TECHNISCHE
UNIVERSITÄT
DARMSTADT

Master Thesis

Securely Realizing Output Privacy in MPC

Liang Zhao

July 11, 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 11, 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 Introduction

Technologies such as Machine Learning (ML) rely heavily on massive data analysis and pose severe privacy concerns as the individual's information in the highly centralized database may be misused. Privacy violation comes in many forms and is not directly visible. Therefore, it is crucial to provide appropriate privacy protections for user data. Many methods have been explored before to protect the privacy of individuals in the database. Since 2008, cryptographers have proposed Privacy-Preserving Machine Learning (PPML) algorithms to avoid the privacy breach of the training dataset based on SMPC. SMPC enables multiple parties to securely perform distributed computations with parties' private inputs so that only the computation results are revealed.

Let us consider a typical scenario of PPML: Alice wishes to investigate if she has genetic disorders while keeping her genomic data secret. As a service provider, Bob has trained an ML model that can predict genetic disorders given genomic data. However, Bob wants to keep his ML model private as it is his intellectual property that he aims to monetize. One unrealistic solution would be to rely on a trusted third party to analyze Alice's genetic data with Bob's ML model. However, as a trusted third party does rarely exist in practice, Alice and Bob can deploy a SMPC protocol to simulate a trusted third party.

Although SMPC can guarantee the users' computational privacy, an adversary can still infer users' sensitive information from the computation output. Shokri et al. [SSSS17] showed a membership inference attack that can determine if a data record was in the model's training dataset by making an adversarial usage of ML algorithms. One solution to mitigate such an attack is to deploy differentially private mechanisms. The concept of DP was introduced by Dwork et al. [Dwo06; DMNS06] that limits private information disclosure by adding calibrated noise to the revealed output. However, Mironov [Mir12] and Jin et al. [JMRO22] demonstrated a series of attacks against the differentially private mechanisms implemented under floating-point arithmetics.

To the best of our knowledge, most prior works [RN10; SCR+11; ÁC11; CSS12; EKM+14; WHWX16; BRB+17; JWEG18; TBA+19; KVH+21; YSMN21] that combine SMPC and DP do not consider the above security issues. This work attempts to fill this gap by providing *secure* noise generation methods in multi-party settings based on the state-of-the-art SMPC framework MOTION [BDST22]. The start point of this work is the secure noise generation methods and differentially private mechanisms from works [Mir12; Tea20b; CKS20]. The fundamental idea of secure noise generation is to generate discrete noise and re-scale it precisely under floating-point implementation to simulate continuous noise such that the noise satisfies the differential privacy requirements. We investigate the potential of applying

these noise generation methods in the SMPC. Besides, we aim to achieve DP and maintain the optimal utility of the computation result by adding a minimal amount of noise.

Contributions.

1. The noise is generated in a fully distributed manner that maintains the optimal utility of the aggregate statistics by introducing the minimal amount of noise required to satisfy DP. We consider the outsourcing scenario [KR11], i.e., the data owners first secret share their private inputs to multiple ($N \geq 2$) non-colluding computation parties, and the computation parties execute the SMPC protocols to compute the desired functionality and perturb the result. MOTION [BDST22] supports full-threshold security, and the computation result is secure if at least one computation party is honest and non-collusive. Therefore, the computation parties can jointly generate the shares of a publicly unknown noise with the same magnitude as the noise generated by a single trusted server.
2. We support a variety of differentially private mechanisms such as the (discrete) Laplace mechanism [CSS12; GRS12; DR$^+$14], the (discrete) Gaussian mechanism [DR$^+$14; CKS20], and the snapping mechanism [Mir12] in SMPC.
3. We implement fixed-point and floating-point operations in the binary circuit-based and the arithmetic sharing SMPC protocols and evaluate the performance. We use Single Instruction Multiple Data (SIMD) instructions to eliminate the independent iterations in the sampling algorithms and improve the protocol performance.

Thesis Outline. This thesis is organized as follows: Chapter 2 gives the preliminaries on the concept of secure multiparty computation and differential privacy with motivating examples and formal definitions. Chapter 3 presents a summary and discussion of the related works. Chapter 4 describes the details of the secure noise generation methods, differentially private mechanisms and our modifications. Chapter 5 provides the procedure to combine SMPC protocols and differentially private mechanisms, building blocks, and the SMPC protocols for differentially private mechanisms. Chapter 6 evaluates the performance of fixed-point and floating-point and the differentially private mechanisms. Chapter 7 concludes this work by summarizing the results and pointing out several directions for further research.

2 Preliminaries

In this chapter, we start with the notations used in this thesis § 2.1. Afterwards, we describe the basic knowledge of secure multi-party computation in § 2.2. Finally, we introduce the background knowledge and theory of differential privacy in § 2.3.

2.1 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. § 2.2.3): 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.

2.2 Secure Multi-Party Computation (SMPC)

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.

2.2.1 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.

2.2.2 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. § 2.2.3), we have $c = a \wedge b$ for Boolean sharing and $c = a \cdot b$ for arithmetic sharing (cf. § 2.2.3). Multiplication triples can be generated using C-OT (cf. § 2.2.2) in the two-party setting [DSZ15] or in the multi-party setting [BDST22].

2.2.3 MPC Protocols

We describe the SMPC protocols that are secure against $N-1$ semi-honest corruptions: Arithmetic sharing (cf. § 2.2.3), Boolean sharing with GMW (cf. § 2.2.3), and Yao sharing with BMR (cf. § 2.2.3). 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. § 2.2.2). 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. 2.1 shows, the gabler encrypts the keys of wire c and permutes the entries to generate Tab. 2.2. 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 2.1: 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 2.2: 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. § 2.2.2) 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.

2.2.4 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 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.

2.3 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.

2.3.1 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 2.3.1 (Continuous Uniform Distribution)**.** *The continuous uniform distribution with two boundary parameters a and b, has the following probability density function:*

$$Uni(x \mid a,b) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

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

$$Exp(x \mid \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x \ge 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{2.2}$$

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

$$\Pr(x \mid \lambda) = \begin{cases} 1 - e^{-\lambda x} & \text{for } x \ge 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{2.3}$$

Definition 2.3.3 (Laplace Distribution [DR+14])**.** *The Laplace distribution (centered at 0) with scale parameter b, has the following probability density function:*

$$Lap(x \mid b) = \frac{1}{2b} e^{\left(-\frac{|x|}{b}\right)} \tag{2.4}$$

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:

$$\Pr(x \le X \mid b) = \begin{cases} \frac{1}{2} e^{\frac{X}{b}} & \text{for } X \le 0 \\ 1 - \frac{1}{2} e^{-\frac{X}{b}} & \text{for } X > 0 \end{cases} \tag{2.5}$$

Definition 2.3.4 (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:*

$$\mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2.6}$$

Discrete Probability Distribution

Definition 2.3.5 (Bernoulli distribution)**.** *The Bernoulli distribution with parameter $p \in [0,1]$ has the following probability mass function:*

$$Bern(x \mid p) = \begin{cases} p & \text{for } x = 1 \\ 1-p & \text{for } x = 0 \end{cases} \tag{2.7}$$

Definition 2.3.6 (Binomial Distribution)**.** *The binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0,1]$ has the following probability mass function for $x \in \{0,1,2,\ldots,n\}$:*

$$Bino(x \mid n,p) = \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x} \tag{2.8}$$

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

$$Geo(x \mid p) = (1-p)^x\, p \tag{2.9}$$

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 2.3.8 (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:*

$$DLap(x \mid t) = \frac{e^{\frac{1}{t}} - 1}{e^{\frac{1}{t}} + 1} \cdot e^{-\frac{|x|}{t}} \tag{2.10}$$

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

$$DGau(x \mid \mu,\sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sum_{y\in\mathbb{Z}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}} \tag{2.11}$$

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:*

$$F^{-1}(u) = \inf\{x : F(x) = u, 0 < u < 1\} \tag{2.12}$$

If $U \sim Uni(0,1)$ (cf. 2.3.1), then $F^{-1}(U)$ has cumulative distribution function F.

Inverse Sampling from a Bernoulli Distribution. Algo. 2.1 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 2.1: 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. Algo. 2.2 [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 2.2: Sampling from a geometric distribution.

Sampling from a Discrete Laplace Distribution. Algo. 2.3 [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 2.3: Sampling from a discrete Laplace distribution.

2.3.2 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. 2.3 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 2.3: 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. 2.4 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. 2.4 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 2.4: $4-anonymous$ 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. 2.4, 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. 2.5 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 2.5: $3-diverse$ 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.

2.3.3 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:

$$Y_i = \begin{cases} X_i & \textit{with probability } p \\ 1 - X_i & \textit{with probability } 1-p \end{cases} \tag{2.13}$$

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:

$$Var\left[\tilde{C}\right] = Var\left[\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right]\right] = \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i]. \tag{2.14}$$

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

$$\begin{aligned} \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i] &= \frac{1}{4\gamma^2 n} Var[Y_i] \\ &\leq \frac{1}{16\gamma^2 n}. \end{aligned} \tag{2.15}$$

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

$$\Pr(|X-\mu| \geq t) \leq \frac{\sigma^2}{t^2}, \tag{2.16}$$

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

$$\begin{aligned} \Pr\left(\left|\tilde{C}-\overline{C}\right| \geq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\leq O(1), \\ \Pr\left(\left|\tilde{C}-\overline{C}\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\geq O(1), \end{aligned} \tag{2.17}$$

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. 2.6 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 2.6: 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 2.3.10 (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. 2.1 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 2.1: Query process of a database.

Definition 2.3.11 (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 2.3.12 (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 2.3.13 (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 2.3.14 (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 2.3.15 (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 § 2.3.3 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. 2.1:

- 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 2.1: 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. 2.2, 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.

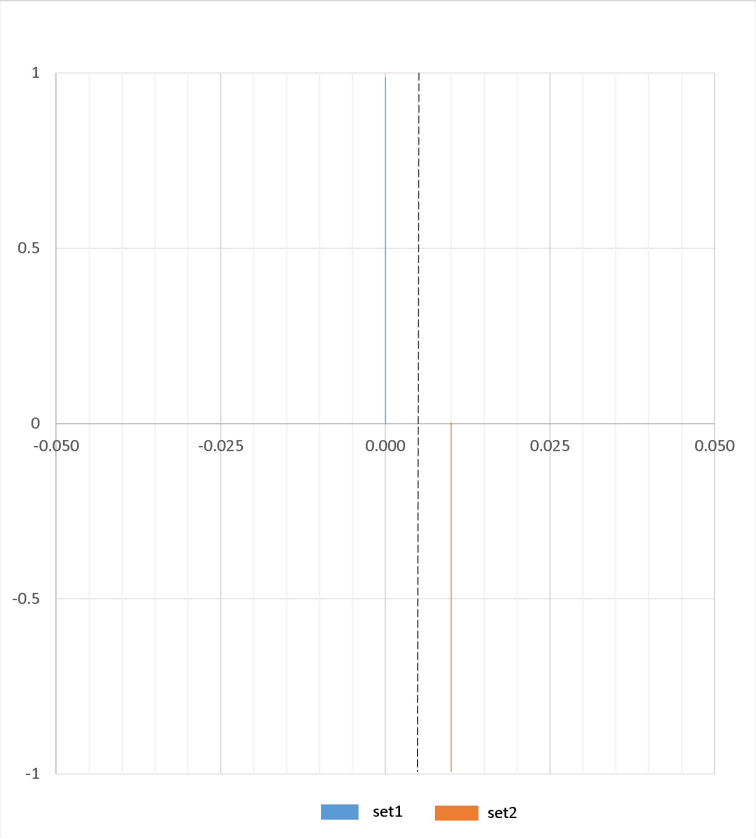

Figure 2.2: Deterministic algorithm.

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. 2.3 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. 2.4 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.

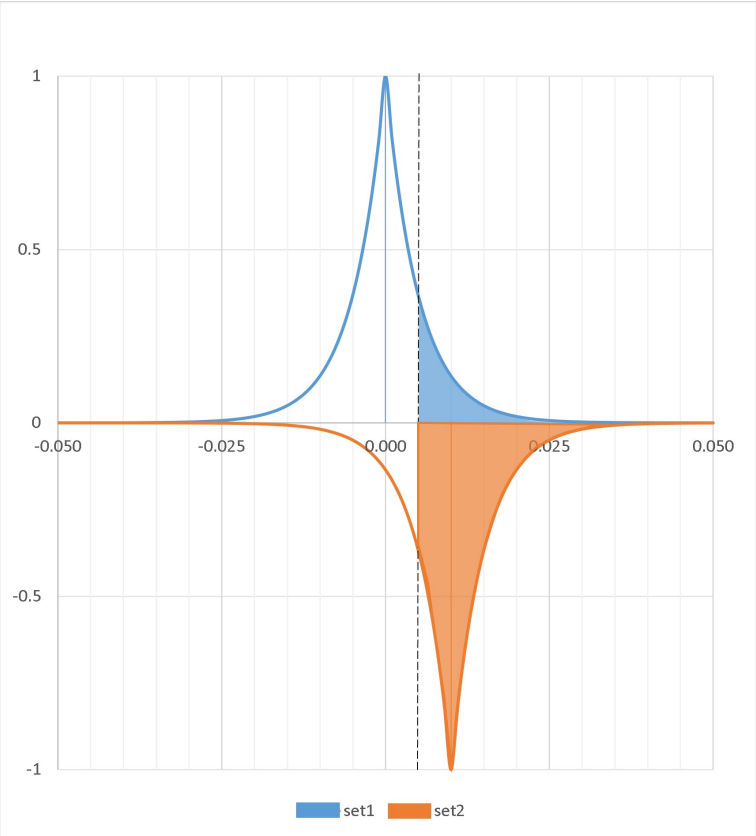

Figure 2.3: Indeterministic algorithm with small noise ($b = 0.005$).

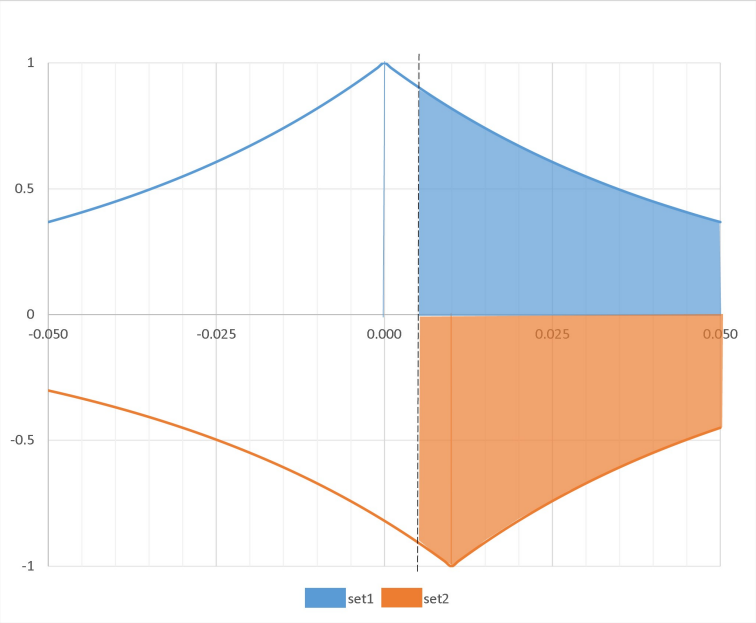

Figure 2.4: Indeterministic algorithm with large noise ($b = 0.05$).

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 2.3.16 (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 2.3.17 (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 $\mathcal{N}(Y_i \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{Y_i - \mu}{\sigma}\right)^2}$ with $\mu = 0$, $\sigma^2 = 2 \cdot \ln\left(\frac{1.25}{\delta} \cdot \left(\frac{\Delta_2^{(f)})^2}{\varepsilon^2}\right)\right)$.*

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 2.3.1 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 2.3.1, $M(D)$ 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.

3 Related Work

3.1 Distributed Differential Privacy (DDP)

The fundamental idea of differential privacy [Dwo06] is to perturb the query on a database such that the influence of each record in the database is bounded. The original definition of DP assumes the existence of a centralized and trusted server that manages the database and processes the queries. Subsequent work [DKM+06] extends DP to a distributed setting, where the central server is replaced by several mutually distrustful and potentially malicious computation parties. To the best of our knowledge, Dwork et al. [DKM+06] were the first to deploy malicious SMPC protocols to generate noise, compute the query with distributed data and perturb the query result. Generally, the works that attempt to realize Distributed Differential Privacy (DDP) can be categorized into two groups: the central DP model and the local DP model. In the central DP model, a trusted central server responsible for computing the aggregate statistics and perturbing it is simulated by several semi-honest (or malicious) computation parties with SMPC. It is typically assumed that the majority of computation parties are not colluding. However, SMPC incurs high computation and communication overhead that reduces the efficiency and scalability of the model. In the local DP model, the server is not trusted anymore. The users generate independent noise to privatize their data before sending it to the server. Therefore, the accuracy of the aggregate statistics is limited as the randomization is applied multiple times. Both models face a series of open challenges, and we discuss the details below.

3.1.1 Local DP Model

The existing solution to local DP model [RN10; SCR+11; ÁC11; CSS12; BRB+17; SCRS17; TBA+19] relied on homomorphic encryption, SMPC and the infinite divisibility of certain probability distribution (e.g., the Laplace distribution [KKP01] or the Gaussian distribution). Specifically, each user perturbs their data independently and encrypts it under a homomorphic encryption scheme such that the server can aggregate the encrypted data and only reveal the noisy aggregated result. Instead of using homomorphic encryption, other works [BP20; BBGN20; GKM+21] had the users mask the locally perturbed data with additional noise and send the masked data to the server. After the aggregation of the server, the additional noise is canceled out, and only the noisy result for satisfying DP is revealed. However, the existing works of the local DP model face two significant challenges. The first challenge is that the collusion users can subtract their noise term from the revealed result, weaken or break the

DP guarantee. Therefore, in order to achieve the required DP guarantee, each user has to add a larger amount of noise, which leads a reduced utility of the aggregated result. The second challenge is that the users have to pay a significant amount of computation effort, which makes the local DP model less practical for devices with limited computation power.

3.1.2 Central DP Model

For the central DP model, prior works [DKM+06; EKM+14; WHWX16; JWEG18; KVH+21; YSMN21; EIKN21] proposed a variety of methods to satisfy DP by generating distributed noise with SMPC. Dwork et al. [DKM+06] sampled noise from two distributions: Gaussian distribution (approximated with binomial distribution) and discrete Laplace distribution (approximated with Poisson distribution). To satisfy (ε, δ)-DP with Gaussian noise, it needs to process $n \geq 64 \log_2(2/\delta)/\varepsilon^2$ (e.g., $\varepsilon = 0.01$, $\delta = 0.0001 \Rightarrow n \approx 2^{23}$) uniform random bits in SMPC to generate binomial noise, that would lead to high SMPC overhead. For discrete Laplace noise, the protocol requires to securely evaluate a circuit in SMPC to generate biased bits. However, the evaluation of the circuit fails with non-zero probability and needs multiple iterations to make the failure probability negligible. Eigner et al. [EKM+14] proposed an architecture called PrivaDA, that combined DP and SMPC, and generated Laplace noise and discrete Laplace noise in SMPC protocols. However, the generated Laplace noise suffers from the floating-point attack [Mir12]. It remains to prove whether the discrete Laplace noise secure against the floating-point attack because its generation procedure is similar to the Laplace noise. Wu et al. [WHWX16] described methods for generating Bernoulli, Laplace, and Gaussian noise in SMPC. The Laplace noise is generated based on the central limit theory [AL06, Example 10.3.2], i.e., the aggregation of n Bernoulli random variable $Bern(0.5)$ approximates a normal random variable $\mathcal{N}\left(0, \frac{1}{4}\right)$ because $\sqrt{n}\left(\frac{\sum_{i=1}^{n} Bern(0.5)}{n} - \mu\right) \approx \mathcal{N}\left(0, \frac{1}{4}\right)$ when $n \to \infty$. However, there is no discussion about how to choose n without affecting the DP guarantee in their work.

Jayaraman et al. [JWEG18], Knott et al. [KVH+21], and Yuan et al. [YSMN21] presented distributed learning approaches that combine SMPC and DP by generating distributed Laplace noise and Gaussian noise with SMPC protocols. The protocols for Laplace noise are similar to the work of Eigner et al. [EKM+14], and the protocols for Gaussian noise are all based on the Box-Muller sampling algorithm [BOX58]. However, Jin et al. [JMRO22] demonstrated a floating-point attack against the Box-Muller method. Eriguchi et al. [EIKN21] provided SMPC protocols to generate two types of noise: the Finite Discrete Laplace (FDL) noise and the binomial noise. In contrast to the discrete Laplace distribution that can sample arbitrarily large integers with very low probability, FDL can only generate integers in a given range $[-N, N]$. The protocol for binomial noise deploys pseudorandom secret-sharing [CDI05] for generating shares of uniform random variables. However, the binomial noise only satisfies computational differential privacy [MPRV09], that is a relaxation of the standard differential privacy definition [DR+14] and only secure against computationally bounded adversaries. Our work provides an alternative solution to the central DP model by *securely* generating distributed noise that is not affected by the attacks [Mir12; JMRO22].

3.2 Arithmetic Operations in SMPC

Generally, most SMPC protocols that support arithmetic operations in SMPC are based on the binary circuit approach or the Linear Secret Sharing Scheme (LSSS). In the binary circuit based approach, the arithmetic operations are represented as a Boolean circuit and evaluated with Yao's Garbled circuit protocol [Yao86] (BMR [BMR90] for multi-party settings) or Boolean GMW protocol [GMW87]. By contrast, in the LSSS-based approach [CCD88; BGW88], the parties split their secret values into shares over a field $\mathbb{F}_q$ (or ring $\mathbb{Z}_{2^\ell}$) and send it to each of the parties. Next, we introduce the relevant works of arithmetic operations.

3.2.1 Binary Circuit Based SMPC

The binary circuits are typically composed of XOR, NOT, and AND gates, where the AND gates cause the primary cost. Therefore, the binary circuits should be optimized based on the cost metric of SMPC protocols. One method to generate low AND-depth and low AND-size binary circuits is to use the CBMC-GC [BHWK16] circuit compiler that derives circuit design with the C program. Pullonen and Siim [PS15] used CBMC-GC [BHWK16] circuit compiler to generate size-optimized circuits for IEEE 754 floating-point operations with the C library [Hau18; Fel21]. Archer et al. [AAS21] applied CBMC-GC [BHWK16] to generate depth-optimized circuits for IEEE 754 floating-point operations. In this work, we use CBMC-GC [AAS21] to generate depth-optimized circuits and size-optimized circuits for fixed-point and floating-point operations. We also use the available depth-optimized circuits for floating-point operations from work [DSZ15].

3.2.2 LSSS-Based SMPC

In order to guarantee the efficiency of the SMPC protocols, prior works [CS10; Lie12; HLOW16; AS19; LFH+20] deployed fixed-point rather than floating-point to represent real number operations. Catrina and Saxena [CS10] built a series of fixed-point operations (e.g., addition, subtraction, multiplication, and division) by representing a fixed-point x as:$x = \overline{x} \cdot 2^{-f}$, where $\overline{x}$ is an integer in a finite field $\mathbb{F}_q$, and f is the length of the fraction bits. The works [Lie12; HLOW16; AS19; LFH+20] proposed protocols for fixed-point operations such as exponential, square root, natural logarithm, and trigonometric functions with polynomial approximation [Har78] or Goldschmidt approximation [Mar04].

Another line of works [ABZS12; KW14; KW15; RBS+22] focused on floating-point operations. Aliasgari et al. [ABZS12] used a quadruple (v, p, z, s) to represent a floating-point number u as: $u = (1-2s) \cdot (1-z) \cdot v \cdot 2^p$. v, p, z, and s are the mantissa, exponent, zero bit, and sign bit of u. Aliasgari et al. [ABZS12] also provided SMPC protocols for operations such as addition, subtraction, multiplication, divisibility, square root, logarithm, and exponentiation. The subsequent works [KW14; KW15; RBS+22] applied a similar form to represent the floating-point numbers. Truex et al. [TBA+19] proposed a hybrid method combing fixed-point

and floating-point arithmetic, i.e., representing the mantissa of a floating-point number as a fixed-point number, and used LSSS-based fixed-point arithmetic when the mantissa is involved in the floating-point arithmetic. However, the fixed-point arithmetic is prone to overflow or underflow, and requires additional steps to correct the computation result that decreases the overall protocol performance. Kamm and Willemson et al. [KW15] provided SMPC protocols for square root, natural exponentiation, and error function by approximating the functions with Taylor series expansion or Chebyshev polynomials.

Rathee et al. [RBS+22] built a precise and efficient 32-bit floating-point operation library (SecFloat) for Two-Party Computation (2PC). One highlight is the mixed-bitwidth computation technique, i.e., using low bitwidth to represent numbers as much as possible. The bitwidth conversion operations between different bitwidth are performed with specialized zero-extension and truncation 2PC protocols. The second highlight is the use of low-degree polynomials to improve accuracy and efficiency. One common method to compute functions such as natural exponentiation is polynomial approximations, where high-degree polynomials yield more accurate results but incur more computation and communication effort. Rathee et al. [RBS+22] replaced the high-degree polynomials with low-degree piecewise polynomials without affecting the accuracy. In specifically, for input $x \in (a, b)$, they approximated functions using different low-degree polynomials of k subintervals $((a, a_1), (a_1, a_2), \ldots, (a_{k-1}, b))$. To determine the active interval of x, they deployed the Lookup Table (LUT) protocol [DKS+17] to compute the corresponding polynomial coefficients. To explore if the efficiency improvement techniques of SecFloat [RBS+22] can be extended to multi-party settings, we implement the building blocks of SecFloat such as conversion operations between low-bitwidth and high-bitwidth, multi-party lookup table protocol [KOR+17], and Most Significant Non-Zero Bit (MSNZB) [RRG+21] in MITION [BDST22].

Protocol $MSNZB\left(\langle x\rangle^A\right)$ computes the most significant non-zero bit index of arithmetic share $\langle x\rangle^A \in \mathbb{Z}_{2^\ell}$ and is a crucial building block of SecFloat. We implement two types of MSNZB protocols in MOTION [BDST22] and compare their performance difference:

1. $MSNZB\left(\langle x\rangle^A\right)$ [ABZS12] uses operations such as bit-decomposition, Boolean sharing to arithmetic sharing conversion and arithmetic sharing addition to compute the most significant non-zero bit index of x.
2. $MSNZB\left(\langle x\rangle^A\right)$ [RRG+21] first decomposes the input $\langle x\rangle^A \in \mathbb{Z}_{2^\ell}$ into $\frac{\ell}{8}$ low-bitwidth arithmetic shares $\langle x_1\rangle^A$, $\langle x_2\rangle^A$, …, $\left\langle x_{\ell/8}\right\rangle^A \in \mathbb{Z}_{2^8}$. Then, each low-bitwidth arithmetic share is used as the input to the lookup table protocol [KOR+17] to compute $MSNZB\left(\langle x_1\rangle^A\right), \ldots, MSNZB\left(\left\langle x_{\ell/8}\right\rangle^A\right)$. Finally, $MSNZB\left(\langle x_1\rangle^A\right), \ldots, MSNZB\left(\left\langle x_{\ell/8}\right\rangle^A\right)$ are combined together to compute $MSNZB\left(\langle x\rangle^A\right)$.

Table 3.1: Online run-times in milliseconds (ms) for protocol MSNZB for the GMW (A). We take the average over 10 protocol runs in the LAN and WAN environments.

	LAN		WAN	
Operation	N=3	N=5	N=3	N=5
$MSNZB$ [ABZS12]	18.41	80.12	886.06	1 036.23
$MSNZB$ [RBS+22]	359.76	567.14	5 436.82	6 355.90

We can see in Tab. 3.1 that the use of mixed-bitwidth and lookup table techniques does not bring efficiency improvement in the LAN (10Gbit/s Bandwidth, 1ms RTT) and WAN (100Mbit/s Bandwidth, 100ms RTT) networks. Details of benchmark environment are in § 6. The reason is as follows:

1. The conversion operations between low-bitwidth and high-bitwidth in SecFloat rely on a 2PC OT-based, highly efficient comparison protocol [RRK+20], and it can not be directly extended to multi-party settings.
2. In the two-party setting, when we take the value of two ℓ-bit arithmetic shares $\langle a \rangle_0^A$, $\langle a \rangle_1^A$ as plaintext values and compute the addition result in real number field, we need an $(\ell+1)$-bit integer $a = \langle a \rangle_0^A + \langle a \rangle_1^A$ to hold the addition result without overflow. The value of the most significant bit of a is used during the bitwidth conversion operation of SecFloat. For $N \geq 3$ parties, the addition result of N ℓ-bit arithmetic plaintext values needs a $\left(\lceil \log_2 N \rceil + l\right)$-bit integer to hold. The $\lceil \log_2 N \rceil$ most significant bits are used for the bitwidth conversion operation in the multi-party setting. Hence, as the number of parties grows, the complexity of conversion operations also increases.

Therefore, we build the LSSS-based floating-point operations in arithmetic sharing protocols with uniform bitwidth such as the work of Aliasgari et al. [ABZS12].

4 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 2.3.3). 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. § 5).

4.1 Snapping Mechanism

Recall that the Laplace mechanism (cf. 2.3.16) satisties ε-DP by adding a Laplace random variable $Y \sim Lap(\lambda)$ to the query function $f(D)$ for database D and $\lambda = \frac{\Delta_1^{(f)}}{\varepsilon}$:

$$M(D, \lambda) = f(D) + Y. \tag{4.18}$$

As discussed in paragraph 2.3.1, 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:

$$Y \leftarrow S \cdot \lambda \ln(U) \tag{4.19}$$

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:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)\right\rceil_\Lambda\right). \tag{4.20}$$

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. 2.3.15) 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:

$$U^* = (1.d_1 \ldots d_{52})_2 \times 2^{e-1023}. \tag{4.21}$$

As discussed above, floating-point U^* is sampled with a probability proportional to its ulp. We sample such floating-point numbers using Algo. 4.1 [Wal74; Mir12]. Specifically, we independently sample a geometric random variable $x \sim Geo(0.5)$ with Algo. 2.2 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 : $(d_1, \ldots, d_{52}) \xleftarrow{\$} \{0,1\}^{52}$

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 4.1: 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 $\sum_{i=1}^{\infty} \frac{1}{2^i} \approx 1$.

4.2 Integer-Scaling Mechanism

Google Differential Privacy Team [Tea20b] proposed practical implementation framework (this work names it as Integer-Scaling mechanisms) for Laplace mechanism (cf. 2.3.16) and Gaussian mechanism (cf. 2.3.17) 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:

$$M_{IS}\left(f\left(D\right), r, \varepsilon, \delta\right) = f_r\left(D\right) + ir, \tag{4.22}$$

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.

4.2.1 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:

$$M_{ISLap}\left(f\left(D\right), r, \Delta_r, \varepsilon\right) = f_r\left(D\right) + ir, \tag{4.23}$$

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. 2.3.8), $\Delta_r = r + \Delta_1^{(f)}$, and $\Delta_1^{(f)}$ is the ℓ_1-sensitivity (cf. 2.3.15) 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.

Algo. 4.2 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. 2.3.7).

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

$$\Pr\left(x \leq 2^{52} \mid x \sim Geo(p)\right) = 1 - e^{-\lambda \cdot 2^{52}+1} = \begin{cases} 0.\overline{9}_{(27)}8396\ldots & \text{for } \lambda = 2^{-48} \\ 0.\overline{9}_{(13)}8734\ldots & \text{for } \lambda = 2^{-49} \end{cases} \tag{4.24}$$

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

We require $\lambda \geq 2^{-48}$ which means that Algo. 4.2 fails (when required to generate $x > 2^{52}$) with a probability $p = 1 - 0.\overline{9}_{(27)}8396 \approx 2^{-92}$.

Algorithm Description. Algo. 4.2 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 Algo. 4.1) 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 4.2: 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. 2.3.8). Algo. 4.3 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 Algo. 4.2 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, Algo. 4.3 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 : $\quad g \leftarrow Geo\left(\frac{1}{t}\right)$

3 : $\quad s \leftarrow\$ \{-1, 1\}$

4 : $\quad$ **IF** $\neg(s == -1 \wedge g == 0)$

5 : $\quad\quad$ **RETURN** $i \leftarrow s \cdot g$ // success

6 : **RETURN** $i \leftarrow 0$ // failure

Algorithm 4.3: Sampling from a discrete Laplace distribution $DLap(t)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that Algo. 4.3 fails for $ITER = i$. Since each loop is independent, we estimate $\Pr(A_{ITER})$ as follows:

$$\begin{aligned}
\Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}(\Pr(A_1)) \\
&= \Pi_{i=1}^{ITER}(\Pr(s == -1) \cdot \Pr(g == 0)) \\
&= \Pi_{i=1}^{ITER}\left(\frac{1}{2} \cdot \Pr\left(0 \leftarrow Geo\left(1 - e^{-\lambda}\right)\right)\right) \\
&= \Pi_{i=1}^{ITER}\frac{1}{2}\left(1 - e^{-\lambda}\right) \\
&= \frac{1}{2^{ITER}}\left(1 - e^{-\lambda}\right)^{ITER}.
\end{aligned} \tag{4.25}$$

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

4.2.2 Integer-Scaling Gaussian Mechanism

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

$$M_{ISGauss}(f(D), r, \varepsilon, \delta) = f_r(D) + ir, \tag{4.26}$$

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. 2.3.6) 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 Algo. 4.4 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 Algo. 4.4 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 4.4: 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 Algo. 4.4 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:

$$\begin{aligned} \Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}(\Pr(A_1)) \\ &= \left(\frac{15}{16}\right)^{ITER} \end{aligned} \tag{4.27}$$

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.

4.3 Discrete Laplace Mechanism

In this section, we describe the discrete Laplace mechanism [CSS12; GRS12; EKM+14] and present a modified sampling algorithm Algo. 4.6 for generating discrete Laplace random variable.

The discrete Laplace mechanism is define as:

$$M_{DLap}\left(f\left(D\right), r, \varepsilon\right) = f\left(D\right) + Y, \tag{4.28}$$

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 Algo. 2.3, 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. Algo. 4.5 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. Algo. 4.5 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 Algo. 2.1 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 4.5: 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 Algo. 4.5 fails with a probability less than 2^{-40}, we first compute $\Pr(A_1)$ and $\Pr(B_1)$ as follows:

$$\begin{aligned}\Pr(A_1) &= \sum_{i=0}^{d-1} \Pr(u=i)\cdot\Pr(b_1=0)\\ &= \sum_{i=0}^{d-1} \frac{1}{d}\cdot\left(1-e^{-\frac{i}{d}}\right)\\ &= 1-\frac{1}{d}\sum_{i=0}^{d-1} e^{-\frac{i}{d}}\\ &= 1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\end{aligned} \tag{4.29}$$

$$\begin{aligned}\Pr\left(A_{ITER_1}\right) &= \prod_{i=1}^{ITER_2} \Pr(A_1)\\ &= \left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}\end{aligned} \tag{4.30}$$

$$\begin{aligned}\Pr\left(B_{ITER_2}\right) &= \prod_{i=1}^{ITER_2} \Pr(B_1)\\ &= \prod_{i=1}^{ITER_2} \Pr(b_2=1)\\ &= e^{-ITER_2}\end{aligned} \tag{4.31}$$

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

$$\begin{aligned}\Pr(Algo.\ 4.5) &= \Pr\left(A_{ITER_1}\vee B_{ITER_2}\right)\\ &= \Pr\left(A_{ITER_1}\right)+\Pr\left(B_{ITER_2}\right)-\Pr\left(A_{ITER_1}\right)\cdot\Pr\left(B_{ITER_2}\right)\\ &= \left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}+e^{-ITER_2}-\left(1-\frac{1}{d}\frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1}\cdot e^{-ITER_2}\end{aligned} \tag{4.32}$$

To guarantee $\Pr(Algo.\ 4.5) < 2^{-40}$, 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. Algo. 4.6 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. 2.3.8), where d and n are positive integers.

Algorithm Description. Algo. 4.6 is based on the same idea as Algo. 4.3 but uses Algo. 4.5 to generate the geometric random variable in line 3.

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

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

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

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

Algorithm 4.6: 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 Algo. 4.6 and Algo. 4.5 fail with a probability less than 2^{40}. Suppose A_i is an event that Algo. 4.6 fails in $ITER_1 = i$ iterations. As $\Pr(s == 1)$ and $\Pr(m == 0)$ are independent of each other, we have:

$$\begin{aligned}\Pr(A_1) &= \Pr(s == 1) \cdot \Pr(m == 0 \wedge Algo.\ 4.5 \text{ successes}) \\ &\quad + \Pr(m == 0 \wedge Algo.\ 4.5 \text{ fails}) \\ &= \frac{1}{2} \cdot \Pr(m == 0 \wedge Algo.\ 4.5 \text{ successes}) + \Pr(Algo.\ 4.5 \text{ fails}) \\ &= \frac{1}{2} \cdot \left(1 - e^{-\frac{n}{d}}\right) + \Pr(Algo.\ 4.5 \text{ fails})\end{aligned} \tag{4.33}$$

Finally, we have:

$$\begin{aligned}\Pr(Algo.\ 4.6 \text{ fails}) &= \Pr(A_{ITER}) \\ &= \prod_{i=1}^{ITER} \Pr(A_1) \\ &= \left(\frac{1}{2} \cdot \left(1 - e^{-\frac{n}{d}}\right) + \Pr(Algo.\ 4.5 \text{ fails})\right)^{ITER}\end{aligned} \tag{4.34}$$

To guarantee Algo. 4.6 and Algo. 4.5 fail with a probability $p < 2^{-40}$, it requires that $ITER = 30$ for $n = 3$ and $d = 2$.

4.4 Discrete Gaussian Mechanism

In this section, we describe the discrete Gaussian mechanism [CKS20] and the modified sampling algorithm Algo. 4.7. The discrete Gaussian mechanism is defined as:

$$M_{DGauss}(D) = f(D) + Y, \tag{4.35}$$

where $f(D) \in \mathbb{Z}$ and $Y \sim DGau(\mu = 0, \sigma)$ (cf. 2.3.9).

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. Algo. 4.7 is a modificaiton of the discrete Gaussian sampling algorithm from the work [CKS20] that samples $Y \sim DGau(\mu = 0, \sigma)$ (cf. 2.3.9). We replace the **WHILE** loop in the original work with **FOR** loop (line 2). We use Algo. 4.6 to generate the discrete Laplace random variable in line 3 and Algo. 2.1 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 : $L \leftarrow DLap(t)$
4 : $B \leftarrow Bern\left(e^{\left(|L| - \sigma^2/t\right)^2 / 2\sigma^2}\right)$
5 : **IF** $B == 1$
6 : **RETURN** $Y \leftarrow L$ // success
7 : **RETURN** $Y \leftarrow 0$ // failure

Algorithm 4.7: Sampling from a discrete Gaussian distribution $DGau(\mu = 0, \sigma)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that Algo. 4.7 fails in $ITER_1 = i$ iterations. We first compute $\Pr(A_1)$ as follows:

$$\begin{aligned}
\Pr(A_1) &= \sum_{i=0}^{\infty} \Pr(B = 0 \wedge L = i \wedge Algo.\ 4.6 \text{ successes}) + \Pr(Algo.\ 4.6 \text{ fails}) \\
&= \sum_{i=0}^{\infty} (\Pr(B = 0) \cdot \Pr(L = i) \cdot \Pr(Algo.\ 4.6 \text{ successes})) \\
&\quad + \Pr(Algo.\ 4.6 \text{ fails}) \\
&= \sum_{i=0}^{\infty} \left(1 - e^{-\frac{\left(|i| - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}} \right) \cdot \frac{\left(e^{\frac{1}{t}} - 1\right) \cdot e^{-\frac{|i|}{d}}}{e^{\frac{1}{t}} + 1} \cdot \Pr(Algo.\ 4.6 \text{ successes}) \\
&\quad + \Pr(Algo.\ 4.6 \text{ fails})
\end{aligned} \tag{4.36}$$

To guarantee that $\Pr(Algo.\ 4.7 \text{ fails}) < 2^{-40}$ and $\Pr(Algo.\ 4.6 \text{ fails}) < 2^{-40}$, it requires that $ITER = 23$ for $\sigma = 1.5$.s

5 SMPC Protocols for Differentially Private Mechanisms

In this chapter, we first present the SMPC-DP procedure that combines SMPC and differentially private mechanisms. Then, we construct the SMPC protocols for the previously introduced secure differentially private mechanisms and noise sampling algorithms (cf. § 4).

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 to determine the most efficient sampling algorithms for differentially private mechanisms in SMPC. 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] and compare their performance in [**sec:ArithmeticOperationsPerformanceEvaluation**]. For the second challenge, we construct the SMPC protocols for the all previously introduced noise sampling algorithms and compare their performance in § 6.2.

SMPC-DP Procedure Overview. In our SMPC-DP procedure, we consider n users, N computation parties and one reconstruction party as Fig. 5.1 shows. The SMPC-DP procedure consists of three steps: (i) The users U_i send the secret shared data $\langle D_i \rangle$ to the computation parties. (ii) The computation parties execute the SMPC protocols to compute function $\langle f\,(D_1, \ldots, D_n) \rangle$, generate secret shared noise $\langle noise \rangle$, and perturb $\langle f\,(D_1, \ldots, D_n) \rangle$. (iii) The computation parties send their share of the perturbed result $\langle f\,(D_1, \ldots, D_n) + noise \rangle$ to the reconstruction party, that reconstructs the perturbed result $f\,(D_1, \ldots, D_n) + noise$. The reconstruction party could be one of the computation parties. 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 $\langle D_i \rangle$ to the computation parties.

Privacy Goals. For users, computation parties, and the reconstruction party, the reconstructed perturbed result $f\,(D_1, \ldots, D_n) + noise$ should satisfy differential privacy, and the whole computation process leak no information about the individual users' private data D_i.

Attacker Model. Since MOTION [BDST22] is secure against $N - 1$ semi-honest corruptions, we assume that the computation parties are semi-honest, and there is at least one computation party that is not corrupted.

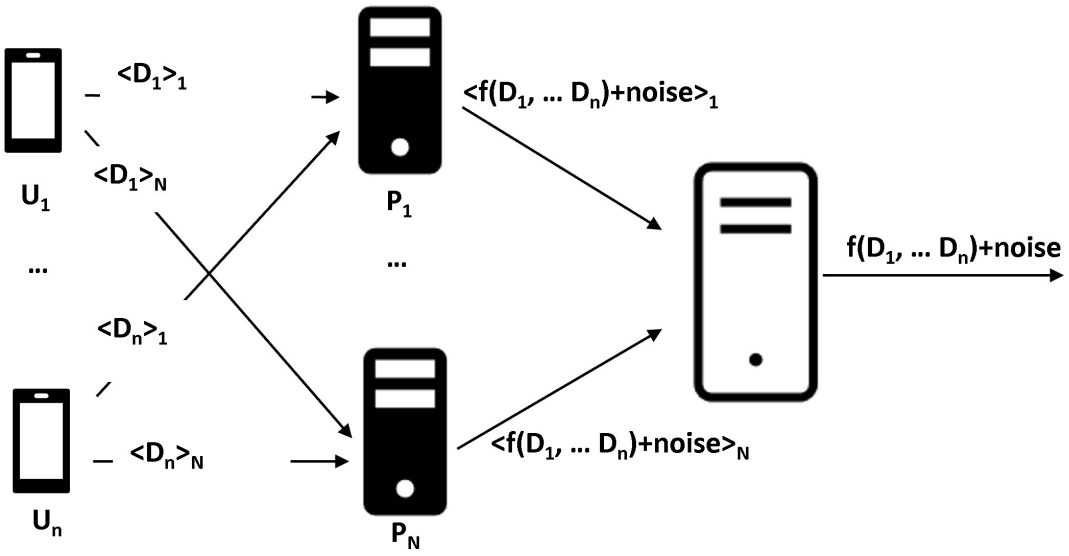

Figure 5.1: SMPC-DP procedure

5.1 Building Blocks

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

1. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{PreOr}\left(\langle \boldsymbol{x} \rangle^B\right)$ computes the prefix-OR of an ℓ-bit string $\boldsymbol{x} = (x_0, \ldots, x_{\ell-1})$ and output $\langle \boldsymbol{y} \rangle^B = \left(\langle y_0 \rangle^B, \ldots, \langle y_{\ell-1} \rangle^B\right)$ such that $y_j = \vee_{k=0}^{j} x_k$ for $j \in [0, \ell-1]$, where $y_0 = x_0$.
2. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{Geometric}$ generates shares of a geometric random variable $y \sim Geo\,(p = 0.5)$ using Algo. 2.2.
3. $\langle y \rangle^{B,UINT} \leftarrow \Pi^{HW}\,(x_0, \ldots, x_\ell)$ [BP08] computes the Hamming weight (i.e., the number of 1 bits) of an ℓ-bit string $x_0, \ldots, x_\ell$.
4. $\langle \boldsymbol{y_i} \rangle^B \leftarrow \Pi^{ObliviousSelection}\left(\langle \boldsymbol{y_0} \rangle^B, \ldots, \langle \boldsymbol{y_{\ell-1}} \rangle^B, \langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$ outputs the bit-string $\langle \boldsymbol{y_i} \rangle^B$, where i is the index of the first non-zero bit c_i for $i \in [0, \ell-1]$. If all the bits $c_0, \ldots, c_{\ell-1}$ are zeros, bit-string $\boldsymbol{y_i}$ is set to a bit-string consisting of all zeros.
5. $\langle y \rangle^{B,UINT} \leftarrow \Pi^{RandInt}\,(m)$ generate shares of a random integer y in the interval $[0, m-1]$.
6. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{COTMult}\left(\langle \boldsymbol{x} \rangle^B, \langle b \rangle^B\right)$ [AHLR18; ST19] computes the multiplication of a bit string $\boldsymbol{x}$ and a single bit b with correlated-OT (cf. § 2.2.2).
7. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{MUX}\left(\langle a \rangle^B, \langle \boldsymbol{x_0} \rangle^B, \langle \boldsymbol{x_1} \rangle^B\right)$ outputs bit-string $\boldsymbol{x_0}$ if $a = 1$, $\boldsymbol{x_1}$ otherwise.

5.2 Number Representations

In SMPC, we use Boolean Sharing with GMW (BGMW) and Arithmetic Sharing with GMW (AGMW) protocols to implement arithmetic operations. We consider integers, fixed-point numbers, and floating-point numbers to represent the numbers in the arithmetic operations.

BGMW Signed/Unsigned Integer. The MOTION framework [BDST22] supports BGMW unsigned integer arithmetic operations, such as $+$, $-$, $*$, $/$, and $<$. We extend it to support further arithmetic operations, e.g., modulo reduction and conversion operations with BGMW fixed-point and BGMW floating-point. We also extend MOTION [BDST22] to support BGMW signed integers with additional operations such as absolute value and negation.

BGMW 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 BGMW fixed-point and BGMW integers, ceil, floor, absolute value, round to the nearest integer, we use CBMC-GC [BHWK16] to generate depth-optimized binary circuits with the C code from Berkeley SoftFloat library [Hau18].

AGMW Floating-Point. We implement the AGMW floating-point protocols primarily based on the work of Aliasgari et al. [ABZS12]. A floating-point number u is represented as a quadruple (v, p, z, s), where each term is defined as follows:

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

This quadruple (v, p, z, s) represents the value $u = (1-2s)\cdot(1-z)\cdot v \cdot 2^p$. The SMPC protocols for floating-point in work [ABZS12] rely on the Shamir secret sharing [Sha79] operations performed over a prime field $\mathbb{F}_p$ (modulo p), whereas the arithmetic sharing operations in MOTION [BDST22] are performed in a ring (modulo $\mathbb{Z}_{2^n}$). One significant difference between finite field and ring is that there is no inverse element (i.e., a^{-1} in a prime field) in the ring. The inverse operations in the work [ABZS12] are primarily for computing the logical right shifting of the shares. Therefore, we replace the protocols that involve inverse operations in the work [ABZS12] with the SMPC protocols in works [EGK+20; DEK20; MRVW21] that support arithmetic sharing operations (e.g., logical/arithmetic right shifting, modulo reduction, etc.) over a ring.

BGMW Fixed-Point. We implement fixed-point numbers with a 48-bit signed integer and a 16-bit fractional part. This gives us precision 2^{-16} and covers the values with a integer part up to 47 bits. We use HyCC [BDK+18] to generate the depth-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 [BHWK16]. For operations such as square root, we use both polynomial approximations [Har78] and Goldschmidt approximations [Mar04; AS19]. The main issue with fixed-point arithmetic is the inherent precision loss during arithmetic operations. We estimate the accuracy of fixed-point exp2 operation by comparing its result to the floating-point exp2 operation with the same input. For 1000 random inputs in $[0, 1]$, the absolute error between the fixed-point exp2 operation results and floating-point exp2 operation results are $10^{-4} \approx 2^{-13}$ in average. For log2 (with 1000 samples in $[0.5, 1]$), the absolute error is $10^{-4} \approx 2^{-13}$, and for the square root operation with polynomial or Goldschmidt approximation, the absolute error is $10^{-4} \approx 2^{-13}$.

AGMW Fixed-Point. We implement the AGMW fixed-point protocols primarily based on the work of Catrina and Saxena [CS10]. A fixed-point number x is represented as $x = \overline{x} \cdot 2^{-f}$, where $\overline{x}$ is an k-bit signed integer and f is the length of the fractional part. We choose $(k, f) = (41, 20)$ as [ACC+21] recommended.

5.3 Random Number Generation

As discussed in § 4, the differentially private mechanisms rely heavily on the generation of random numbers. Generally, four types of random numbers are used in our protocols:

1. random BGMW bits,
2. random unsigned integers in a specific range,
3. uniform random fixed in range $(0, 1)$,
4. uniform random floating-point numbers in range $(0, 1)$.

Random Bit String. As Prot. 5.1 shows, to generate a random BGMW ℓ-bit string $\langle \boldsymbol{b} \rangle^B$ in a N-party setting, each computation party locally generates a uniform random ℓ-bit string $\boldsymbol{r_i}$ and set $\langle \boldsymbol{b} \rangle_i^B \leftarrow \boldsymbol{r_i}^B$.

Protocol: $\Pi^{RandBits}(\ell)$

Input: ℓ

Output: $\langle \boldsymbol{b} \rangle^B$**, where** $\boldsymbol{b}$ **is a uniform random** ℓ**-bit string**

1 : Each party P_i locally samples $\boldsymbol{r_i} \leftarrow\!\!\$\ \{0,1\}^\ell$

2 : Set $\langle \boldsymbol{b} \rangle_i^B \leftarrow \boldsymbol{r_i}$

Protocol 5.1: SMCP protocol for sampling random bit-string.

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

Random Unsigned Integer. To generate a random unsigned integer in range $(0, m)$, we use the Simple Modular Method [BK15], that first generates an uniform random ℓ-bit string $\langle \boldsymbol{r} \rangle^B$ ($\ell = 128$ or $\ell = 256$), and computes $\langle \boldsymbol{r} \rangle^B \mod m$ by taking $\langle \boldsymbol{r} \rangle^B$ as an ℓ-bit unsigned integer. We keep the security parameter $s = \ell - \log_2 m$ greater than 64 by limiting $m < 2^{64}$.

Uniform Random Fixed-Point Numbers. Suppose the fixed-point numbers have f fractional bits and $k - f$ integer bits. To generate such a uniform random fixed-point number in the range $[0, 1)$, we first generate f random bits and set it as the fractional part of the random fixed-point number and fill the $(k - f)$-bit integer part with zero bits.

Uniform Random Floating-Point Numbers. To generate a uniform random floating point in range $(0, 1)$, we follow Algo. 4.1 and construct corresponding SMPC protocol Prot. 5.2. 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 shares of a 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 5.2: SMCP protocol for sampling uniform random floating-point $U \in \mathbb{D} \cap (0, 1)$.

5.4 SMPC Protocols for Snapping Mechanism

Recall that the snapping mechanism (cf. § 4.1) is defined as follows:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U)\right\rceil_\Lambda\right). \tag{5.37}$$

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:

$$\langle M_S(f(D), \lambda, B)\rangle = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(\langle f(D)\rangle) \oplus \langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U\rangle)\right\rceil_\Lambda\right). \tag{5.38}$$

Parameters λ, Λ, and B are publicly known. The random sign $S \in \{0,1\}$ and the random floating-point number $U \in (0,1)$ can be generated with Prot. 5.1 and Prot. 5.2. Floating-point operations such as natural logarithm operation $LN(\cdot)$, addition ($\oplus$) and multiplication ($\otimes$) are available in both BGMW and AGMW protocols. Covington [Cov19] provided a plaintext implementation of function $\lfloor x\rceil_\Lambda$ that relies on the bit manipulation of the binary representation of floating-point numbers. Therefore, we choose BGMW protocols and generate corresponding depth-optimized circuits with CBMC-GC [BHWK16]. The BGMW protocol of function $\lfloor\langle x\rangle\rceil_\Lambda$ is denoted by $\Pi^{RoundToLambda}$. Prot. 5.3 provides an implementation of function clamp_B (cf. § 4.1) using BGMW floating-point arithmetic operations.

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

SMPC Protocols for Snapping Mechanism. We integrate the sub-protocols and present the BGMW protocol for the snapping mechanism in Prot. 5.4. We assume that the query function $\langle f(D)\rangle^{B,FL}$ is already computed.

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 5.4: SMPC protocol for snapping mechanism (BGMW).

5.5 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. § 4.2.1) re-scales a discrete Laplace random variable $i \sim DLap\left(t = \frac{\Delta_r}{r\varepsilon}\right)$ to simulate a continuous Laplace random variable, where i can be generated with Algo. 4.2 and Algo. 4.3. We provide the SMPC protocols for both sampling algorithms and the Integer-Scaling Laplace mechanism in the following part.

SMPC Protocols for Binary Search Based Geometric Sampling Algorithm

Prot. 5.5 samples a geometric random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ based on Algo. 4.2. First, each party locally computes L_0, R_0, M_0 and Q_0 in plaintext to save SMPC computations (line $1-7$). Then, the parties generate a uniform random floating-point U_0 with Prot. 5.2 and use the comparison result of U_0 and Q_0 to choose one subinterval (either $(L_0 \ldots M_0]$ or $(M_0 \ldots R_0]$) for further computation (line $8-12$). In line 13, the parties compute a flag fg_0 that indicates whether the termination condition of the **WHILE** loop (cf. Algo. 4.2, line 2) is satisfied. In Line $15-26$, the parties execute the binary search in SMPC completely for $ITER-1$ times. For $L_0 \leftarrow 0$ and $R_0 \leftarrow 2^5 2$, we set $ITER \leftarrow 52$ because the binary search takes at most $\log_2\left(2^{52}\right) = 52$ iteration to finish the search.

The $\langle M_j \rangle^{B,UINT}$ in line 15 is compute as follows:

$$\langle M_j \rangle^{B,UINT} = \langle L_{j-1} \rangle^{B,UINT} - \Pi^{FL2UI} \left(\frac{\ln\left(0.5 + 0.5 \cdot e^{-\lambda \cdot \Pi^{UI2FL}\left(\langle R_{j-1} \rangle^{B,UINT} - \langle L_{j-1} \rangle^{B,UINT}\right)}\right)}{\lambda} \right) \tag{5.39}$$

We first compute $\langle R_{j-1} \rangle^{B,UINT} - \langle L_{j-1} \rangle^{B,UINT}$ as unsigned integer, then convert the subtraction result to a floating-point number for natural exponentiation operation and division. Note that the division result is converted back to an unsigned integer because the integer addition and comparison operations are more efficient than that in the floating-point.

In line 19, the parties choose the correct split-point M_j as follows:

$$\begin{aligned} \langle M_j \rangle^{B,UINT} = \; & \Pi^{COTMult}\left(\langle cond_{M_j \leq L_{j-1}} \rangle^{B}, \Pi^{UINT\_Add}\left(\langle L_{j-1} \rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\langle cond_{M_j \geq R_{j-1}} \rangle^{B}, \Pi^{UINT\_Sub}\left(\langle R_{j-1} \rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\langle cond_{L_{j-1} < M_j < R_{j-1}} \rangle^{B}, \langle M_j \rangle^{B,UINT}\right) \end{aligned} \tag{5.40}$$

In line 24, the parties compute $\langle R_j \rangle^{B,UINT}$ as follows:

$$\begin{aligned} \langle R_j \rangle^{B,UINT} = \; & \Pi^{COTMult}\left(\langle cond_{U_j \leq Q_j} \rangle^{B}, \langle M_j \rangle^{B,UINT}\right) \\ & \oplus \Pi^{COTMult}\left(\langle cond_{U_j > Q_j} \rangle^{B}, \langle R_{j-1} \rangle^{B,UINT}\right) \end{aligned} \tag{5.41}$$

In line 25, the parties compute $\langle L_j \rangle^{B,UINT}$ as follows:

$$\begin{aligned} \langle L_j \rangle^{B,UINT} = \; & \Pi^{COTMult}\left(\langle cond_{U_j > Q_j} \rangle^{B}, \langle M_j \rangle^{B,UINT}\right) \\ & \oplus \Pi^{COTMult}\left(\langle cond_{U_j \leq Q_j} \rangle^{B}, \langle L_{j-1} \rangle^{B,UINT}\right) \end{aligned} \tag{5.42}$$

In line 27, the parties extract the correct result from $\left(\langle R_0 \rangle^{B,UINT}, \ldots, \langle R_{iter-1} \rangle^{B,UINT}\right)$ based on flags $\langle fg_0 \rangle^{B}, \ldots, \langle fg_{iter-1} \rangle^{B}$.

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

Input: λ

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

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

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

3: **IF** $M_0 \leq L_0$

4: $\quad M_0 \leftarrow L_0 + 1$

5: **ELSE IF** $M_0 \geq R_0$

6: $\quad M_0 \leftarrow R_0 - 1$

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

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

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

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

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

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

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

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

15: $\quad \left\langle M_j \right\rangle^{B,UINT} \leftarrow Eq.\ (5.39)$

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

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

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

19: $\quad \left\langle M_j \right\rangle^{B,UINT} \leftarrow Eq.\ (5.40)$

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

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

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

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

24: $\quad \left\langle R_j \right\rangle^{B,UINT} \leftarrow Eq.\ (5.41)$

25: $\quad \left\langle L_j \right\rangle^{B,UINT} \leftarrow Eq.\ (5.42)$

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

27: $\langle x \rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(\langle R_0 \rangle^{B,UINT}, \ldots \langle fg_0 \rangle^B, \ldots\right)$

Protocol 5.5: SMPC protocol for sampling from a geometric distribution $Geo\left(p = 1 - e^{-\lambda}\right)$ using binary search.

SMPC Protocols for Two-Side Geometric Sampling Algorithm

We construct Prot. 5.6 that converts a geometric random variable x to a discrete Laplace random variable i based on Algo. 4.3.

In line 2, 3, the parties generate the sign s_j and integer part g_j of a 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. Algo. 4.3) 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 : $\quad \langle s_j\rangle^B \leftarrow \Pi^{RandBits}(1)$.

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

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

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

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

Implementaion with SIMD. As discussed in § 4.2.1, Prot. 5.6 iterates for $ITER$ times to decrease the failure probability ($p < 2^{-40}$) for 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 Prot. 5.6.

We integrate the sub-protocols (Prot. 5.5 and Prot. 5.6) to construct the complete BGMW protocol for the Integer-Scaling Laplace mechanism. Integer-scaling Laplace mechanism can be reformulated in secret sharing as:

$$\left\langle M_{ISLap}\left(f_r(D), r, \Delta_r, \varepsilon\right)\right\rangle = \langle f_r(D)\rangle + \langle i\rangle \cdot r \tag{5.43}$$

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,$t = \dfrac{\Delta_r}{r\varepsilon}$

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 5.7: SMPC Protocol for Integer-Scaling Laplacian mechanism.

5.6 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. § 4.2.2) use a symmetric binomial random variable $i \sim SymmBino(n, p = 0.5)$ to simulate a Gaussian random variable. We provide the SMPC protocols for Algo. 4.4 and the Integer-Scaling Gaussian mechanism in following part.

SMPC Protocol for Symmetrical Binomial Sampling Algorithm

Prot. 5.8 samples a symmetric binomial random variable $x \sim SymmBino(n, p = 0.5)$ based on Algo. 4.4. Given value of $\sqrt{n}$, each party first computes following publicly known parameters:

$$\begin{aligned} m &= \left\lfloor \sqrt{2}\cdot\sqrt{n} + 1 \right\rfloor, \\ x_{min} &= -\frac{\sqrt{n}\cdot\sqrt{\ln\sqrt{n}}}{\sqrt{2}}, \\ x_{max} &= -x_{min}, \\ \nu_n &= \frac{0.4\cdot 2^{1.5}\cdot \ln^{1.5}(\sqrt{n})}{\sqrt{n}}, \\ \tilde{p}_{coe} &= \sqrt{\frac{2}{\pi}}\cdot(1-\nu_n)\cdot\frac{1}{\sqrt{n}}. \end{aligned} \tag{5.44}$$

Note that Prot. 5.8 can only be implemented with floating-point arithmetic because the intermediate value exceeds the value range of the built fixed-point numbers. In line 13, one way to generate the Bernoulli random variable c_j is to compare p_{Bern} with a uniform random variable $U \in (0, 1)$ (sampled with Prot. 5.2) as $c_j = (U < p_{Bern})$.

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

Input: $\sqrt{n}$

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

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

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

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

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

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

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

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

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

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

15 : $\left\langle f\,g_j\right\rangle^{B} \leftarrow \left\langle cond_{x_{min}\le x_j\le x_{max}}\right\rangle^{B} \wedge \left\langle cond_{c_j==1}\right\rangle^{B}$

16 : $\langle i\rangle^{B,INT} \leftarrow \Pi^{ObliviousSelection}\left(\langle x_0\rangle^{B,INT}, \ldots, \langle x_{ITER-1}\rangle^{B,INT}, \langle f\,g_0\rangle^{B}, \ldots, \langle f\,g_{ITER-1}\rangle^{B}\right)$

Protocol 5.8: SMPC protocol for sampling from a symmetrical binomial distribution $SymmBino\,(n, p = 0.5)$.

We use Prot. 5.8 to construct the SMPC protocol for the Integer-Scaling Gaussian mechanism. Integer-scaling Gaussian mechanism can be reformulated in secret sharing from as:

$$\langle M_{ISGauss}\,(f_r\,(D), r, \Delta_r, \varepsilon, \delta)\rangle = \langle f_r\,(D)\rangle + \langle i\rangle \cdot r, \tag{5.45}$$

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 5.9: SMPC protocol for Integer-Scaling Gaussian mechanism.

5.7 SMPC Protocols for Discrete Laplace Mechanism

Recall that the discrete Laplace mechanism $M_{DLap}(f(D), r, \varepsilon) = f(D) + Y$ (cf. § 4.3) use discrete Laplace random variable Y to perturb $f(D)$, where Y can be generated with Algo. 4.5. In this section, we provide the SMPC protocols for Algo. 4.5, Algo. 4.6 and 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 : **GOTO** 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 : $\langle u\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(\langle u_0\rangle^{B,UINT}, \ldots, \left\langle u_{ITER_1-1}\right\rangle^{B,UINT}, \left\langle b_0^1\right\rangle^{B}, \ldots, \left\langle b_{ITER_1-1}^1\right\rangle^{B}\right)$

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 5.10: SMPC Protocol for sampling from a geometric distribution $Geo\left(1-e^{-\frac{n}{d}}\right)$.

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

Input: n, d

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

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

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

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

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

5 : $\langle m \rangle^{B,UINT} \parallel \langle s \rangle^{B} \leftarrow \Pi^{ObliviousSelection}\left(\langle m_0 \rangle^{B,UINT} \parallel \langle s_0 \rangle^{B}, \ldots \langle fg_0 \rangle^{B}, \ldots\right)$

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 5.11: SMPC Protocol for sampling from a discrete Laplace distribution $DLap(t)$.

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 5.12: SMPC protocols for discrete Laplace mechanism.

5.8 SMPC Protocol for Discrete Gaussian Mechanism

Recall that the discrete Gaussian mechanism $M_{DGauss}(D) = f(D) + Y$ use § 4.4 to generate a discrete Gaussian random variable Y. We provide the SMPC protocols for § 4.4 and the discrete Gaussian mechanisms.

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

Input: σ

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

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 : $\left\langle b_j \right\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(\Pi^{FL\_Exp}\left(\frac{\left(\Pi^{UI2FL}\left(\left|\left\langle Y_j \right\rangle^{B,INT}\right|\right) - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}\right)\right)$

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

Protocol 5.13: SMPC Protocol for sampling from a discrete Gaussian distribuion $DGauss(\mu = 0, \sigma)$.

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 5.14: SMPC protocol for discrete Gaussian mechanism.

5.9 Security Discussion

In this section, we explain why our SMPC protocols can guarantee computational privacy and differential privacy. In other words, a semi-honest adversary learns nothing beyond the protocol's output, and the reconstructed output satisfies the differential privacy requirement. Generally, the computational privacy guarantee of our SMPC protocols follows directly from the BGMW and AGMW protocols. At the beginning of the SMPC-DP procedure, each user secret shares their input to N 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 compute the query function f and generate different types of noise that is fundamentally based on the shares of uniform random bits generated with Prot. 5.1. As the shares of random bits $\langle \boldsymbol{b} \rangle$ are actually generated by each party independently, the BGMW protocols guarantee that $\bigoplus_{i=1}^{N} \langle \boldsymbol{b} \rangle_i$ is publicly unknown to all computing parties if at least one computation party is non-collusive. Hence, the generated random noise is also publicly unknown to all the computation parties. If the noise is unknown

to all the parties and sampled precisely from a proper probability distribution as discussed in paragraph 4.2, the differential privacy requirement is satisfied.

6 Evaluation

In this chapter, we implement and measure the performance of all the protocols discussed in § 5 in a semi-honest scenario. First, we benchmark the performance of fixed-point and floating-point operations in different SMPC protocols (BGMW and AGMW) and choose the most efficient SMPC protocols to build the differentially private mechanisms. For comparison, we also implement the *insecure* differentially private mechanisms [EKM+14] in SMPC. Our code is public on GitHub [1];

Experimental Setup. The experiments are performed in five connected servers (Intel Core i9-7960X process, 128GB RAM, 10 Gbps network). We define two network settings to analyze the performance of our SMPC protocols.

1. LAN10: 10Gbit/s Bandwidth, 1ms RTT.
2. WAN: 100Mbit/s Bandwidth, 100ms RTT.

6.1 Arithmetic Operations Performance Evaluation

In this section, we compare the performance of arithmetic operations in different data types (BGMW fixed/floating-point, AGMW fixed/floating-point). All the arithmetic operations and underlying sub-protocols were tested for correctness with Google Test [2]. Specifically, the following arithmetic operations were benchmarked: addition, subtraction, multiplication, division, exp2, log2, square root, comparison (<), and data type conversions. We measure the total runtime (offline + online) in microseconds (ms) and average the runtime results over 10 iterations.

Fixed-Point Benchmarks. In the experiments, we use the BGMW fixed-point numbers with total bit-length k=64 and bit-length f=16 for the fractional part. For the AGMW fixed-point numbers, we use k=41 for the total bit-length and f=20 for the bit-length of the fractional part. We compare the BGMW and AGMW fixed-point arithmetic operations in Fig. 6.1. As Fig. 6.1 suggests, addition and subtraction are very fast for AGMW fixed-point because they could be computed locally with the precomputed MTs (cf. § 2.2.2). For the multiplication operation, the BGMW and AGMW fixed-point have very close performance,

[1] https://github.com/liangzhao-darmstadt/MOTION
[2] https://github.com/google/googletest

while the arithemtic truncation operation [CS10] is the major overhead of AGMW fixed-point. The division operation is an expensive operation for both BGMW and AGMW fixed-point, where the AGMW fixed-point is $1.21\times - 2.67\times$ faster than the BGMW fixed-point. The base-2 exponential function (exp2) and the binary logarithm (log2) are substantially faster for the BGMW than the AGMW fixed-point (exp2: $7.64\times - 11.98\times$, log2: $5.89\times - 7.43\times$). For the square root operation, the BGMW fixed-point and AGMW fixed-point have close performance in the 2PC and Three-Party Computation (3PC) settings, whereas BGMW fixed-point is $2.06\times$ faster than the AGMW fixed-point in the five-party settings. The comparision operation between BGMW and AGMW fixed-point are very close to each other, while the BGMW fixed-point is more efficient when the number of computation parties is greater than three. The conversion operation from fixed-point to integer (fx2int) is up to $22.93\times$ faster for the BGMW than the AGMW fixed-point. The remaining conversion operation (fx2fl) is up to $2.10\times$ faster for the AGMW than the BGMW fixed-point. Finally, we choose BGMW fixed-point as the primary data type for fixed-point operations because of the advantages in operations such as exp2 and fx2int.

Floating-Point Benchmarks. We use the BGMW floating-point numbers that have a 53-bit mantissa, an 11-bit exponent, and an 1-bit sign. In contrast, the AGMW floating-point numbers are represented by a quadruple (v, p, z, s) (cf. paragraph 5.2), where v, p, z, and s are represented by 128-bit arithmetic shares. As Fig. 6.2 shows, the BGMW floating-point is faster than the AGMW floating-point in almost all the arithmetic operations except for the division operation in 2PC settings. For addition, subtraction and multiplication operations, the BGMW floating-point is $1.59\times - 4.22\times$ faster than the AGMW floating-point. The primary overhead of AGMW is the truncation and bit-decomposition operations [ABZS12]. The division operation is slightly faster for the AGMW than the BGMW floating-point only in the five-party setting. For base-2 exponential function (exp2), binary logarithm (log2) and square root operations, the BGMW floating-point is substantially faster than the AGMW floating-point by a factor up to $13.94\times$. The reason for the low efficient performance of the AGMW floating-point is the multiply invocations of the floating-point multiplication in these operations [ABZS12]. The comparison operation of AGMW floating-point is more expensive than that in the BGMW floating-point. For the remaining conversion operations (fl2int and fl2fx), the BGMW floating-point is more efficient as the conversion operation to integer or fixed-point can be realized through bit manipulation at low cost. Based on the above analysis, we choose BGMW floating-point for floating-point arithmetic operations.

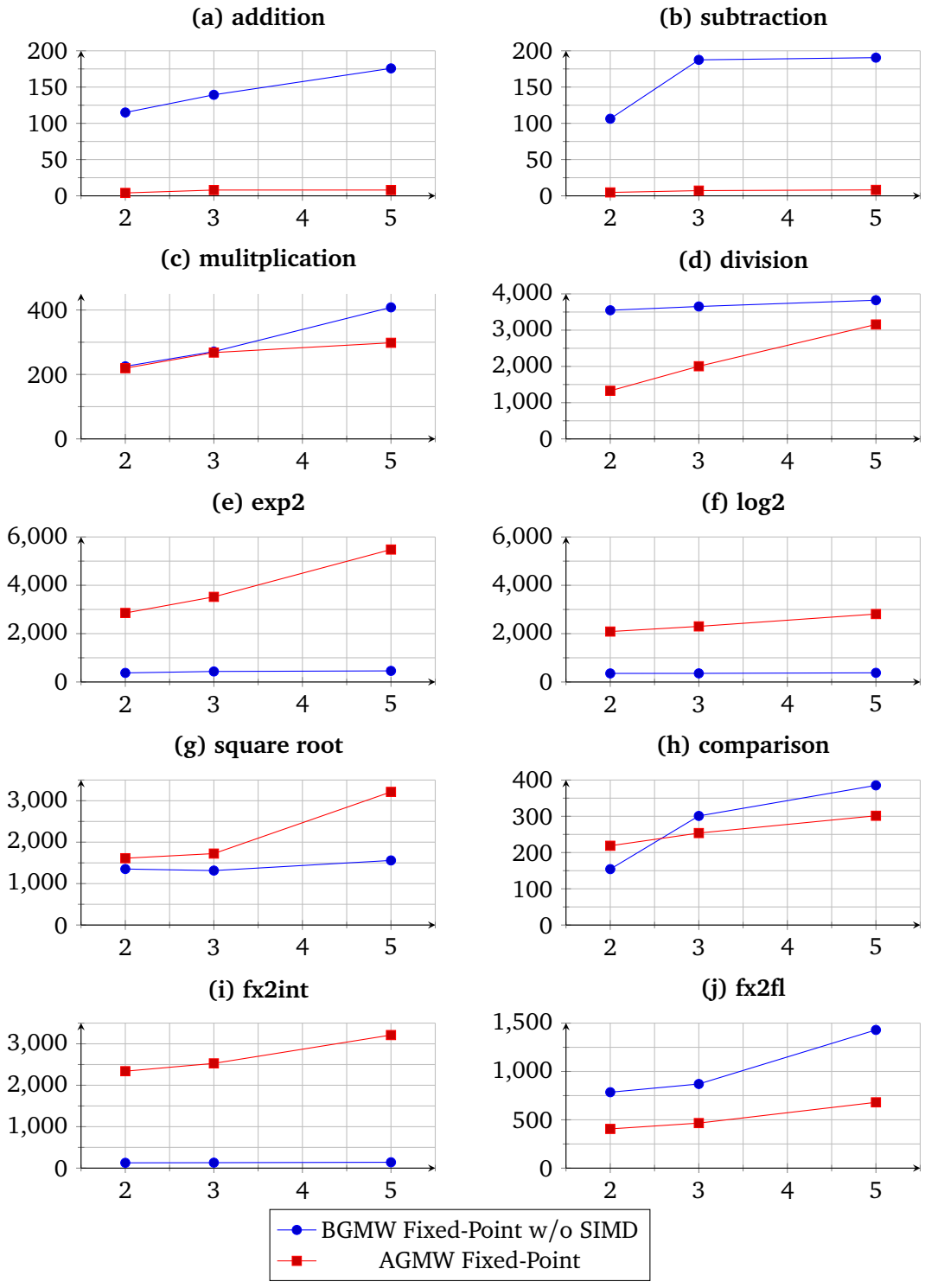

Figure 6.1: Total runtime in microseconds (y-axis) for the BGMW and AGMW fixed-point arithmetic operations in WAN test encironments with different parties (x-axis). We take the average results over 10 protocol runs.

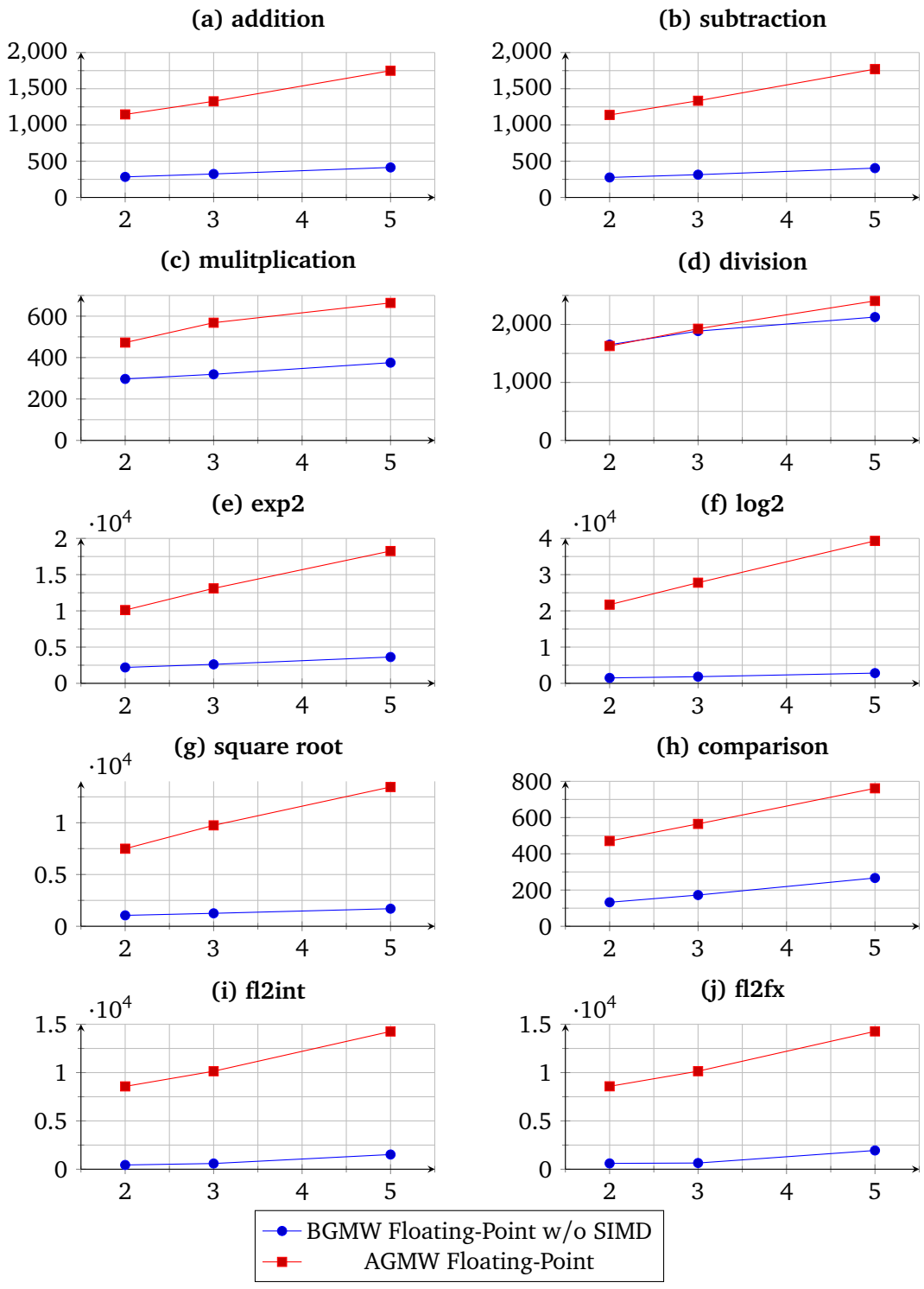

Figure 6.2: Total runtime in microseconds (y-axis) for the BGMW and AGMW floating-point arithmetic operations in WAN test encironments with different parties (x-axis). We take the average results over 10 protocol runs.

6.2 Differentially Pirvate Mechanism Benchmarks

In this seciton, we present the performance evaluation of the SMPC protocols for differentially private mechanisms.

Laplace Mechanisms. Recall that Integer-Scaling Laplace mechanism M_{ISLap} (cf. § 4.2.1) is a *secure* implementation of the Laplace mechanism using a re-scaled discrete Laplace random variable to simulate the continuous Laplace random variable.

In § 5, we construct two SMPC protocols (Prot. 5.6 and Prot. 5.11) for sampling discrete Laplace random variables. However, Prot. 5.6 incurs memory overflow ($>$ 128GB in 2PC) because its sub-protocol Prot. 5.5 use binary search (with 52 successive iterations) to generate a geometric random variable. In contrast, the **FOR** loop in Prot. 5.11 and its sub-protocol Prot. 5.10 can be parallelized with SIMD that improve the efficiency and decrease the memory overhead. Nevertheless, we can only generate ten discrete Laplace random variables with Prot. 5.11 without memory overflow for a single run. We use Prot. 5.11 to implemented the Integer-Scaling Laplace mechanism M_{ISLap}.

We can see that the Integer-Scaling Laplace mechanism M_{ISLap} with Prot. 5.11 under BGMW fixed-point is $1.07 \times -1.32\times$ slower than that with BGMW floating-point implementation. The major overhead of BGMW fixed-point is the division operation.

For comparison, we implement the *insecure* Laplace mechanisms M_{Lap} [EKM$^+$14]. Note that Mironov [Mir12] showed that M_{Lap} [EKM$^+$14] suffered from floating-point attacks and proposed the snapping mechanism M_{SM} as a solution.

The snapping mechanism M_{SM} has the best total runtimes performance and it is $554\times-1373\times$ faster than M_{ISLap} and $1.32 \times -1.62\times$ faster than M_{Lap} [EKM$^+$14]. However, it is worth to mention that the snapping mechanism M_{SM} introduces additional errors (beyond the necessary amount Laplace noise), that leads to a significant reduction in utility [Cov19; Tea20b]. In contrast, we could reduce the unnecessary errors of the Integer-Scaling Laplace mechanism M_{ISLap} by setting an appropriate resolution parameter r.

Gaussian Mechanisms. The Integer-Scaling Gaussian mechanism M_{ISLap} (cf. § 4.2.2) deploys the sampling protocol Prot. 5.8 under BGMW floating-point simulate the continuous Gaussian random variable. We can see that the Integer-Scaling Gaussian mechanism M_{ISGau} is $3,27 \times -4.11\times$ slower than the Integer-Scaling Laplace mechanism M_{ISLap} under BGMW floating-point.

Table 6.1: Online run-times in milliseconds (ms) for differentially private mechanisms for the GMW (B). We specify the run-time of a single operation amortized over corresponding Parallel Factor (PF) values. We take the average over 10 protocol runs in the LAN and WAN environments. *: insecure methods.

			LAN			WAN		
Parties N	PF	Protocol	N=2	N=3	N=5	N=2	N=3	N=5
M_{Lap} [EKM$^+$14] *	1000	B, FL	7.58	8.36	11.08	197.68	223.92	248.53
M_{SM} **(this work)**	1000	B, FL	4.68	5.77	—	145.70	168.74	182.60
M_{ISLap} **(this work)**	10	B, FX	7 292.91	7 925.98	—	95 448.89	110 170.46	—
M_{ISLap} **(this work)**	10	B, FL	5 770.09	5 981.37	—	85 892.69	93 238.29	—
M_{ISGau} **(this work)**	1	B, FL	18 916.06	19 413.08	26 929.39	394 248.57	435 126.28	470 813.95
M_{DLap} [EKM$^+$14]	1000	B, FX	2.26	3.26	—	19.01	24.11	69.88
M_{DLap} [EKM$^+$14]	1000	B, FL	5.91	6.60	9.06	144.93	163.96	177.91
M_{DLap} **(this work)**	10	B, FX	6 665.97	7 107.66	—	93 155.60	101 739.97	—
M_{DLap} **(this work)**	10	B, FL	5 394.89	5 749.11	—	85 606.52	92 518.90	—
M_{Dau} **(this work)**	1	B, Fx	28 159.07	29 575.94	—	562 198.23	617 524.26	—
M_{Dau} **(this work)**	1	B, FL	24 906.67	26 350.66	—	526 837.85	588 904.17	—

Discrete Laplace Mechanisms. The discrete Laplace mechanism M_{DLap} deploys the same sampling protocol Prot. 5.11 as M_{ISLap}. We also implement M_{DLap} [EKM$^+$14] for comparison. It can be seen, that M_{DLap} [EKM$^+$14] is at least 1764× faster than our implementation of M_{DLap} in BGMW floating-point in the LAN setting. However, the security of M_{DLap} [EKM$^+$14] remains to prove as it applies similar noise generation procedure as M_{Lap} [EKM$^+$14].

Discrete Gaussian Mechanisms. We encounter memory overflow when implement BGMW M_{DGau} with Prot. 5.14. To get a rough estimation, we split Prot. 5.14 into several independent parts, save the intermediate result, and run MOTION [BDST22] for several times. As Tab. 6.1 shows, the discrete Gaussian mechanism M_{DGau} takes the longest runtime. To successfully generate a discrete Gaussian random variable, Prot. 5.14 needs about 23 discrete Laplace random variables to be generated in its sub-protocol (cf. Prot. 5.11).

7 Final Remarks

This chapter presents the conclusion of this work and propose some suggestions for further research.

7.1 Conclusions

This work aims to realize DP in SMPC in a secure manner so that the computation result can preserve the users' privacy and maintain an optimal utility. We investigate the potential of realizing different secure noise generation methods in SMPC and provide a variety of SMPC protocols for differentially private mechanisms. As the benchmark results in § 6.1 indicates, BGMW floating-point is the most efficient SMPC protocols for implementing the sampling algorithms. However, the benchmark result of the differentially private mechanisms (cf. § 6.2) implies that the generation of secure noise in SMPC incurs a significant amount of overhead than the *insecure* methods.

7.2 Future Research

In future works, we first intend to redesign the interfaces of MOTION [BDST22] to optimize the memory overhead. Another important research direction is to explore the possibilities of extending the SMPC protocols for differentially private mechanisms into malicious settings. Recall that the differentially private mechanisms we implemented satisfy the standard differential privacy definition [Dwo06; DMNS06], i.e., they are secure against computationally unbounded adversary. However, a computationally bounded adversary might be more practical in specific scenarios where the computation result's efficiency and utility have a higher priority than the security. Therefore, it is worth to investigate if an relaxation of the standard differential privacy can improve the efficiency of the SMPC protocols.

List of Figures

List of Tables

List of Protocols

List of Abbreviations

SMPC Secure Multi-Party Computation

DP Differential Privacy

DDP Distributed Differential Privacy

BGMW Boolean Sharing with GMW

AGMW Arithmetic Sharing with GMW

PPML Privacy-Preserving Machine Learning

ML Machine Learning

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

FDL Finite Discrete Laplace

LSSS Linear Secret Sharing Scheme

2PC Two-Party Computation

3PC Three-Party Computation

LUT Lookup Table

MSNZB Most Significant Non-Zero Bit

SIMD Single Instruction Multiple Data

Bibliography

[AAS21] D. W. ARCHER, S. ATAPOOR, N. P. SMART. **"The Cost of IEEE Arithmetic in Secure Computation"**. In: *International Conference on Cryptology and Information Security in Latin America*. Springer. 2021, pp. 431–452.

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

[ÁC11] G. ÁCS, C. CASTELLUCCIA. **"I have a dream!(differentially private smart metering)"**. In: *International Workshop on Information Hiding*. Springer. 2011, pp. 118–132.

[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.

[AL06] K. B. ATHREYA, S. N. LAHIRI. **"Measure theory and probability theory"**. Vol. 19. Springer, 2006.

[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.

[BBGN20] B. BALLE, J. BELL, A. GASCÓN, K. NISSIM. **"Private summation in the multi-message shuffle model"**. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 657–676.

[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.

[BGW88] M. BEN-OR, S. GOLDWASSER, A. WIGDERSON. **"Completeness theorems for non-cryptographic fault-tolerant distributed computation"**. In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 1988, pp. 1–10.

[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.

[BHWK16] N. BÜSCHER, A. HOLZER, A. WEBER, S. KATZENBEISSER. **"Compiling low depth circuits for practical secure computation"**. In: *European Symposium on Research in Computer Security*. Springer. 2016, pp. 80–98.

[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.

[BOX58] G. BOX. **"A Note on the Generation of Random Normal Deviates"**. In: *Annals of Mathematical Statistics* 29 (1958), pp. 610–611.

[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.

[BP20] D. BYRD, A. POLYCHRONIADOU. **"Differentially private secure multi-party computation for federated learning in financial applications"**. In: *Proceedings of the First ACM International Conference on AI in Finance*. 2020, pp. 1–9.

[BRB+17] V. BINDSCHAEDLER, S. RANE, A. E. BRITO, V. RAO, E. UZUN. **"Achieving differential privacy in secure multiparty data aggregation protocols on star networks"**. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 2017, pp. 115–125.

[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.

[CCD88] D. CHAUM, C. CRÉPEAU, I. DAMGARD. **"Multiparty unconditionally secure protocols"**. In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 1988, pp. 11–19.

[CDI05] R. CRAMER, I. DAMGÅRD, Y. ISHAI. **"Share conversion, pseudorandom secret-sharing and applications to secure computation"**. In: *Theory of Cryptography Conference*. Springer. 2005, pp. 342–362.

[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.

[DKM+06] C. DWORK, K. KENTHAPADI, F. MCSHERRY, I. MIRONOV, M. NAOR. **"Our data, ourselves: Privacy via distributed noise generation"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, pp. 486–503.

[DKS+17] G. DESSOUKY, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER, S. ZEITOUNI, M. ZOHNER. **"Pushing the Communication Barrier in Secure Computation using Lookup Tables"**. In: *NDSS* (2017).

[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.

[EIKN21] R. ERIGUCHI, A. ICHIKAWA, N. KUNIHIRO, K. NUIDA. **"Efficient Noise Generation to Achieve Differential Privacy with Applications to Secure Multiparty Computation"**. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2021, pp. 271–290.

[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).

[Fel21] R. FELKER. **"musl libc"**. 2021.

[GKM+21] B. GHAZI, R. KUMAR, P. MANURANGSI, R. PAGH, A. SINHA. **"Differentially private aggregation in the shuffle model: Almost central accuracy in almost a single message"**. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 3692–3701.

[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).

[HLOW16] B. HEMENWAY, S. LU, R. OSTROVSKY, W. WELSER IV. **"High-precision secure computation of satellite collision probabilities"**. In: *International Conference on Security and Cryptography for Networks*. Springer. 2016, pp. 169–187.

[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.

[JWEG18] B. JAYARAMAN, L. WANG, D. EVANS, Q. GU. **"Distributed learning without distress: Privacy-preserving empirical risk minimization"**. In: *Advances in Neural Information Processing Systems* 31 (2018).

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

[KKP01] S. KOTZ, T. KOZUBOWSKI, K. PODGÓRSKI. **"The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance"**. 183. Springer Science & Business Media, 2001.

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

[KOR+17] M. KELLER, E. ORSINI, D. ROTARU, P. SCHOLL, E. SORIA-VAZQUEZ, S. VIVEK. **"Faster secure multi-party computation of AES and DES using lookup tables"**. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 229–249.

[KR11] S. KAMARA, M. RAYKOVA. **"Secure outsourced computation in a multi-tenant cloud"**. In: *IBM Workshop on Cryptography and Security in Clouds*. 2011, pp. 15–16.

[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.

[KVH+21] B. KNOTT, S. VENKATARAMAN, A. HANNUN, S. SENGUPTA, M. IBRAHIM, L. v. d. MAATEN. **"Crypten: Secure multi-party computation meets machine learning"**. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4961–4973.

[KW14] T. KRIPS, J. WILLEMSON. **"Hybrid model of fixed and floating point numbers in secure multiparty computations"**. In: *International Conference on Information Security*. Springer. 2014, pp. 179–197.

[KW15] L. KAMM, J. WILLEMSON. **"Secure floating point arithmetic and private satellite collision analysis"**. In: *International Journal of Information Security* 14.6 (2015), pp. 531–548.

[LFH+20] W.-j. LU, Y. FANG, Z. HUANG, C. HONG, C. CHEN, H. QU, Y. ZHOU, K. REN. **"Faster secure multiparty computation of adaptive gradient descent"**. In: *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*. 2020, pp. 47–49.

[Lie12] M. LIEDEL. **"Secure distributed computation of the square root and applications"**. In: *International Conference on Information Security Practice and Experience*. Springer. 2012, pp. 277–288.

[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.

[MPRV09] I. MIRONOV, O. PANDEY, O. REINGOLD, S. VADHAN. **"Computational differential privacy"**. In: *Annual International Cryptology Conference*. Springer. 2009, pp. 126–142.

[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.

[MRVW21] E. MAKRI, D. ROTARU, F. VERCAUTEREN, S. WAGH. **"Rabbit: Efficient Comparison for Secure Multi-Party Computation"**. In: *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I*. 2021, pp. 249–270.

[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.

[PS15] P. PULLONEN, S. SIIM. **"Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations"**. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 172–183.

[RBS+22] D. RATHEE, A. BHATTACHARYA, R. SHARMA, D. GUPTA, N. CHANDRAN, A. RASTOGI. **"SecFloat: Accurate Floating-Point meets Secure 2-Party Computation"**. In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 1553–1553.

[RN10] V. RASTOGI, S. NATH. **"Differentially private aggregation of distributed time-series with transformation and encryption"**. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 735–746.

[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.

[RRG+21] D. RATHEE, M. RATHEE, R. K. K. GOLI, D. GUPTA, R. SHARMA, N. CHANDRAN, A. RASTOGI. **"SiRnn: A math library for secure RNN inference"**. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1003–1020.

[RRK+20] D. RATHEE, M. RATHEE, N. KUMAR, N. CHANDRAN, D. GUPTA, A. RASTOGI, R. SHARMA. **"CrypTFlow2: Practical 2-party secure inference"**. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 325–342.

[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.

[SCR+11] E. SHI, T.-H. CHAN, E. RIEFFEL, R. CHOW, D. SONG. **"Privacy-Preserving Aggregation of Time-Series Data"**. In: vol. 2. 2011.

[SCRS17] E. SHI, T.-H. H. CHAN, E. RIEFFEL, D. SONG. **"Distributed private data analysis: Lower bounds and practical constructions"**. In: *ACM Transactions on Algorithms (TALG)* 13.4 (2017), pp. 1–38.

[Sha79] A. SHAMIR. **"How to share a secret"**. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

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

[SSSS17] R. SHOKRI, M. STRONATI, C. SONG, V. SHMATIKOV. **"Membership inference attacks against machine learning models"**. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 3–18.

[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.

[TBA+19] S. TRUEX, N. BARACALDO, A. ANWAR, T. STEINKE, H. LUDWIG, R. ZHANG, Y. ZHOU. **"A hybrid approach to privacy-preserving federated learning"**. In: *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 2019, pp. 1–11.

[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.

[WHWX16] G. WU, Y. HE, J. WU, X. XIA. **"Inherit differential privacy in distributed setting: Multiparty randomized function computation"**. In: *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE. 2016, pp. 921–928.

[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.

[YSMN21] S. YUAN, M. SHEN, I. MIRONOV, A. C. NASCIMENTO. **"Practical, label private deep learning training based on secure multiparty computation and differential privacy"**. In: *Cryptology ePrint Archive* (2021).

[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 Building Blocks

Oblivious Selection. $\langle \boldsymbol{y_i} \rangle^B \leftarrow \Pi^{ObliviousSelection}\left(\langle \boldsymbol{y_0} \rangle^B, \ldots, \left\langle \boldsymbol{y_{\ell-1}} \right\rangle^B, \langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$ outputs the bit-string $\langle \boldsymbol{y_i} \rangle^B$, where i is the index of the first non-zero bit c_i for $i \in [0, \ell-1]$. If all the bits $c_0, \ldots, c_{\ell-1}$ are zeros, bit-string $\boldsymbol{y_i}$ is set to a bit-string consisting of all zeros. $\Pi^{ObliviousSelection}$ is inspired by the works [JLL+19; MRT20]. $\Pi^{ObliviousSelection}$ uses a 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 $j \in \left[0, \log_2 \ell\right]$, the values of a 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 upper 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:

$$\left(\left\langle c_{a,b} \right\rangle, \left\langle \boldsymbol{y_{a,b}} \right\rangle^B\right) = \begin{cases} \left(\langle c_a \rangle, \langle \boldsymbol{y_a} \rangle^B\right), & \text{if } \langle c_a \rangle == 1 \\ \left(\langle c_b \rangle, \langle \boldsymbol{y_b} \rangle^B\right), & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 1 \\ (0,0) & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 0, \end{cases} \tag{A.46}$$

which is equivalent to

$$\left\langle c_{a,b} \right\rangle = \langle c_a \rangle \oplus \langle c_b \rangle \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \tag{A.47}$$

$$\begin{aligned} \left\langle \boldsymbol{y_{a,b}} \right\rangle = \quad & (\langle c_a \rangle \oplus \langle c_b \rangle) \cdot \left(\langle \boldsymbol{y_a} \rangle^B \cdot \langle c_a \rangle \oplus \langle \boldsymbol{y_b} \rangle^B \cdot \langle c_b \rangle\right) \\ & \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \cdot \langle \boldsymbol{y_a} \rangle^B \end{aligned} \tag{A.48}$$

$a \cdot \boldsymbol{b}$ denotes the $\wedge$ between a bit a and a bit-string $\boldsymbol{b}$. The nodes are evaluated from the 0th layer to the bottom root.

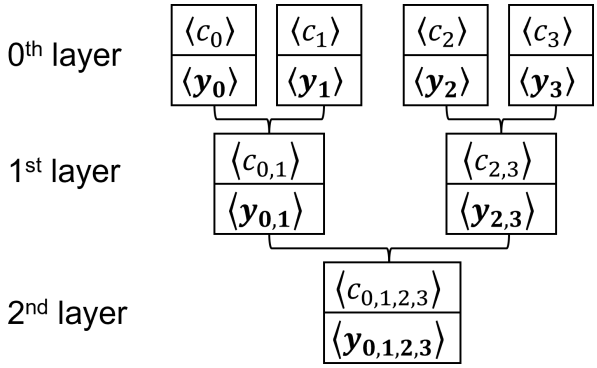

Figure A.1: Example inverted binary tree for $\Pi^{ObliviousSelection}$.

SMPC Protocol for Geometric Sampling Algorithm. We construct Prot. A.1 than generates shares of a geometric random variable $y \sim Geo(0.5)$ based on Algo. 2.2.

Protocol: $\Pi^{Geometric}$

Input: None

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

1: $(u_0, \ldots, u_{\ell-1}) \leftarrow \Pi^{RandBits}(\ell)$

2: $\left(\langle p_0 \rangle^B, \ldots, \langle p_{\ell-1} \rangle^B\right) = \Pi^{PreOr}\left(\langle u_0 \rangle^B, \ldots, \langle u_{\ell-1} \rangle^B\right)$

3: $\left(\langle b_0 \rangle^B, \ldots, \langle b_{\ell-1} \rangle^B\right) = \left(\neg\left(\langle p_0 \rangle^B\right), \ldots, \neg\left(\langle p_{\ell-1} \rangle^B\right)\right)$

4: $\langle y \rangle^{B,UINT} = \Pi^{HW}\left(\langle b_0 \rangle^B, \ldots, \langle b_{\ell-1} \rangle^B\right)$.

Protocol A.1: SMPC protocol for sampling from a geometric distribution $Geo(0.5)$.