bits) in a ℓ-bit string $x_0, \ldots, x_\ell$.
6. $\langle \boldsymbol{y_i} \rangle^B \leftarrow \Pi^{ObliviousSelection}\left(\langle \boldsymbol{y_0} \rangle^B, \ldots, \langle \boldsymbol{y_{\ell-1}} \rangle^B, \langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$ outputs bit-string $\boldsymbol{y_i}$ where i is the first index such that $c_i == 1$ for $i \in [0, \ell-1]$.
7. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{RandInt}(m)$ generate shares of a random integer y in the interval $[0, m-1]$.
8. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{COTMult}\left(\langle \boldsymbol{x} \rangle^B, \langle b \rangle^B\right)$ [AHLR18; ST19] computes the multiplication of a bit string $\langle \boldsymbol{x} \rangle^B$ and a single bit $\langle b \rangle^B$ with correlated-OT (cf. § 1.2.2).
9. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{Mux}\left(\langle a \rangle^B, \langle \boldsymbol{x_0} \rangle^B, \langle \boldsymbol{x_1} \rangle^B\right)$ outputs bit-string $\langle \boldsymbol{x_0} \rangle^B$ if $a = 1$, $\langle \boldsymbol{x_0} \rangle^B$ otherwise.

3.2 Number Representations and Arithmetic Operations

We consider SMPC protocols for arithmetic operations based on Boolean sharing GMW (BGMW), Yao sharing with BMR (Y), and arithmetic sharing GMW (AGMW). Further, our SMPC protocols use integer, fixed-point and floating-point arithmetic interchangeably to achieve the best SMPC performance (regarding the communication and computation cost).

Binary Circuit-Based Signed/Unsigned Integer. The MOTION framework [BDST22] supports most unsigned integer arithmetic operations (e.g., $+$, $-$, $*$, $/$, and $<$). We extend it to support further arithmetic operations (e.g., modulo reduction and conversion operations with fixed-point and floating-point) by generating depth-optimized binary circuits for BGMW-based unsigned integer and size-optimized binary circuits for BMR-based unsigned integer with HyCC and CBMC-GC. Except for the arithmetic operations supported by the unsigned integer, we also implement signed integer arithmetic that supports operations such as absolute value and negation.

Binary Circuit-Based Floating-Point. We consider 64-bit floating-point arithmetic. For operations such as $+$, $-$, $*$, $/$, $<$, square, square root, exp2, and log2, we use the binary circuits from ABY [DSZ15]. For other operations such as conversion operations with fixed-point and integers, ceil, floor, absolute value, round to the nearest integer, we use CBMC-GC to generate depth-optimized and size-optimized binary circuits with the C code from Berkeley SoftFloat library [Hau18].

Binary Circuit-Based Fixed-Point. The main issue with fixed-point arithmetic is the inherent precision loss during arithmetic operations. Therefore, we implement fixed-point arithmetic in two precision modes: (i) 64-bit fixed-point with a 16-bit fractional part, (ii) 64-bit fixed-point with a 33-bit fractional part. We use HyCC to generate the depth-optimized and size-optimized circuits for fixed-point operations such as +, -, *, /, and <. For operations

such as exp2 and log2, we use polynomial approximations [Har78; AS19] and generate corresponding binary circuits using CBMC-GC. For operations such as sqrt, we use both polynomial approximations [Har78; AS19] and Goldschmidt approximations [Mar04; AS19].

AGMW-Based Floating-point. We implement the arithmetic sharing-GMW based 64-bit floating-point protocols based on the work of Aliasgari et al. [ABZS12]. A floating-point number $u = (1-2s)\cdot(1-z)\cdot v\cdot 2^p$ is represented as a quadruple (v,p,z,s), where each term is defined as follows:

$v \in \left[2^{\ell-1}, 2^{\ell}\right)$ is a ℓ-bit mantissa (with most significnant bit always set to one), $p \in \mathbb{Z}_k$ is a k-bit signed exponent, z is the zero bit that is set to 1 when $u = 0$, s is the sign bit.

The arithmetic operations in work [ABZS12] are performed over a prime field $\mathbb{F}_p$ (modulo p) , whereas the arithmetic operations in MOTION are performed in the ring field (modulo $\mathbb{Z}_{2^n}$). One significant difference between these two fields is that there is no inverse element (i.e., a^{-1} in a prime field) in a ring field $\mathbb{Z}_{2^n}$. Therefore, we use the protocols (e.g., logical shifting, truncation, truncation by a secret value, eta.) from works [EGK+20; DEK20] to replace the inverse element operation in work [ABZS12].

AGMW-based Fixed-point. We implement the arithmetic sharing GMW based fixed-point protocols from Catrina and Saxena's work [CS10], where a fixed-point number x is represented as $x = \overline{x}\cdot 2^{-f}$, with $\overline{x} \in \mathbb{Z}_{\langle k\rangle}$, $\mathbb{Z}_{\langle k\rangle} = \left\{x \in \mathbb{Z} \mid -2^{k-1}+1 \le x \le 2^{k-1}-1\right\}$. f is the length of fraction bits. We choose $(k,f) = (41,20)$ as [ACC+21] recommended.

Random Number Generation As discussed in (cf. § 2), the differentially private mechanisms rely heavily on generating of random numbers. Generally, four types of random numbers are used in our protocols: (i) random BGMW bits, random unsigned integers in a specific range, random fixed/floating-point numbers in the range $(0,1)$. To generate a random Boolean bit-string of length ℓ, each computation party P_i locally generate a random ℓ-bit string $\boldsymbol{r_i^B}$ and set $\boldsymbol{r_i^B}$ as a Boolean sharing with GMW protocol of a publicly unknown random bit string $\boldsymbol{r_0^B} \oplus \ldots \oplus \boldsymbol{r_{N-1}^B}$. Let $\Pi^{RandBits}(\ell)$ denote the SMPC protocol for generating random bits of length ℓ.

The last three types of random numbers are generated based on random Boolean bits with additional conversion.

To generate a random unsigned integer in range $(0,m)$, we use the Simple Modular Method [BK15] that first generate a random bit string $\langle \boldsymbol{r}\rangle^B$ of length ℓ ($\ell = 128$ or $\ell = 256$), and compute $\langle \boldsymbol{r}\rangle^B \mod m$ by taking $\langle \boldsymbol{r}\rangle^B$ as a ℓ-bit unsigned integer. We keep the security parameter $s = \ell - \log_2 m$ greater than 64 by limiting $m < 2^{64}$. To generate a uniform random fixed-point in the range $[0,1)$ with f fractional bits and $k-f$ integer bits, we first generate f random bits as the fractional part of the random fixed-point and set the $(k-f)$-bit integer part with zero bits.

To generate a uniform random floating point in range $(0,1)$, we follow the methods discussed in Algorithm 2.1 and construct corresponding SMPC protocol $\Pi^{RandFloat1}$.

As $\Pi^{RandFloat1}$ shows, we first generate shares of the mantissa bits $\langle d_1\rangle^B, \ldots, \langle d_{52}\rangle^B$ with $\Pi^{RandBits}(52)$, and generate a share of geometric random variable $\langle x\rangle^{B,UINT}$ with $\Pi^{Geometric}(0.5)$ to build the biased exponent $\langle e\rangle^{B,UINT}$, where $e = 1023 - (x+1)$. Finally, the parties use the mantissa and exponent bits to build the floating-point number $U^* = (1.d_1 \ldots d_{52})_2 \times 2^{e-1023}$.

Protocol: $\Pi^{RandFloat1}$

Input: None

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

1 : $\left(\langle d_1\rangle^B, \ldots, \langle d_{52}\rangle^B\right) \leftarrow \Pi^{RandBits}(52)$

2 : $\langle x\rangle^{B,UINT} \leftarrow \Pi^{Geometric}(0.5)$

3 : $\langle e\rangle^{B,UINT} \leftarrow 1023 - \left(\langle x\rangle^{B,UINT} + 1\right)$

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

Protocol 3.1: SMPC protocol for sampling uniform random floating-point $U^* \in \mathbb{D} \cap (0,1)$.

3.3 SMPC Protocols for Snapping Mechanism

Recall that the snapping mechanism (cf. § 2.1) is defined as follows:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)\right\rceil_\Lambda\right). \tag{3.37}$$

The SMPC protocol for the snapping mechanism is to unfold the mathematical operations and replaces them with the corresponding SMPC protocols. We reformulate $M_S(f(D), \lambda, B)$ in secret sharing form as follows:

$$\langle M_S(f(D), \lambda, B)\rangle = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(\langle f(D)\rangle) \oplus \langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)\right\rceil_\Lambda\right). \tag{3.38}$$

We assume that the computation parties has already computed $\langle f(D)\rangle$. $\langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)$ is the share of noise. We use $\Pi^{RandFloat1}$ and $\Pi^{RandBits}$ to generate random floating-point number $U^* \in (0,1)$ and random sign $S \in \{0,1\}$. Floating-point operations such as natural logarithm operation $LN(\cdot)$, addition $\oplus$ and multiplication $\otimes$ are available in SMPC protocols. Operation $\lfloor\langle x\rangle\rceil_\Lambda$ rounds input x to the nearest multiple of Λ (a publicly known value). The plaintext implementation of $\lfloor x\rceil_\Lambda$ can be found in Covington's work [Cov19], that relies on the bit manipulation of floating-point. Therefore, we choose BGMW floating-point arithmetic and generate depth-optimized circuits with CBMC-GC for operation $\lfloor\langle x\rangle\rceil_\Lambda$. The SMPC protocol of $\lfloor\langle x\rangle\rceil_\Lambda$ is denoted by $\Pi^{RoundToLambda}$. The SMPC protocol of operation clamp_B can be directly realized with the floating-point arithmetic operation, and it is denoted by Π^{ClampB}.

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

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

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

1 : $\left\langle cond_{|x|<B}\right\rangle^{B} \leftarrow \Pi^{FL\_Lt}\left(\Pi^{FL\_Abs}\left(\langle x\rangle^{B,FL}\right), B\right)$

2 : $\left\langle x_{clampB}\right\rangle^{B,FL} \leftarrow \Pi^{Mux}\left(\left\langle cond_{|x|<B}\right\rangle^{B}, \langle x_0\rangle^{B,FL}, B\right)$

3 : Extract the sign bit of $\langle x\rangle^{B,FL}$ and set it as the sign bit of $\left\langle x_{clampB}\right\rangle^{B,FL}$

Protocol 3.2: SMPC protocol for $\text{clamp}_B(x)$.

SMPC Protocols for Snapping Mechanism. We integrate the protocols and present the complete SMPC protocol for the snapping mechanism.

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

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

Output: $\langle x_{SM}\rangle^{B,FL}$, **where** $x_{SM} = M_S(f(D), \lambda, B)$

1 : $\langle U^*\rangle^{B,FL} \leftarrow \Pi^{RandFloat1}$.

2 : $\langle S\rangle^{B} \leftarrow \Pi^{RandBits}(1)$.

3 : $\left\langle f(D)_{clampB}\right\rangle^{B,FL} \leftarrow \Pi^{ClampB}\left(\langle f(D)\rangle^{B,FL}, B\right)$.

4 : $\left\langle Y_{LapNoise}\right\rangle^{B,FL} = \Pi^{FL\_Mul}\left(\lambda, \Pi^{FL\_Ln}\left(\langle U^*\rangle^{B,FL}\right)\right)$.

5 : Set $\langle S\rangle^{B}$ as the sign bit of $\left\langle Y_{LapNoise}\right\rangle^{B,FL}$.

6 : $\langle x\rangle^{B,FL} \leftarrow \Pi^{FL\_Add}\left(\left\langle f(D)_{clampB}\right\rangle^{B,FL}, \left\langle Y_{LapNoise}\right\rangle^{B,FL}\right)$.

7 : $\langle \lfloor x\rceil_\Lambda\rangle^{B,FL} \leftarrow \Pi^{RoundToLambda}\left(\langle x\rangle^{B,FL}, \Lambda\right)$.

8 : $\langle x_{SM}\rangle^{B,FL} \leftarrow \Pi^{ClampB}\left(\langle \lfloor x\rceil_\Lambda\rangle^{B,FL}, B\right)$

Protocol 3.3: SMPC protocol for snapping mechanism.

3.4 SMPC Protocols for Integer-Scaling Laplace Mechanism

Recall that the Integer-Scaling Laplace mechanism $M_{ISLap}(f(D), r, \varepsilon, \Delta_r) = f_r(D) + ir$ (cf. § 2.2) use a discrete Laplace random variable $i \sim DLap\left(t = \frac{\Delta_r}{r\varepsilon}\right)$ to simulate a continuous Laplace random variable.

As discussed in § 2.2.1, one way to generate discrete Laplace random variable i is to use $Algo^{GeoExpBinarySearch}$ and $Algo^{TwoSideGeo}$. We provide the SMPC protocols for both algorithms and the SMPC protocols for the Integer-Scaling Laplace mechanism.

SMPC Protocols for Binary Search Based Geometric Sampling Algorithm $\Pi^{GeoExpBinarySearch}(\lambda)$ samples a geometric random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ based on $Algo^{GeoExpBinarySearch}$ (cf. Algorithm 2.2).

In line $1-7$, each party locally calculates L_0, R_0, M_0 and Q_0 in plaintext to save SMPC computations. In line $8-12$, the parties generate a floating-point random variable U_0 and use the comparison result of U_0 and Q_0 to choose one subinterval (either $(L_0 \dots M_0]$ or $(M_0 \dots R_0]$) and compute a flag fg_0 that indicates whether the termination condition of **WHILE** (cf. Algorithm 2.2) is satisfied. In line $16-26$, the parties execute the binary search for $ITER-1$ times. In line 19, the parties choose the correct split-point M_j using $\Pi^{COTMult}$ as follows:

$$\begin{aligned}\left\langle M_j \right\rangle^{B,UINT} = \quad & \Pi^{COTMult}\left(\left\langle cond_{M_j \leq L_{j-1}} \right\rangle^{B}, \Pi^{UINT\_Add}\left(\left\langle L_{j-1} \right\rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\left\langle cond_{M_j \geq R_{j-1}} \right\rangle^{B}, \Pi^{UINT\_Sub}\left(\left\langle R_{j-1} \right\rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\left\langle cond_{L_{j-1} < M_j < R_{j-1}} \right\rangle^{B}, \left\langle M_j \right\rangle^{B,UINT}\right)\end{aligned} \tag{3.39}$$

In line 27, the parties extract the correct result from $\left(\langle R_0 \rangle^{B,UINT}, \dots, \langle R_{iter-1} \rangle^{B,UINT}\right)$ based on flags $\langle fg_0 \rangle^{B}, \dots, \langle fg_{iter-1} \rangle^{B}$. Note that in line 15, 20, we use operator such as $+$, $-$, $*$, $/$ and e^x instead of SMPC protocols of arithmetic operations for simplicity.

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

Input: λ

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

$L_0 \leftarrow 0, R_0 \leftarrow 2^{52}$

$M_0 \leftarrow L_0 - \dfrac{\ln(0.5) + \ln\left(1+e^{-\lambda(R_0-L_0)}\right)}{\lambda}$

IF $M_0 \leq L_0$

  $M_0 \leftarrow L_0 + 1$

ELSE IF $M_0 \geq R_0$

  $M_0 \leftarrow R_0 - 1$

$Q_0 \leftarrow \dfrac{e^{-\lambda(M_0-L_0)}-1}{e^{-\lambda(R_0-L_0)}-1}$

$\langle U_0\rangle^{B,FL} \leftarrow \Pi^{RandFloat1}$

$\left\langle cond_{U_0\leq Q_0}\right\rangle^{B} \leftarrow \Pi^{FL\_LEQ}\left(\langle U_0\rangle^{B,FL}, Q_0\right)$

$\left\langle cond_{U_0>Q_0}\right\rangle^{B} \leftarrow \neg\left\langle cond_{U_0\leq Q_0}\right\rangle^{B}$

$\langle R_0\rangle^{B,UINT} \leftarrow \Pi^{COTMult}\left(\left\langle cond_{U_0\leq Q_0}\right\rangle^{B}, M_0\right) \oplus \Pi^{COTMult}\left(\left\langle cond_{U_0>Q_0}\right\rangle^{B}, R_0\right)$

$\langle L_0\rangle^{B,UINT} \leftarrow \Pi^{COTMult}\left(\left\langle cond_{U_0>Q_0}\right\rangle^{B}, M_0\right) \oplus \Pi^{COTMult}\left(\left\langle cond_{U_0\leq Q_0}\right\rangle^{B}, L_0\right)$

$\langle fg_0\rangle^{B} \leftarrow \Pi^{UINT\_GEQ}\left(\Pi^{UINT\_Add}\left(\langle L_0\rangle^{B,UINT}, 1\right), \langle R_0\rangle^{B,UINT}\right)$

FOR $j \leftarrow 1$ **TO** $ITER-1$

  $\left\langle M_j\right\rangle^{B,UINT} \leftarrow \left\langle L_{j-1}\right\rangle^{B,UINT} - \Pi^{FL2UI}\left(\dfrac{\ln(0.5) + \ln\left(1+e^{-\lambda\cdot\Pi^{UI2FL}\left(\left\langle R_{j-1}\right\rangle^{B,UINT} - \left\langle L_{j-1}\right\rangle^{B,UINT}\right)}\right)}{\lambda}\right)$

  $\left\langle cond_{M_j\leq L_{j-1}}\right\rangle^{B} \leftarrow \Pi^{UINT\_LEQ}\left(\left\langle M_j\right\rangle^{B,UINT}, \left\langle L_{j-1}\right\rangle^{B,UINT}\right)$

  $\left\langle cond_{M_j\geq R_{j-1}}\right\rangle^{B} \leftarrow \Pi^{UINT\_GEQ}\left(\left\langle M_j\right\rangle^{B,UINT}, \left\langle R_{j-1}\right\rangle^{B,UINT}\right)$

  $\left\langle cond_{L_{j-1}<M_j<R_{j-1}}\right\rangle^{B} \leftarrow \neg\left(\left\langle cond_{M_j\leq L_{j-1}}\right\rangle^{B} \vee \left\langle cond_{M_j\geq R_{j-1}}\right\rangle^{B}\right)$

  $\left\langle M_j\right\rangle^{B,UINT} \leftarrow$ Eq. (3.39)

  $\left\langle Q_j\right\rangle^{B,FL} \leftarrow \dfrac{e^{-\lambda\cdot\Pi^{UI2FL}\left(\left\langle M_j\right\rangle^{B,UINT} - \left\langle L_{j-1}\right\rangle^{B,UINT}\right)}-1}{e^{-\lambda\cdot\Pi^{UI2FL}\left(\left\langle R_{j-1}\right\rangle^{B,UINT} - \left\langle L_{j-1}\right\rangle^{B,UINT}\right)}-1}$

  $\left\langle U_j\right\rangle^{B,FL} \leftarrow \Pi^{RandFloat1}$

  $\left\langle cond_{U_j\leq Q_j}\right\rangle^{B} \leftarrow \Pi^{FL\_LEQ}\left(\left\langle U_j\right\rangle^{B,FL}, \left\langle Q_j\right\rangle^{B,FL}\right)$

  $\left\langle cond_{U_j>Q_j}\right\rangle^{B} \leftarrow \neg\left\langle cond_{U_j\leq Q_j}\right\rangle^{B}$

  $\left\langle R_j\right\rangle^{B,UINT} \leftarrow \Pi^{COTMult}\left(\left\langle cond_{U_j\leq Q_j}\right\rangle^{B}, \left\langle M_j\right\rangle^{B,UINT}\right) \oplus \Pi^{COTMult}\left(\left\langle cond_{U_j>Q_j}\right\rangle^{B}, \left\langle R_{j-1}\right\rangle^{B,UINT}\right)$

  $\left\langle L_j\right\rangle^{B,UINT} \leftarrow \Pi^{COTMult}\left(\left\langle cond_{U_j>Q_j}\right\rangle^{B}, \left\langle M_j\right\rangle^{B,UINT}\right) \oplus \Pi^{COTMult}\left(\left\langle cond_{U_j\leq Q_j}\right\rangle^{B}, \left\langle L_{j-1}\right\rangle^{B,UINT}\right)$

  $\left\langle fg_j\right\rangle^{B} \leftarrow \Pi^{UINT\_GEQ}\left(\Pi^{UINT\_Add}\left(\left\langle L_j\right\rangle^{B,UINT}, 1\right), \left\langle R_j\right\rangle^{B,UINT}\right)$

$\langle x\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(\langle R_0\rangle^{B,UINT}, \ldots, \langle R_{ITER-1}\rangle^{B,UINT}, \langle fg_0\rangle^{B}, \ldots, \langle fg_{ITER-1}\rangle^{B}\right)$

Protocol 3.4: SMPC protocol for sampling geometric random variable $x \sim Geo\left(p = 1-e^{-\lambda}\right)$.

SMPC Protocols for Two-Side Geometric Sampling Algorithm Next, we construct $\Pi^{TwoSideGeometric}$ that converts a geometric random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ to a discrete Laplace random variable $i \sim DLap\left(\frac{1}{\lambda}\right)$ based on $Algo^{TwoSideGeo}$ (cf. Algorithm 2.3).

In line 2, 3, the parties generate the sign s_j and integer part g_j of discrete Laplace random variable $i = (-1)^{s_j} \cdot g_j$. In line 4, the parties compute the flag $f\,g_j$ to record if the termination condition of **WHILE** loop (cf. Algorithm 2.3) is satisfied. In line 5, we extract the correct sign s and the number part g based on flags $\langle f\,g_0\rangle^B, \ldots, \langle f\,g_{iter-1}\rangle^B$.

Protocol: $\Pi^{TwoSideGeometric}(t)$

Input: t

Output: $\langle i\rangle^{B,UINT}$, **where** $i \sim DLap(t)$

1 : **FOR** $j \leftarrow 0$ **TO** $ITER - 1$

2 : $\quad \left\langle s_j \right\rangle^B \leftarrow \Pi^{RandBits}(1)$.

3 : $\quad \left\langle g_j \right\rangle^{B,UINT} \leftarrow \Pi^{GeoExpBinarySearch}\left(\lambda = \frac{1}{t}\right)$.

4 : $\quad \left\langle f\,g_j \right\rangle^B \leftarrow \neg\left(\left(\left\langle s_j \right\rangle^B == 1\right) \wedge \left(\left\langle g_j \right\rangle^{B,UINT} == 0\right)\right)$

5 : $\langle g\rangle^{B,UINT} \,\|\, \langle s\rangle^B \leftarrow \Pi^{ObliviousSelection}\left(\langle g_0\rangle^{B,UINT} \,\|\, \langle s_0\rangle^B, \ldots, \langle g_{ITER-1}\rangle^{B,UINT} \,\|\, \langle s_{ITER-1}\rangle^B, \langle f\,g_0\rangle^B, \ldots, \langle f\,g_{ITER-1}\rangle^B\right)$

6 : Set $\langle s\rangle^B$ as the sign bit of $\langle g\rangle^{B,UINT}$

7 : $\langle i\rangle^{B,UINT} \leftarrow \langle g\rangle^{B,UINT}$

Protocol 3.5: SMPC Protocol for sampling two-side geometric random variable $x \sim DLap(t)$.

Implementaion with SIMD. As discussed in **??**, $Algo^{TwoSideGeo}$ iterates for $ITER$ times to increase the success probability ($> 1 - 2^{-40}$) of outputting the correct result. Since each iteration (line $2 - 4$) is independent, we uses SIMD to eliminate the FOR loop and increase the efficiency of $\Pi^{TwoSideGeometric}$. Specifically, we first generate $\langle s_{simd}\rangle^B = \left(\langle s_0\rangle^B, \ldots, \left(\langle s_{ITER-1}\rangle^B\right)\right)$ and $\langle g_{simd}\rangle^{B,UINT} = \left(\langle g_0\rangle^{B,UINT}, \ldots, \langle g_{ITER-1}\rangle^{B,UINT}\right)$ by running $\Pi^{RandBits}$ and $\Pi^{GeoExpBinarySearch}$ for single time but with $ITER$ bits in each wire. Note that $\langle s_{simd}\rangle^B$ and $\langle g_{simd}\rangle^{B,UINT}$ has the same number of Boolean GMW wires as $\left\langle s_j \right\rangle^B$ and $\left\langle g_j \right\rangle^{B,UINT}$ but $ITER$ number of bits in each wire. Then, we run the rest operations in the FOR loop with $\langle s_{SIMD}\rangle^B$ and $\langle g_{SIMD}\rangle^{B,UINT}$. We also applid this FOR loop elimination method in other SMPC protocols that has independent operations in their FOR loops.

SMPC Protocol for Integer-Scaling Laplace Mechanism Next, we integrate the protocols to construct the complete SMPC protocol for the Integer-Scaling Laplace mechanism.

Integer-scaling Laplace mechanism can be reformulated in secret sharing as:

$$\left\langle M_{ISLap}\left(f_r(D), r, \Delta_r, \varepsilon\right)\right\rangle = \langle f_r(D)\rangle + \langle i\rangle \cdot r \tag{3.40}$$

where r, ε, Δ_r are publicly known values. We assume that the parties have already computed $\langle f_r(D)\rangle$.

Protocol: $\Pi^{ISLap}\left(\langle f_r(D)\rangle^{B,FL}, r, \Delta_r, \varepsilon\right)$

Input: $\langle f_r(D)\rangle^{B,FL}$, r ,Δ_r,ε,$t = \frac{\Delta_r}{r\varepsilon}$

Output: $\left\langle M_{ISLap}\right\rangle^{B,FL}$

1 : $\langle i\rangle^{B,UINT} \leftarrow \Pi^{TwoSideGeometric}(t)$.

2 : $\left\langle Y_{LapNoise}\right\rangle^{B,FL} \leftarrow \Pi^{FL\_MUL}\left(\Pi^{UINT2FL}\left(\langle i\rangle^{B,UINT}\right), r\right)$.

3 : $\left\langle M_{ISLap}\right\rangle^{B,FL} \leftarrow \Pi^{FL\_Add}\left(\langle f_r(D)\rangle^{B,FL}, \left\langle Y_{LapNoise}\right\rangle^{B,FL}\right)$.

Protocol 3.6: SMPC Protocol for Integer-Scaling Laplacian mechanism.

3.5 SMPC Protocol for Integer-Scaling Gaussian Mechanism

Recall that the Integer-Scaling Gaussian Mechanism $M_{ISGauss}(f(D), r, \Delta_r, \varepsilon, \sigma) = f_r(D) + ir$ (cf. § 2.2.2) use a symmetric binomial random variable $i \sim SymmBino(n, p = 0.5)$ to simulate a Gaussian random variable.

We first provide the SMPC protocols for $Algo^{SymmetricBinomial}$, and then, the SMPC protocols for the Integer-Scaling Gaussian mechanism.

SMPC Protocol for Symmetrical Binomial Sampling Algorithm $\Pi^{SymmBino}(n, p = 0.5)$ samples a symmetric binomial random variable $x \sim SymmBino(n, p = 0.5)$ based on $Algo^{SymmetricBinomial}$ (cf. Algorithm 2.4).

Given value of $\sqrt{n}$, each party first compute following publicly known parameters:

$$\begin{aligned} m &= \left\lfloor \sqrt{2}\cdot\sqrt{n}+1 \right\rfloor, \\ x_{min} &= -\frac{\sqrt{n}\cdot\sqrt{\ln\sqrt{n}}}{\sqrt{2}}, \\ x_{max} &= -x_{min}, \\ v_n &= \frac{0.4\cdot 2^{1.5}\cdot \ln^{1.5}(\sqrt{n})}{\sqrt{n}}, \\ \tilde{p}_{coe} &= \sqrt{\frac{2}{\pi}}\cdot(1-v_n)\cdot\frac{1}{\sqrt{n}}. \end{aligned} \tag{3.41}$$

Note that in line 15, the parties compute the condition $\left\langle cond_{x_{min}\leq x_j\leq x_{max}}\right\rangle^B \wedge \left\langle cond_{c_j==1}\right\rangle^B$ to record if the **WHILE** loop condition (cf. Algorithm 2.4) is satisfied. In line 16, the parties extract the correct $\langle i\rangle^{B,INT}$ based on flags $\langle fg_0\rangle^B,\ldots,\langle fg_{ITER-1}\rangle^B$.

Protocol: $\Pi^{SymmBinomial}\left(\sqrt{n}\right)$

Input: $\sqrt{n}$

Output: $\langle i\rangle^{B,UINT}$, **where** $i\sim SymmBino\left(n,p=0.5\right)$

1 : **FOR** $j\leftarrow 0$ **TO** $ITER-1$

2 : $\left\langle s_j\right\rangle^{B,INT}\leftarrow\Pi^{Geometric}\left(0.5\right)$

3 : $\left\langle s_j\right\rangle^{B,FL}\leftarrow\Pi^{SI2FL}\left(\left\langle s_j\right\rangle^{B,INT}\right)$

4 : $\left\langle s_j'\right\rangle^{B,INT}\leftarrow\Pi^{INT\_NEG}\left(\Pi^{INT\_ADD}\left(\left\langle s_j\right\rangle^{B,INT},1\right)\right)$

5 : $\left\langle b_j\right\rangle^{B}\leftarrow\Pi^{RandBits}\left(1\right)$

6 : $\left\langle k_j\right\rangle^{B,INT}\leftarrow\Pi^{Mux}\left(\left\langle b_j\right\rangle^{B},\left\langle s_j\right\rangle^{B,INT},\left\langle s_j'\right\rangle^{B,INT}\right)$

7 : $\left\langle l_j\right\rangle^{B,INT}\leftarrow\Pi^{RandInt}\left(m\right)$

8 : $\left\langle x_j\right\rangle^{B,INT}\leftarrow\Pi^{INT\_Add}\left(\left(\Pi^{INT\_Mul}\left\langle k_j\right\rangle^{B,INT},m\right),\left\langle l_j\right\rangle^{B,INT}\right)$

9 : $\left\langle cond_{x_{min}\leq x_j\leq x_{max}}\right\rangle^B\leftarrow\Pi^{INT\_GEQ}\left(\left\langle x_j\right\rangle^{B,INT},x_{min}\right)\wedge\Pi^{INT\_LEQ}\left(\left\langle x_j\right\rangle^{B,INT},x_{max}\right)$

10 : $\left\langle\tilde{p}_j\right\rangle^{B,FL}\leftarrow\Pi^{FL\_MUL}\left(\tilde{p}_{coe},\Pi^{FL\_Exp}\left(\Pi^{FL\_Sqr}\left(\Pi^{FL\_MUL}\left(\frac{-2}{\sqrt{n}},\left\langle x_j\right\rangle^{B,FL}\right)\right)\right)\right).$

11 : $\left\langle cond_{\tilde{p}_j>0}\right\rangle^B\leftarrow\left\langle cond_{x_{min}\leq x_j\leq x_{max}}\right\rangle^B$

12 : $\langle p_{Bernoulli}\rangle^{B,FL}\leftarrow\Pi^{FL\_MUL}\left(\left\langle\tilde{p}_j\right\rangle^{B,FL},\Pi^{FL\_MUL}\left(\Pi^{UINT2FL}\left(\Pi^{UINT\_Exp2}\left(\left\langle s_j\right\rangle^{B,UINT}\right)\right),\frac{m}{4}\right)\right)$

13 : $\left\langle c_j\right\rangle^B\leftarrow\Pi^{Bernoulli}\left(\langle p_{Bernoulli}\rangle^{B,FL}\right)$

14 : $\left\langle cond_{c_j==1}\right\rangle^B\leftarrow\neg\left\langle c_j\right\rangle^B$

15 : $\left\langle fg_j\right\rangle^B\leftarrow\left\langle cond_{x_{min}\leq x_j\leq x_{max}}\right\rangle^B\wedge\left\langle cond_{c_j==1}\right\rangle^B$

16 : $\langle i\rangle^{B,INT}\leftarrow\Pi^{ObliviousSelection}\left(\langle x_0\rangle^{B,INT},\ldots,\langle x_{ITER-1}\rangle^{B,INT},\langle fg_0\rangle^B,\ldots,\langle fg_{ITER-1}\rangle^B\right)$

Protocol 3.7: SMPC protocol for sampling symmetrical binomial random variable $i\sim SymmBino\left(n,p=0.5\right)$.

SMPC Protocol for Integer-Scaling Gaussian Mechanism We integrate the protocols to construct the complete SMPC protocol for the Integer-Scaling Gaussian mechanism $\Pi^{IsGauss}$.

Integer-scaling Gaussian mechanism can be reformulated in secret sharing from as:

$$\langle M_{ISGauss}\left(f_r\left(D\right),r,\Delta_r,\varepsilon,\delta\right)\rangle=\langle f_r\left(D\right)\rangle+\langle i\rangle\cdot r, \tag{3.42}$$

where r, Δ_r, ε,δ are publicly known values. We assume that the parties have already computed $\langle f_r(D)\rangle$.

Protocol: $\Pi^{ISGauss}\left(\langle f_r(D)\rangle^{B,FL}, r, \sqrt{n}\right)$

Input: $\langle f_r(D)\rangle^{B,FL}$,r, $\sqrt{n}$

Output: $\langle M_{ISGauss}\rangle^{B,FL}$

1 : $\langle i\rangle^{B,UINT} \leftarrow \Pi^{SymmBinomial}\left(\sqrt{n}\right)$.

2 : $\left\langle Y_{SymmBinoNoise}\right\rangle^{B,FL} \leftarrow \Pi^{FL\_MUL}\left(\Pi^{UINT2FL}\left(\langle i\rangle^{B,UINT}\right), r\right)$.

3 : $\langle M_{ISGauss}\rangle^{B,FL} \leftarrow \Pi^{FL\_Add}\left(\langle f_r(D)\rangle^{B,FL}, \left\langle Y_{SymmBinoNoise}\right\rangle^{B,FL}\right)$.

Protocol 3.8: SMPC protocol for Integer-Scaling Gaussian mechanism.

3.6 SMPC Protocols for Discrete Laplace Mechanism

Recall that the discrete Laplace mechanism $M_{DLap}(f(D), r, \varepsilon) = f(D) + Y$ (cf. § 2.3) use discrete Laplace random variable Y to perturb $f(D)$, where Y can be generated with $Algo^{GeoExp}$.

We first provide the SMPC protocols for the above sampling algorithms and then the SMPC protocols for the discrete Laplace mechanism.

Protocol: $\Pi^{GeometricExp}(n,d)$

Input: n, d

Output: $\langle x\rangle^{B,UINT}$, **where** $x \sim Geo\left(1-e^{-\frac{n}{d}}\right)$

1 : IF $d == 1$

2 : GO**TO** Line 7 // 1th loop terminates

3 : **FOR** $j \leftarrow 0$ **TO** $ITER_1 - 1$

4 : $\left\langle u_j\right\rangle^{B,UINT} \leftarrow \Pi^{RandInt}(d)$

5 : $\left\langle b_j^1\right\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(\Pi^{FL\_Exp}\left(\Pi^{FL\_Div}\left(\Pi^{UINT2FL}\left(\langle u\rangle^{B,UINT}\right),-d\right)\right)\right)$

6 : $\left\langle c_j\right\rangle^{B} = \langle b_1\rangle^{B}$

7 : **FOR** $j \leftarrow 0$ **TO** $ITER_2 - 1$

8 : $\langle b_2\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(e^{-1}\right)$

9 : $b_j^2 \leftarrow \neg\left(\langle b_2\rangle^{B}\right)$

10 : $\langle u\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(\langle u_0\rangle^{B,UINT},\ldots,\left\langle u_{ITER_1-1}\right\rangle^{B,UINT},\left\langle b_j^1\right\rangle^{B},\ldots,\left\langle b_{ITER_1-1}^1\right\rangle^{B}\right)$

11 : $\langle k\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(0,\ldots,ITER_2-1,\left\langle b_0^2\right\rangle^{B},\ldots,\left\langle b_{ITER_1-1}^2\right\rangle^{B}\right)$

12 : $\langle x\rangle^{B,UINT} \leftarrow \Pi^{UINT\_Div}\left(\Pi^{UINT\_Add}\left(\langle u\rangle^{B,UINT},\Pi^{UINT\_Mul}\left(\langle k\rangle^{B,UINT},d\right)\right),n\right)$

Protocol 3.9: SMPC Protocol for sampling geometric random variable $x \sim Geo\left(1-e^{-\frac{n}{d}}\right)$.

Protocol: $\Pi^{DLap}\left(t=\frac{d}{n}\right)$

Input: n, d

Output: $\langle Y\rangle^{B,INT}$, **where** $Y \sim DLap\left(t=\dfrac{d}{n}\right)$

1 : **FOR** $j \leftarrow 1$ **TO** $ITER-1$

2 : $\left\langle s_j\right\rangle^{B} \leftarrow \Pi^{RandBits}(1)$

3 : $\left\langle m_j\right\rangle^{B,UINT} \leftarrow \Pi^{GeometricEXP}(n,d)$

4 : $\left\langle f\,g_j\right\rangle^{B} = \neg\left(\left(\left\langle s_j\right\rangle^{B} == 1\right)\wedge\left(\left\langle m_j\right\rangle^{B,UINT} == 0\right)\right)$

5 : $\langle m\rangle^{B,UINT} \,\|\, \langle s\rangle^{B} \leftarrow \Pi^{ObliviousSelection}\left(\langle m_0\rangle^{B,UINT} \,\|\, \langle s_0\rangle^{B},\ldots,\langle m_{ITER-1}\rangle^{B,UINT} \,\|\, \langle s_{ITER-1}\rangle^{B},\langle f\,g_0\rangle^{B},\ldots,\langle f\,g_{ITER-1}\rangle^{B}\right)$

6 : Set $\langle s\rangle^{B}$ as the sign bit of $\langle m\rangle^{B,UINT}$

7 : $\langle Y\rangle^{B,INT} \leftarrow \langle m\rangle^{B,UINT}$

Protocol 3.10: SMPC Protocol for sampling discrete Laplace random variable $x \sim DLap(t)$.

Protocol: $\Pi^{DLap}\left(\langle f(D)\rangle^{B,INT}, t\right)$

Input: $\langle f(D)\rangle^{B,INT}$, t

Output: $\left\langle M_{DLap}\right\rangle^{B,INT}$

1 : $\langle Y\rangle^{B,INT} \leftarrow \Pi^{DLap}(t)$

2 : $\left\langle M_{DLap}\right\rangle^{B,INT} \leftarrow \Pi^{INT\_Add}\left(\langle f(D)\rangle^{B,INT}, \langle Y\rangle^{B,INT}\right)$

Protocol 3.11: SMPC protocols for discrete Laplace mechanism.

3.7 SMPC Protocol for Discrete Gaussian Mechanism

We provide the SMPC protocols for sampling discrete Gaussian random variable and the discrete Gaussian mechanisms based on (cf. § 2.4).

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

Input: σ

Output: $\langle Y\rangle^{B,UINT}$, $Y \sim DGau(\mu = 0, \sigma)$

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

2 : **FOR** $j \leftarrow 0$ **TO** $ITER - 1$

3 : $\left\langle Y_j\right\rangle^{B,INT} \leftarrow \Pi^{DiscreteLap}(t)$

4 : $\left\langle b_j\right\rangle^{B} \leftarrow \Pi^{BernoulliEXP}\left(\frac{\left(\Pi^{UI2FP}\left(\left|\left\langle Y_j\right\rangle^{B,INT}\right|\right) - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}\right)$

5 : $\langle Y\rangle^{B,INT} \leftarrow \Pi^{ObliviousSelection}\left(\langle Y_0\rangle^{B,INT}, \ldots, \langle Y_{ITER-1}\rangle^{B,INT}, \langle b_0\rangle^{B}, \ldots, \langle b_{ITER-1}\rangle^{B}\right)$

Protocol 3.12: SMPC Protocol for sampling discrete Gaussian random variable $Y \sim DGauss(\mu = 0, \sigma)$.

Protocol: $\Pi^{DGauss}\left(\langle f(D)\rangle^{B,INT}, \sigma\right)$

Input: $\langle f(D)\rangle^{B,INT}$, σ

Output: $\langle M_{DGauss}\rangle^{B,INT}$

1 : $\langle Y\rangle^{B,INT} \leftarrow \Pi^{DGauss}(\sigma)$

2 : $\langle M_{DGauss}\rangle^{B,INT} \leftarrow \Pi^{INT\_Add}\left(\langle f(D)\rangle^{B,INT}, \langle Y\rangle^{B,INT}\right)$

Protocol 3.13: SMPC protocol for discrete Gaussian mechanism.

3.8 Security Discussion

In this section, we explain why our SMPC protocols can guarantee computational privacy and differential privacy, i.e., a semi-honest adversary learns nothing beyond the protocol's output, and the reconstructed output satisfies differential privacy. Generally, the computational privacy guarantee of our SMPC protocols follows directly from the Boolean GMW protocols. At the beginning of the SMPC-DP procedure, each user secret shares their input to N non-colluding computation parties. For computation parties, these secret-shared values are indistinguishable random values and leak no information about the users' private input. Then, the computation parties collaboratively generate different types of noise that is based on generating random Boolean GMW bit strings $\langle \boldsymbol{r_1} \rangle^B, \ldots, \langle \boldsymbol{r_N} \rangle^B$. As $\langle \boldsymbol{r_1} \rangle^B, \ldots, \langle \boldsymbol{r_N} \rangle^B$ are generated by each computation party locally and $\langle \boldsymbol{r_1} \rangle^B \oplus \langle \boldsymbol{r_2} \rangle^B \oplus \ldots \oplus \langle \boldsymbol{r_N} \rangle^B$ is publicly unknown to all computing parties, the generated random noise satisfies the computational privacy and guarantees the differential privacy of the reconstructed result.

List of Figures

List of Tables

List of Protocols

List of Abbreviations

SMPC Secure Multi-Party Computation

DP Differential Privacy

BMR Beaver, Micali and Rogaway

GMW Goldreich, Micali and Wigderson

OT Oblivious Transfer

C-OT Correlated Oblivious Transfer

R-OT Random Oblivious Transfer

MTs Multiplication Triples

SIMD Single Instruction Multiple Data

Bibliography

[ABZS12] M. ALIASGARI, M. BLANTON, Y. ZHANG, A. STEELE. **"Secure computation on floating point numbers"**. In: *Cryptology ePrint Archive* (2012).

[ACC+21] A. ALY, K. CONG, D. COZZO, M. KELLER, E. ORSINI, D. ROTARU, O. SCHERER, P. SCHOLL, N. SMART, T. TANGUY. **"Scale–mamba v1. 12: Documentation"**. 2021.

[AHLR18] G. ASHAROV, S. HALEVI, Y. LINDELL, T. RABIN. **"Privacy-Preserving Search of Similar Patients in Genomic Data"**. In: *Proceedings on Privacy Enhancing Technologies* 2018.4 (2018), pp. 104–124.

[ALSZ17] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. **"More efficient oblivious transfer extensions"**. In: *Journal of Cryptology* 30.3 (2017), pp. 805–858.

[AS19] A. ALY, N. P. SMART. **"Benchmarking privacy preserving scientific operations"**. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2019, pp. 509–529.

[BDK+18] N. BÜSCHER, D. DEMMLER, S. KATZENBEISSER, D. KRETZMER, T. SCHNEIDER. **"HyCC: Compilation of hybrid protocols for practical secure computation"**. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 847–861.

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. **"MOTION–A Framework for Mixed-Protocol Multi-Party Computation"**. In: *ACM Transactions on Privacy and Security* 25.2 (2022), pp. 1–35.

[Bea91] D. BEAVER. **"Efficient multiparty protocols using circuit randomization"**. In: *Annual International Cryptology Conference*. Springer. 1991, pp. 420–432.

[BHKR13] M. BELLARE, V. T. HOANG, S. KEELVEEDHI, P. ROGAWAY. **"Efficient garbling from a fixed-key blockcipher"**. In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 478–492.

[BK15] E. BARKER, J. KELSEY. **"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"**. en. 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.

[BMR90] D. BEAVER, S. MICALI, P. ROGAWAY. **"The round complexity of secure protocols"**. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 503–513.

[BP08] J. BOYAR, R. PERALTA. **"Tight bounds for the multiplicative complexity of symmetric functions"**. In: *Theoretical Computer Science* 396.1-3 (2008), pp. 223–246.

[BW18] B. BALLE, Y.-X. WANG. **"Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising"**. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 394–403.

[CKS20] C. L. CANONNE, G. KAMATH, T. STEINKE. **"The discrete gaussian for differential privacy"**. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15676–15688.

[Cov19] C. COVINGTON. **"Snapping Mechanism Notes"**. 2019.

[CRW04] G. CASELLA, C. P. ROBERT, M. T. WELLS. **"Generalized accept-reject sampling schemes"**. In: *Lecture Notes-Monograph Series* (2004), pp. 342–347.

[CS10] O. CATRINA, A. SAXENA. **"Secure computation with fixed-point numbers"**. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2010, pp. 35–50.

[CSS12] T.-H. H. CHAN, E. SHI, D. SONG. **"Privacy-preserving stream aggregation with fault tolerance"**. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2012, pp. 200–214.

[DEK20] A. DALSKOV, D. ESCUDERO, M. KELLER. **"Secure evaluation of quantized neural networks"**. In: *Proceedings on Privacy Enhancing Technologies* 2020.4 (2020), pp. 355–375.

[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: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY-A framework for efficient mixed-protocol secure two-party computation."** In: *NDSS*. 2015.

[Dwo06] C. DWORK. **"Differential privacy"**. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2006, pp. 1–12.

[EGK+20] D. ESCUDERO, S. GHOSH, M. KELLER, R. RACHURI, P. SCHOLL. **"Improved primitives for MPC over mixed arithmetic-binary circuits"**. In: *Annual International Cryptology Conference*. Springer. 2020, pp. 823–852.

[EKM+14] F. EIGNER, A. KATE, M. MAFFEI, F. PAMPALONI, I. PRYVALOV. **"Differentially private data aggregation with optimal utility"**. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 316–325.

[EKR17] D. EVANS, V. KOLESNIKOV, M. ROSULEK. **"A pragmatic introduction to secure multi-party computation"**. In: *Foundations and Trends® in Privacy and Security* 2.2-3 (2017).

[GMP16] I. GAZEAU, D. MILLER, C. PALAMIDESSI. **"Preserving differential privacy under finite-precision semantics"**. In: *Theoretical Computer Science* 655 (2016), pp. 92–108.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **"How to play ANY mental game"**. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987, pp. 218–229.

[GRS12] A. GHOSH, T. ROUGHGARDEN, M. SUNDARARAJAN. **"Universally utility-maximizing privacy mechanisms"**. In: *SIAM Journal on Computing* 41.6 (2012), pp. 1673–1693.

[Har78] J. F. HART. **"Computer approximations"**. Krieger Publishing Co., Inc., 1978.

[Hau18] J. HAUSER. **"Berkeley SoftFloat"**. In: (2018).

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending oblivious transfers efficiently"**. In: *Annual International Cryptology Conference*. Springer. 2003, pp. 145–161.

[IR89] R. IMPAGLIAZZO, S. RUDICH. **"Limits on the provable consequences of one-way permutations"**. In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. 1989, pp. 44–61.

[Isr06] O. (I. O. S. G. (ISRAEL)). **"Foundations of Cryptography: Volume 1, Basic Tools"**. Cambridge University Press, 2006.

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E.-S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical privacy-preserving indoor localization using outsourcing"**. In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 448–463.

[JMRO22] J. JIN, E. MCMURTRY, B. RUBINSTEIN, O. OHRIMENKO. **"Are We There Yet? Timing and Floating-Point Attacks on Differential Privacy Systems"**. In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 1547–1547.

[Kam20] G. KAMATH. **"Lecture 3 - Intro to Differential Privacy"**. 2020.

[KL51] S. KULLBACK, R. A. LEIBLER. **"On information and sufficiency"**. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[KS08] V. KOLESNIKOV, T. SCHNEIDER. **"Improved garbled circuit: Free XOR gates and applications"**. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, pp. 486–498.

[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.

[LP09] Y. LINDELL, B. PINKAS. **"A proof of security of Yao's protocol for two-party computation"**. In: *Journal of cryptology* 22.2 (2009), pp. 161–188.

[Mar04] P. MARKSTEIN. **"Software division and square root using Goldschmidt's algorithms"**. In: *Proceedings of the 6th Conference on Real Numbers and Computers (RNC'6)*. Vol. 123. 2004, pp. 146–157.

[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.

[MRT20] P. MOHASSEL, M. ROSULEK, N. TRIEU. **"Practical Privacy-Preserving K-means Clustering"**. In: *Proceedings on Privacy Enhancing Technologies* 2020.4 (2020), pp. 414–433.

[NPS99] M. NAOR, B. PINKAS, R. SUMNER. **"Privacy preserving auctions and mechanism design"**. In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. 1999, pp. 129–139.

[RR21] M. ROSULEK, L. ROY. **"Three halves make a whole? beating the half-gates lower bound for garbled circuits"**. In: *Annual International Cryptology Conference*. Springer. 2021, pp. 94–124.

[RSA78] R. L. RIVEST, A. SHAMIR, L. ADLEMAN. **"A method for obtaining digital signatures and public-key cryptosystems"**. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[RTG00] Y. RUBNER, C. TOMASI, L. J. GUIBAS. **"The earth mover's distance as a metric for image retrieval"**. In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[SS98] P. SAMARATI, L. SWEENEY. **"Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression"**. In: (1998).

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: efficient privacy-preserving similar sequence queries on outsourced genomic databases"**. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 2019, pp. 315–327.

[Ste87] J. M. STEELE. **"Non-Uniform Random Variate Generation (Luc Devroye)"**. 1987.

[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.

[Tea20a] G. D. P. TEAM. **"Google's differential privacy libraries"**. 2020.

[Tea20b] G. D. P. TEAM. **"Secure Noise Generation"**. 2020.

[VV17] P. VOIGT, A. VON DEM BUSSCHE. **"The eu general data protection regulation (gdpr)"**. In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10.3152676 (2017), pp. 10–5555.

[Wal74] A. J. WALKER. **"Fast generation of uniformly distributed pseudorandom numbers with floating-point representation"**. In: *Electronics Letters* 10.25 (1974), pp. 533–534.

[War65] S. L. WARNER. **"Randomized response: A survey technique for eliminating evasive answer bias"**. In: *Journal of the American Statistical Association* 60.309 (1965), pp. 63–69.

[Whi97] C. R. WHITNEY. **"Jeanne Calment, World's Elder, Dies at 122"**. 1997.

[Yao82] A. C. YAO. **"Protocols for secure computations"**. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164.

[Yao86] A. C.-C. YAO. **"How to generate and exchange secrets"**. In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE. 1986, pp. 162–167.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two halves make a whole"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 220–250.

A Appendix

A.1 MPC Protocols

Oblivious Array Access $\Pi^{ObliviousSelection}$ is inspired by the works [JLL+19; MRT20]. $\Pi^{ObliviousSelection}$ uses the inverted binary tree, i.e., the leaves represent input elements, and the root represents the output element. The tree has $\log_2 \ell$-depth for input array of length ℓ and each node hold two shares: $\langle c \rangle^B$ and $\langle \boldsymbol{y} \rangle^B$. Fig. A.1 shows an example of the inverted binary tree. For $i \in [0, \ell - 1]$ and $j \in \left[\log_2 \ell\right]$, the values of node $(\left(\left\langle c_{a,b} \right\rangle, \left\langle \boldsymbol{y_{a,b}} \right\rangle^B\right))$ in the j-th layer are computed with the value of two nodes (with value $\left(\langle c_a \rangle, \langle \boldsymbol{y_a} \rangle^B\right)$ and $\left(\langle c_b \rangle, \langle \boldsymbol{y_b} \rangle^B\right)$) in the $j-1$-th layer as follows:

$$\left(\left\langle c_{a,b} \right\rangle, \left\langle \boldsymbol{y_{a,b}} \right\rangle^B\right) = \begin{cases} \left(\langle c_a \rangle, \langle \boldsymbol{y_a} \rangle^B\right), & \text{if } \langle c_a \rangle == 1 \\ \left(\langle c_b \rangle, \langle \boldsymbol{y_b} \rangle^B\right), & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 1 \\ (0,0) & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 0, \end{cases} \tag{A.43}$$

which is equivalent to

$$\left\langle c_{a,b} \right\rangle = \langle c_a \rangle \oplus \langle c_b \rangle \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \tag{A.44}$$

$$\begin{aligned} \left\langle \boldsymbol{y_{a,b}} \right\rangle = \quad & (\langle c_a \rangle \oplus \langle c_b \rangle) \cdot \left(\langle \boldsymbol{y_a} \rangle^{B,UINT} \cdot \langle c_a \rangle \oplus \langle \boldsymbol{y_b} \rangle^{B,UINT} \cdot \langle c_b \rangle\right) \\ & \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \cdot \langle \boldsymbol{y_a} \rangle^{B,UINT} \end{aligned} \tag{A.45}$$

The nodes is evaluated from the 1th layer until the root.

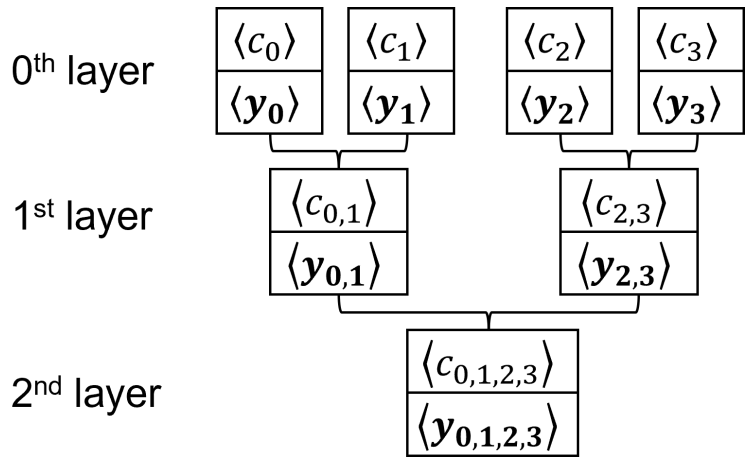

Figure A.1: Example inverted binary tree for $\Pi^{ObliviousSelection}$.