TECHNISCHE
UNIVERSITÄT
DARMSTADT

Thesis Type

Securely Realizing Output Privacy in MPC

Student Name

July 1, 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, Student Name, die vorliegende Thesis Type ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

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

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

_____

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

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

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

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

Darmstadt, July 1, 2022

_____

Student Name

Abstract

Nowadays, the world has turned into an information-driven society where the distribution and processing of information is one important economic activity. Nonetheless, this global trend also brings a severe privacy risk because the increased amount of interconnected devices and services may reveal users' sensitive information to untrusted third-party service providers. Secure multi-party computation (MPC) was introduced in the 1980s and enabled secure computations between two or more parties, such that nothing beyond what can be inferred from the output is revealed. However, it can be possible that an adversary is able to determine if a particular data record was used in the computation of a concrete computation result. Such a so-called membership inference attack raises privacy concerns. In 2006, the concept of differential privacy (DP) was introduced, which guarantees the result of a group to be similar independent of whether an individual is in the queried database or not. One research direction for privacy protection is to combine both MPC and DP.

Although works that combine MPC and DP exist, they ignore the theoretical assumption of DP in the practical implementation. Specifically, DP assumes precise noise sampling and computation under real numbers. The major obstacle is guaranteeing DP and sample noise efficiently under MPC with finite precision.

The main goal of this thesis is to design efficient and secure protocols that combine MPC and DP to guarantee privacy under fixed/floating-point arithmetic. We convert existing secure differentially private mechanisms that require floating-point and integer sampling methods into MPC protocols.

Acknowledgments

If you like, you can add acknowledgments here.

Contents

1 Introduction

Artificial intelligence (AI) has ushered in rapid development since 2012, where the number of software projects that rely on AI has increased significantly [Cla15]. However, as an essential branch of AI, machine learning (ML) heavily relies on massive data analysis, posing severe privacy concerns as the individual's sensitive information in the highly centralized database may be misused. Therefore, it is crucial to provide privacy protections for the users' data. Since 2008, cryptographers have proposed many privacy-preserving ML (PPML) algorithms based on secure multiparty computation (MPC). MPC enables multiple parties to perform computations with parties' inputs securely such that only the computation result is revealed. The MPC paradigm was first proposed by Yao [Yao86] and became sufficiently efficient for practical deployment until the late 2000s.

Let us consider a typical scenario of PPML: Alice wishes to investigate if she has the 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. Besides, Bob wants to keep his ML model private as his intellectual property. One unrealistic solution would be to rely on a trusted third party to analyze Alice's genetic data with Bob's ML model. However, since such a trusted third party rarely exists in practice, Alice and Bob can deploy an MPC protocol to simulate a trusted third party.

Although MPC 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 resist such an attack is to deploy differentially private algorithms. The concept of differential privacy (DP) is introduced by Dwork et.al [Dwo06; DMNS06] that limits private information disclosure by adding calibrated noise to the revealed computation output.

To achieve both computational and output privacy, the natural approach is to combine both MPC and DP as already investigated by prior works [EKM+14; PL15; BP20]. However, to the best of our knowledge, none of them have considered the security issues of DP that arise in many practical implementations. Mironov [Mir12] showed that the textbook noise generation methods could break DP due to floating-point arithmetics' finite precision and rounding effects. Our works attempt to fill this gap by providing efficient and secure MPC protocols based on the state-of-the-art MPC framework MOTION [BDST22].

Research Goal In this thesis, we investigate how to combine differentially private algorithms and MPC techniques under floating-point arithmetics in a secure manner. Specifically, we

design novel MPC protocols for differentially private algorithms and secure noise generation methods based on works [Mir12; Tea20b; CKS20]. The basic idea is to generate discrete noise and re-scale it precisely under floating-point implementation such that the distribution of the noise *really* satisfy differential privacy requirements. Besides, we aim to achieving DP and maintain the utility of the computation result by adding the minimal amount of noise required to achieve DP. We also aim at efficient MPC protocols by evaluating the practical performance of various MPC optimization techniques [BDST22].

Contributions We support a variety of differentilly private mechanisms such as (discrete) Laplace mechanism [CSS12; GRS12; DR$^+$14], (discrete) Gaussian mechanism [DR$^+$14; CKS20] and snapping mechanism [Mir12] that are suitable for various applications such as web analytics, health services. We consider the outsourcing scenario [KR11], i.e., the data owners first secret share their private input to multiple ($N \geq 2$) non-colluding computation parties, and the computation parties execute the MPC protocols to securely compute the desired functionality and perturb the result. We rely on the MOTION framework [BDST22] that supports full-threshold security, which means that the computation result is secure as long as one computation party is honest. 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. This guarantees that the computation result is perturbed with minimal amount of noise required to achieve DP. To find the most efficient implementation of the arithmetic operations in MPC, we explore both fixed-point and floating-point arithmetic and implement them in the binary circuit-based and arithmetic sharing approaches. In constructing MPC protocols, we use Single Instruction Multiple Data (SIMD) instructions to eliminate the independent iterations in the sampling algorithms and improve the performance.

Thesis Outline This thesis is organized as follows: Chapter 2 give the preliminaries on the concept of secure multiparty computation and differential privacy with motivating examples and formal definitions. Chapter 3 describes the details of the differentially private mechanisms we wish to realize in MPC and our modifications. Chapter 4 provides the procedure to combine MPC protocols and differentially private mechanisms, necessary MPC building blocks, and our MPC protocols for differentially private mechanisms. Chapter 5 evaluates the performance of our MPC protocols. TODO: add after revision

2 Preliminaries

In this chapter, we first discuss the notations in § 2.1. Then, we recap the basic knowledge of secure multi-party computation in § 2.2. Next, we introduce the background knowledge and theory of differential privacy in § 2.3.

2.1 Notations

revised based on feedback

For $a, b, c \in \mathbb{N}$, let $[a]$ denote $\{x \in \mathbb{Z} \mid 1 \leq x \leq a\}$, (a, b) is $\{x \in \mathbb{R} \mid a < x < b\}$, and $[a, b]$ is $\{x \in \mathbb{R} \mid a \leq x \leq b\}$. $\{a, b, c\}$ is a set containing the three numbers, and $(a_i)_{i \in [n]} = (a_1, \ldots, a_n)$ is a sequence of n numbers. Let $\mathbb{D}$ denote the set of floating-point numbers, and $\mathbb{D} \cap (a, b)$ contains floating-point numbers in the interval (a, b). The notation $\overline{b}_{(l)}$ indicates that bit $b \in \{0, 1\}$ is repeated l times in bit-string $\overline{b}_{(l)}$. $(\cdot)_2$ is the binary representation of an integer.

Let $P_1, \ldots P_N$ denote N computation parties. The value x that is secret shared among N parties are denoted by $\langle x \rangle^S = \left(\langle x \rangle_1^S, \ldots, \langle x \rangle_N^S\right)$, where $\langle x \rangle_i^S$ is hold by party P_i. $S \in \{A, B, Y\}$ denotes the sharing type (cf. § 2.2.4): A for arithmetic sharing, B for Boolean sharing with GMW, Y for Yao sharing with BMR. B2A denotes the share conversion from B to A, and other share conversions are defined similarly as discussed in [DSZ15]. $\langle \boldsymbol{x} \rangle$ denotes a vector of ℓ shared bits. $D \in \{UINT, INT, FX, FL\}$ 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.

For the logical operations, we use XOR ($\oplus$), AND ($\wedge$), and NOT ($\neg$). Let $\langle a \rangle^D \odot \langle b \rangle^D$ be the arithmetic operations on two shared numbers of type D, where $\odot \in \{+, -, \cdot, \div, >, ==\}$ and $D \in \{UINT, INT, FX, FL\}$.

Similarly, $\ln(\langle a \rangle)$, $2^{\langle a \rangle}$, $e^{\langle a \rangle^D}$, $\left|\langle a \rangle^D\right|$, $\left\lfloor \langle a \rangle^D \right\rfloor$, $\left\lceil \langle a \rangle^D \right\rceil$, $\langle a \rangle^D \mod \langle b \rangle^D$ denote the arithmetic operations of shared numbers $\langle a \rangle^D$ and $\langle b \rangle^D$ of type D. Let $\langle a \rangle^{FL} = \Pi^{UINT2FL}\left(\langle a \rangle^{UINT}\right)$ denotes 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.

Let $\langle a \rangle^B \cdot \langle \boldsymbol{b} \rangle^B$ represent the bitwise $\wedge$ operations between $\langle a \rangle^B$ and every Boolean sharing bit $\langle b \rangle^B \in \langle \boldsymbol{b} \rangle^B$.

2.2 Secure Multi-Party Computation

Yao [Yao86] first introduced the 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 as a solution. Afterwards, Beaver, Micali and Rogaway (BMR) [BMR90] generalized Yao's garbled circuit protocol to multi-party settings. Goldreich, Micali and Wigderson (GMW) [GMW19] use Boolean sharing to realize multi-party computation. Generally, the execution of MPC protocols is divided into the offline phase (no private inputs are involved) and the online phase (private inputs are involved).

Adversary Model We consider the semi-honest (passively corrupted) [EKR17, Chapter 2] adversaries that follow the protocol specifications but try to infer additional information of other parties from the messages during the protocol execution. In contrast to the malicious (active) adversaries that can deviate from the protocol, the semi-honest model is less restricted but more efficient regarding communication and computation complexity. We focus on designing and implementing efficient MPC protocols in the semi-honest adversary model.

2.2.1 Useful Tools for Multi-Party Computation

Oblivious Transfer

Oblivious transfer (OT) was first defined by Rabin [Rab05] as a mode of transferring information, where the receiver P_R received the message from the sender P_S successfully with probability $p = 0.5$, and the sender was oblivious of whether the message was received. Later, Even et al. [EGL85] redefined OT as the 1-out-of-2 OT ($\binom{2}{1}$-OT). $\binom{2}{1}$-OT has the functionality that accepts inputs (x_0, x_1) from the sender and a choice bit c from the receiver. After computation, it outputs $\perp$ to the sender and x_c to the receiver. $\binom{2}{1}$-OT 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* technique that extends 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 MPC protocols such as correlated OT (C-OT) and random OT (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), the receiver inputs a choice bit c and receives x_c.

Multiplication Triples

Multiplication triples (MTs) are proposed by Beaver [Bea91] that can be precomputed to reduce the time cost in the online phase of MPC protocols. 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. paragraph 2.2.4), we have $c = a \wedge b$ is for Boolean sharing and $c = a \times b$ in arithmetic sharing (cf. paragraph 2.2.4). Multiplication triples can be generated using C-OT (cf. § 2.2.1) as Braun et al. [BDST22] showed. The advantage of MTs is that it can reduce the MPC protocols' online complexity by converting expensive operations (e.g., arithmetic multiplication and logical AND) to linear operations (e.g., arithmetic addition and logical XOR).

2.2.2 Beaver-Micali-Rogaway (BMR)

We first present Yao's Garbled Circuit protocol, and then, extend it to multiparty setting with BMR [BMR90] protocols. Yao [Yao86] introduced the garbled circuit protocol that enables two parties (garbler P_G and evaluator P_E) to securely evaluate any functionality represented as a Boolean circuit. We follow the steps of Yao's garbled circuit protocol described in [LP09]: circuit garbling, input encoding, and circuit evaluation.

Circuit Garbling The garbler P_G 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 0 and 1 for each wire i in C. For each gate g in the Boolean circuit C (with input wire: a and b, and output wire: c), P_G 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 gate $\tilde{g}$ based on the function table of g. For example, suppose gate g is an AND gate and has function table Tab. 2.1, P_G encrypts the keys of wire c and permutes the entries in Tab. 2.1 to generate the garbled table Tab. 2.2. Note that the symmetric encryption function Enc_k uses a secret-key k to encrypt the plaintext, and its corresponding decryption function Dec_k decrypts the ciphertext successfully only when the identical secret-key is given (otherwise it outputs an error). When all the gates in circuit C are garbled, P_G sends the garbled circuit $\widetilde{C}$ that consists of garbled tables to P_E 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.

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

Table 2.2: Garbled table of AND gate g with permuted entries.

Input Encoding P_G sends the wire keys corresponding to its input directly to P_E. To evaluate the garbled circuit $\widetilde{C}$, P_E also needs the wire keys corresponding to its input. For each of P_G's input wire i with corresponding input c, P_E and P_G run a $\binom{2}{1}$-OT, where P_G acts as a sender with inputs $\left(k_0^i, k_1^i\right)$, and P_E acts as a receiver with input c and receives k_c^i. Recall that $\binom{2}{1}$-OT (cf. § 2.2.1) guarantees that P_G learns nothing about c and P_E learns nothing about k_{1-c}^i.

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

Optimizations for Yao's Garbled Circuits

In this part, we present several prominent optimizations for Yao's garbled circuit protocol. Note that in the evaluation of Yao's garbled circuit $\widetilde{C}$, P_E needs to decrypt at most four entries to obtain the correct key of the output wire. Point and permute [BMR90] technique helps P_E to identify the entry that should be decrypted (instead of decrypting four entries) in garbled tables by adding a permutation bit to each wire key. Garbled row reduction [NPS99] reduces the number of entries in the garble table from four to three by fixing the first entry to a constant value. Free-XOR [KS08] allows the evaluation of XOR gates free of interactions (between P_E and P_G) by choosing the key pairs $\left(k_0^i, k_1^i\right)$ with fixed distance R (R is kept secret to P_E) for all wires, e.g., k_0^i is chosen at random and $k_1^i = R \oplus k_0^i$. Fixed-Key AES garbling [BHKR13] reduces the encryption and decryption workload of Yao's garbled circuit by 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 P_E. Three-halves garbling [RR21] requires less 25% communication bits than two-halves garbling at the cost of more computation.

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 first garbled by one party

and evaluated by another party. BMR protocol enables the multi-party computation by jointly garbling the circuit by all parties, and then, each party evaluates the garbled circuit locally. Ben-Efraim et al. [BLO16] proposed several optimization techniques for BMR protocol, e.g., OT-based garbling and BGW-based garbling [BGW19]F.

2.2.3 Goldreich-Micali-Wigderson (GMW)

GMW protocol [GMW19] enables multiple parties to evaluate a function represented as a Boolean circuit (or arithmetic circuit). We describe a generic case of GMW protocol where two parties P_0 and P_1 wish to securely evaluate a Boolean circuit C. Each party first secret shares its input bits with other parties using an XOR-based secret sharing scheme. For example, P_0 has input bit x and sends its share x_1 to P_1, where $x = x_0 \oplus x_1$. P_1 with input bit y follows the same secret sharing steps. The XOR gate $z = x \oplus y$ can be evaluated by each party P_i locally with $z_i = x_i \oplus y_i$ since $z = x \oplus y = (x_0 \oplus x_1) \oplus (y_0 \oplus y_1) = (x_0 \oplus y_0) \oplus (x_1 \oplus y_1)$.

We describe the steps to evaluate an AND gate $z = x \wedge y$ using OT (cf. § 2.2.1). Other techniques such as Multiplication Triple [Bea91] can also be applied for the evaluation.

Evaluation of AND Gate with OT [CHK+12; Zoh17]

P_0 acts as the sender of a $\binom{4}{1}$-OT with inputs (s_0, s_1, s_2, s_3) that is calculated with his shared bits x_0, received bits y_0, and a random bit z_0 as follows:

$$\begin{aligned} s_0 &= z_0 \oplus ((x_0 \oplus 0) \wedge (y_0 \oplus 0)) \\ s_1 &= z_0 \oplus ((x_0 \oplus 0) \wedge (y_0 \oplus 1)) \\ s_2 &= z_0 \oplus ((x_0 \oplus 1) \wedge (y_0 \oplus 0)) \\ s_3 &= z_0 \oplus ((x_0 \oplus 1) \wedge (y_0 \oplus 1)) . \end{aligned} \tag{2.1}$$

P_1 acts as the receiver with choice $(x_1 y_1)_2$ and receives $z_1 = s_{(x_1 y_1)_2} = z_0 \oplus ((x_0 \oplus x_1) \wedge (y_0 \oplus y_1))$. It is easy to verify that $z = z_0 \oplus z_1$.

2.2.4 Secret Sharing and Sharing Conversions

We introduce the secret sharing types and sharing conversions used based on work [DSZ15].

Arithmetic Sharing (A) A 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 $\langle x \rangle_i^A$. x can be reconstructed by letting each party P_i sends $\langle x \rangle_i^A$ to one party who calculates $x = \sum_{i=1}^N \langle x \rangle_i^A \mod 2^\ell$. The addition of arithmetic shares can be calculated locally and multiplication of arithmetic shares can be performed with arithmetic multiplication triple (cf. § 2.2.1).

Boolean Sharing with GMW (B) A bit x 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 sharing can be seen as a special case of arithmetic sharing where XOR and AND are used instead of arithmetic addition and multiplication.

Yao Sharing with BMR (Y) Recall that in Yao's garbled circuit protocol (cf. § 2.2.2), we introduce the optimization techniques such as point and permute [BMR90] and Free-XOR [KS08]. Specifically, the garbler generates a random κ-bit string R with $R[0] = 1$ (as permutation bit) and a random key k_0^i, and set $k_1^i = k_0^i \oplus R$ for each wire i. Based on the above techniques, the evaluator can share a bit x with the garbler by running a C-OT (cf. § 2.2.1). The garbler inputs a correlation function f_R, obtains (k_0, k_1) with $k_1 = k_0 \oplus R$, and set $\langle x\rangle_0^Y = k_0$. The evaluator inputs a choice bit x, and receives $\langle x\rangle_1^Y = k_x = k_0 \oplus xR$. To reconstruct x, the evaluator sends $\langle x\rangle_1^Y[0]$ to the garbler who computes $x = \langle x\rangle_0^Y[0] \oplus \langle x\rangle_1^Y[0]$. For the evaluation of XOR gate, each party can locally calculate $\langle z\rangle^Y = \langle x\rangle^Y \oplus \langle y\rangle^Y$ based on Free-XOR [KS08]. The AND gate $\langle z\rangle^Y = \langle x\rangle^Y \wedge \langle y\rangle^Y$ can be evaluated using the method from [BHKR13], where one party garbles the circuit with its share and another party evalutes the garbled circuit with its share. Yao's sharing can also be extended to multi-party case as the work [BDST22] showed.

Sharing Conversions

Different sharing types and corresponding MPC protocols have advantages in specific operations, e.g., arithmetic GMW (cf. § 2.2.3) allows parties to execute linear operations locally, and BMR (cf. § 2.2.2) is often more efficient for comparison. Therefore, we consider the sharing conversion methods introduced in works [DSZ15; BDST22].

2.2.5 MPC Framework - MOTION

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

1. Support for MPC 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 MPC protocols at the circuit's gate level and evaluate it asynchronously, i.e., each gate is separately evaluated once their parent gates become ready.
3. Support for Single Instruction Multiple Data (SIMD), i.e., vectors of data are processed instead of single data, that can reduce memory footprint and communication.
4. Integration of HyCC compiler [BDK+18] that can generate efficient circuits for hybrid MPC protocols with functionality described in C programming language.

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.

Continuous Probability Distribution

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

$$Uni(x \mid a, b) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$Uni(a, b)$ denotes the continuous uniform distribution with parameters a and b. We 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:*

$$Exp(x \mid \lambda) = \begin{cases} \lambda e^{\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.3}$$

$Exp(\lambda)$ denotes the exponential distribution with parameter λ. $x \sim Exp(b)$ is a exponential random variable. The cumulative distribution function of an exponential distribution is:

$$\Pr(x \mid \lambda) = \begin{cases} 1 - e^{\lambda x} & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases} \tag{2.4}$$

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

$$Lap(x \mid b) = \frac{1}{2b} e^{\left(-\frac{|x|}{b}\right)} \tag{2.5}$$

The Laplace distribution is a symmetric version of the exponential distribution and is also called the double exponential distribution because it can be thought of as an exponential distribution assigned a randomly chosen sign. We write $Lap(b)$ to denote the Laplace distribution with scale parameter b and $x \sim Lap(b)$ to denote a Laplace random variable.

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

$$\Pr(x \leq X \mid b) = \begin{cases} \frac{1}{2}e^{\frac{X}{b}} & \text{for } X \leq 0 \\ 1 - \frac{1}{2}e^{-\frac{X}{b}} & \text{for } X > 0 \end{cases} \tag{2.6}$$

Definition 2.3.4 (Gaussian Distribution)**.** *The univariate Gaussian (or standard normal) distribution with mean μ and standard deviation σ, has the following probability density function:*

$$\mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2.7}$$

$\mathcal{N}(\mu, \sigma)$ denotes the Gaussian distribution with mean μ and standard deviation σ. $x \sim \mathcal{N}(\mu, \sigma)$ is a Gaussian random variable.

Discrete Probability Distribution

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

$$Bern(x \mid p) = \begin{cases} p & \text{for } x = 1 \\ 1 - p & \text{for } x = 0 \end{cases} \tag{2.8}$$

$Bern(p)$ denotes the Bernoulli distribution with parameter p. $x \sim Bern(p)$ is a Bernoulli random variable.

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 $x \in \{0, 1, 2, \ldots, n\}$:*

$$Bino(x \mid n, p) = \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x} \tag{2.9}$$

$Bino(n, p)$ denotes the binomial distribution with parameters n and p. $x \sim Bino(n, p)$ is a binomial random variable. 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 $x \in \{0, 1, 2, \ldots\}$:*

$$Geo(x \mid p) = (1-p)^x p \tag{2.10}$$

$Geo(p)$ denotes the geometric distribution with parameter p. $x \sim Geo(p)$ is a geometric random variable. Note that the geometric distribution counts the number of failures until the first success (each trial with 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:*

$$DLap(x \mid t) = \frac{e^{\frac{1}{t}}-1}{e^{\frac{1}{t}}+1} \cdot e^{-\frac{|x|}{t}} \tag{2.11}$$

$DLap(t)$ denotes the discrete Laplace distribution with parameter t. $x \sim DLap(t)$ is a discrete Laplace random variable. $DLap(t)$ can be generated by reflecting the $Geo(p)$ 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 $x \in \mathbb{Z}$:*

$$DGau(x \mid \mu, \sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sum_{y \in \mathbb{Z}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}} \tag{2.12}$$

$DGau(\mu, \sigma)$ denotes the discrete Gaussian distribution with mean μ and standard deviation σ. $x \sim DGau(\mu, \sigma)$ is a discrete Gaussian random variable.

Probability Sampling Methods

Inverse Transform Sampling Method Inverse transform sampling method is a common method to generate random variables from certain distribution f using its inverted cumulative distribution 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:*

$$F^{-1}(u) = \inf\{x : F(x) = u, 0 < u < 1\} \tag{2.13}$$

If $U \sim Uni(0,1)$ (cf. 2.3.1), then $F^{-1}(U)$ is a distribution with cumulative function F. Also, if X has cumulative function F, then $F(X)$ is uniformly distributed in the interval $[0,1]$.

Inverse Transform-Based Laplace Sampling Method For example, we can sample a Laplace random variable $Y \sim Lap(b)$ from an exponential distribuion with cumulative function: $F(x \mid b) = 1 - e^{-\frac{x}{b}}$ as follows [Knu14, Chapter 3.4]:

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

Sampling from a Bernoulli Distribution $Algo^{Bern}(p)$ [KVH+21] samples a random variable $x \sim Bern(p)$ based on the comparison result between the generated uniform random variable $u \in (0,1)$ and parameter p.

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

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

Algorithm 2.1: Algorithm for sampling from Bernoulli distribution.

Sampling from a Geometric Distribution $Algo^{Geo}(0.5)$ [Wal74; Tea20b] generates a geometric random variable $x \sim Geo(0.5)$ by first generating a ℓ-bit random string $r \in \{0,1\}^{\ell}$ (i.e., ℓ Bernoulli trials) and counting its leading zeros (i.e., number of trials before the first success). If there is no 1 bit in r (i.e., $LeadingZeros(r) = \ell$), the sampling 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: Algorithm for sampling from geometric distribution.

Sampling from a Discrete Laplace Distribution $Algo^{DLap\_EKMPP}(t)$ [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: Algorithm for sampling from discrete Laplace distribution.

Number Representation

In this work, we rely on fixed-point and floating-point to perform the arithmetic operations.

Floating-Point Representation. Double-precision floating-point numbers occupy 64 bits (1 bit for sign, 11 bits for exponent, 52 bits for mantissa/significant) and are represented as follows:

$$(-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023}, \tag{2.14}$$

where $S \in \{0,1\}$ denotes the sign, $d_1 \dots d_{52} \in \{0,1\}^{52}$ denotes the mantissa (with implicit value of 1 at the first bit), $e_1 \dots e_{11} \in \{0,1\}^{11}$ denotes the binary representation of exponent (with 1023 as an offset).

Fixed-Point Representation. Fixed-point numbers are rational numbers represented as k-bit digits with an e-bit integer part (including sign bit s) and a f-bit fraction part as follows:

$$s \cdot \left(d_{e-2} \dots d_0 . d_{-1} \dots d_{-f}\right), \tag{2.15}$$

where $s \in \{-1, 1\}$ and $e = k - f$.

2.3.1 Traditional Techniques for Privacy Preservation

revised based on feedback Suppose a fictitious hospital has collected massive data from thousands of patients and wants to make the data available to academic researchers such as data analysts. However, the data contains sensitive information of the patients, e.g., $ZipCode$, Age, $Nationality$ and $HealthCondition$. 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 take 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 (SSN) of the patients. Tab. 2.3 shows the anonymized medical records from the fictitious hospital. The attributes are divided into two groups: the non-sensitive attributes and the sensitive attribute. The value of the sensitive attributes must be kept secret for each individual in the records. We want to guarantee that no attacker can identify the patient and discover 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].

A typical attack is the re-identification attack [Swe97] that combines the released anonymized data with publicly available information to re-identify individuals. One traditional approach against the re-identification attacks is to deploy privacy preservation methods that satisfy the notion of $k-anonymity$ [SS98] to anonymize the data and prevent the data subjects from being re-identified. More specifically, the $k-anonymity$ requires that for all individuals whose information appears in the dataset, each individual's information cannot be distinguished from at least $k-1$ other individuals.

Samarati et al. [SS98] introduced two techniques to achieve $k-anonymity$: data generalization and suppression. The former method makes the data less informative by mapping specific attribute values to a broader value range, and the latter method removes specific attribute values. As Tab. 2.4 shows, the values of attribute Age in the first eight records are replaced by value ranges such as < 30 and ≥ 40 after generalization. The values of attribute $Nationality$ 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., $ZipCode$: $130**$, 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].

$k-anonymity$ alleviates re-identification attacks but is still vulnerable to the so-called homogeneity attacks and background knowledge attacks [MKGV07]. One example for the background knowledge attack is that, suppose we know one patient who is about thirty years old, has visited the hospital and his record is in Tab. 2.4, then we could conclude that he has cancer. Afterward, $l-Diversity$ [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 every equivalent class as Tab. 2.5 shows. However, the definition of $l-Diversity$ is proved to suffer from other attacks [LLV07]. Then, in 2007, Li et al. [LLV07] introduced the concept of $t-closeness$ as an enhancement of $l-diversity$. $t-closeness$ that requires the distance (e.g., Kullback-Leibler distance [KL51] or Earth Mover's distance [RTG00]) between the distribution of the sensitive attributes in each equivalent class differs from the distribution of the sensitive attributes in the whole table to be less than the given threshold t. However, $t-closeness$ was later showed to significantly affect the quantity of valuable information the released data contains by Li et al. [LLV09].

	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, protecting privacy in such a system is nontrivial. For instance, the curator must prohibit queries targeting a specific individual, such as "Does Bob suffers from heart disease?". In addition, a single query that seems not to target individuals may still leak sensitive information when several such queries are combined. Instead of releasing the actual query result, releasing approximate statistics could prevent the above attack. However, Dinur et al. [DN03] showed that the adversary could 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. Finally, the problem turns into finding a theory that can interpret the relation between preserving privacy and providing valuable statistical information. Differential privacy [Dwo06] is a robust definition that can support quantitative analysis of how much useful statistical information should be released while preserving a desired level of privacy.

2.3.2 Differential Privacy Formalization

Randomized Response

In this part, we introduce differential privacy and start with a very early differentially private algorithm, the Randomized Response [DN03].

We adapt an example from [Kam20] to illustrate the basic idea of Randomized Response. Suppose a psychologist wishes to study the psychological impact of cheating on high school students. The psychologist first needs to find out the number of students who have cheated. Undoubtedly, most students would not admit honestly if they had cheated in exams. More precisely, suppose there are n students, and each student has a sensitive information bit $X_i \in \{0,1\}$, where 0 denotes $nevercheated$ and 1 denotes $havecheated$. Every student want to keep their sensitive information X_i secret, but they need to answer whether they have cheated. Then, each student sends the psychologist an answer Y_i which may be equal to X_i or a random bit. Finally, the psychologist collects all the answers and tries to get an accurate estimation of the fraction of cheating students $CheatFraction = \frac{1}{n}\sum_{i=1}^{n} X_i$.
The strategy of students can be expressed with following formulas:

$$Y_i = \begin{cases} X_i & \text{with probability } p \\ 1 - X_i & \text{with probability } 1-p \end{cases} \tag{2.16}$$

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

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

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

$$Var\left[\tilde{C}\right] = Var\left[\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right]\right] = \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i]. \tag{2.17}$$

Since Y_i is a Bernoulli random variable, we have $Var[Y_i] = p(1-p) \le \frac{1}{4}$ and

$$\begin{aligned}\frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i] &= \frac{1}{4\gamma^2 n} Var[Y_i] \\ &\leq \frac{1}{16\gamma^2 n}.\end{aligned} \tag{2.18}$$

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

$$\Pr(|X-\mu| \geq t) \leq \frac{\sigma^2}{t^2}, \tag{2.19}$$

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

$$\begin{aligned}\Pr\left(\left|\tilde{C} - CheatFraction\right| \geq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\leq O(1) \\ \Pr\left(\left|\tilde{C} - CheatFraction\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\geq O(1),\end{aligned} \tag{2.20}$$

and $\left|\tilde{C} - CheatFraction\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)$ with high probability. The error term $\left|\tilde{C} - CheatFraction\right| \to 0$ as $n \to \infty$ with high probability. The conclusion is that the error increases as the privacy protection level increases $\gamma \to 0$. To maintain accuracy, more data $n \to \infty$ is needed. To further quantify the privacy and accuracy, we need to define differential privacy.

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

Terms and Definitions

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

Take Tab. 2.6 as an example. The database contains the names and exam scores of five students. The database is represented by its rows. The data universe $\mathcal{X}$ contains all the combinations of student names and exam scores.

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

Table 2.6: Database example.

Data Curator. A data curator is trusted to manage and organize the database, and its primary goal is to ensure that the database can be reused reliably. In terms of differential privacy, the data curator is responsible for preserving the privacy of individuals represented in the database. The curator can also be replaced by cryptographic protocols such as secure multiparty protocols [GMW19].

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

Definition 2.3.10 (Privacy Mechanism [DR$^+$14])**.** *A privacy mechanism $M : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$ is an algorithm that takes databases, queries as input and produces an output string, where $\mathcal{Q}$ is the query space and $\mathcal{Y}$ is the output space of M.*

The query process is as Fig. 2.1 shows, a data curator manages the database and provides an interface that deploys a privacy mechanism for a data analyst/adversary to query. After the querying, the data analyst/adversary receives an output.

Figure 2.1: DP setting.

Definition 2.3.11 (Neighboring Databases [DR$^+$14])**.** *Two databases D_0, $D_1 \in \mathcal{X}^n$ are called neighboring if they differ in exact one entry. This is expressed as $D_0 \sim D_1$.*

Definition 2.3.12 (Differential Privacy [DR$^+$14])**.** *A privacy mechanism $M : \mathcal{X}^n \times \mathcal{Q} \to \mathcal{Y}$ is (ε, δ)-differential privacy if for any two neighboring databases D_1, $D_1 \in \mathcal{X}^n$, and for all $T \subseteq \mathcal{Y}$, we have $\Pr[M(D_0) \in T] \leq e^{\varepsilon} \cdot \Pr[M(D_1) \in T] + \delta$, where the randomness is over the choices made by M.*

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

Definition 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 $f : \mathcal{X}^n \to \mathbb{R}^k$ is defined as $\Delta_t^{(f)} = \max_{D_0, D_1} \|f(D_0) - f(D_1)\|_t$, where D_0, D_1 are neighboring databases and $t \in \{1, 2\}$.*

Recall the Differential Privacy Definition 2.3.12 attempts to *blur* the contribution of any individual in the database using the notion of neighboring databases. Therefore, the sensitivity is a natural quantity when considering differential privacy since it calculates the upper bound of how much f can change when modifying a single entry.

Motivating Example of Differential Privacy

The previous example about randomized response § 2.3.2 indicates that we need DP to solve the trade-off problem between learning useful statistics and preserving the individuals' privacy. In other words, the psychologist wants to find the fraction of students who have cheated in the exam while guaranteeing that no students suffer from privacy leakage by participating in the questionnaire. To illustrate how DP solves such problems, we adapt the example from [Zum15]. Consider a game as Prot. 2.1 shows,

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

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

Protocol 2.1: A motivating example of differential privacy.

Suppose the adversary A has chosen two data sets:

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

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

The Deterministic Case. Suppose the challenger wants to calculate the mean value of data sets and chooses $M(D) = mean(D)$. Since $M(D_0) = 0$ and $M(D_1) = 0.01$, the adversary can set $Q = [0.005, 1]$ and identify precisely the database D used in $M(D)$ every time they play the game. In Fig. 2.2, the blue line represents the distribution of $M(D_0)$, whereas the orange line represents the distribution of $M(D_1)$ (plotted upside down for clarity). The vertical dotted line represents the threshold $T = 0.005$ which separates D_0 and D_1 perfectly.

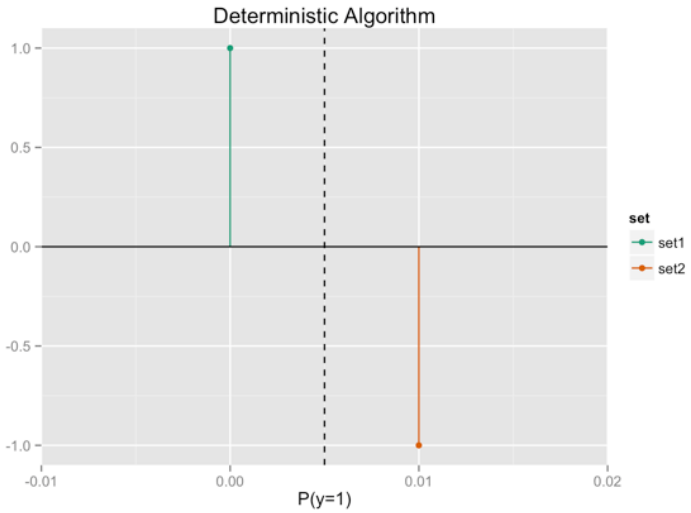

Figure 2.2: Deterministic algorithm.

The Indeterministic Case. The challenger needs to take some measures to *blur* the difference between $M(D_0)$ and $M(D_1)$. Suppose the challenger decides to add Laplace noise $lap \sim Laplace(b = 0.05)$ to the result of $M(D)$ as Fig. 2.3 shows. The shaded blue region is the chance that $M(D_0)$ would return a value greater than the adversary's threshold T. In other words, the probability that the adversary would mistake D_0 for D_1. In contrast, the shaded orange area is the probability that the adversary identify D as D_1. The challenger can decrease the adversary's probability of winning by adding more noise as Fig. 2.4 shows, where the shaded blue and orange areas are almost of the same size. Comparing $M(D)$ with T is no longer reliable to distinguish D_0 and D_1. In fact, we have $\varepsilon = \log\left(\frac{\text{blue area}}{\text{orange area}}\right)$, where ε expresses the degree of differential privacy and a smaller ε guarantees a stronger privacy protection. Although the challenger can add more noise to decrease the adversary's success probability, the mean estimation accuracy is also decreased.

TODO: need reproduce following figures

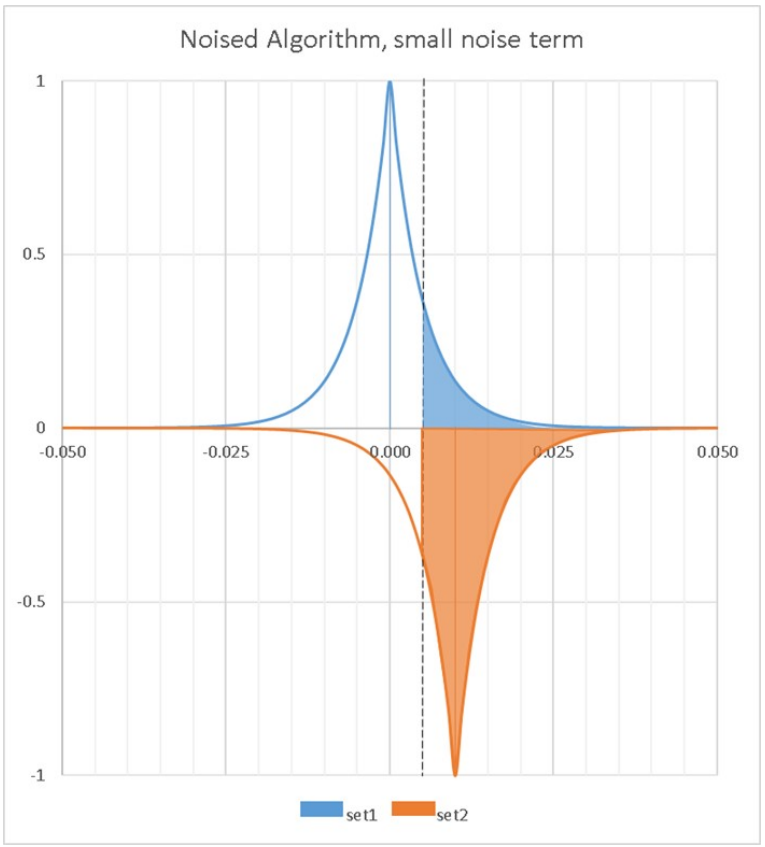

Figure 2.3: Indeterministic algorithm with small noise ($b = 0.005$).

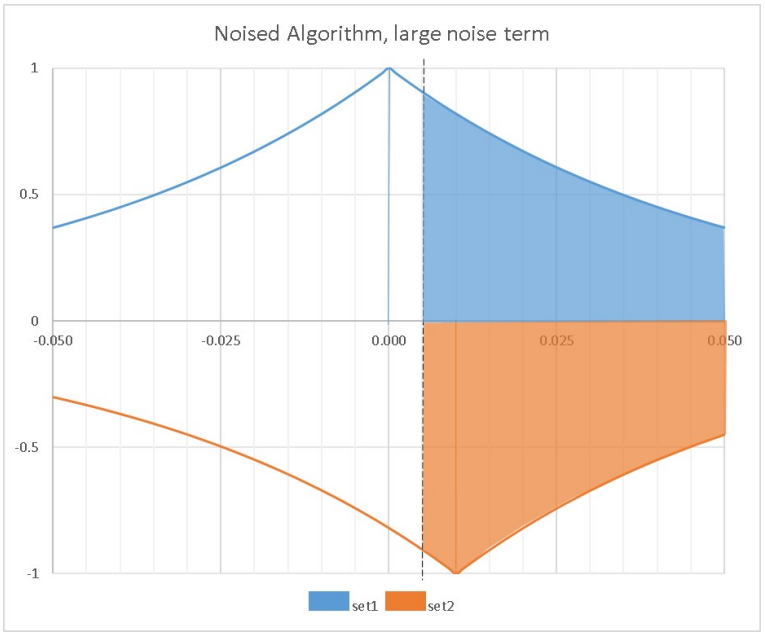

Figure 2.4: Indeterministic algorithm with large noise ($b = 0.05$).

Properties of Differential Privacy

Post-Processing

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

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

Group Privacy

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

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

Basic Composition

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

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

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.

ε-Differential Privacy

Definition 2.3.16 (Laplace Mechanism [DR$^+$14])**.** *Let $f : \mathcal{X}^n \to \mathbb{R}^k$. The Laplace mechanism is defined as $M_{Lap}(X) = f(X) + (Y_1, \dots, Y_k)$, 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 5. *Laplace Mechanism preservers ε-DP [DR$^+$14].*

(ε, δ)-Differential Privacy

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

Definition 2.3.17 (Gaussian Mechanism [DR$^+$14])**.** *Let $f : \mathcal{X}^n \to \mathbb{R}^k$. The Gaussian mechanism is defined as $M(X) = f(X) + (Y_1, \dots, Y_k)$, where the Y_i are independent Gaussian random variables drawn from distribution $\mathcal{N}\left(Y_i | \mu, \sigma^2\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{Y_i - \mu}{\sigma}\right)^2}$ with $\mu = 0$, $\sigma^2 = 2\ln\left(\frac{1.25}{\delta} \cdot \left(\frac{\Delta_2^{(f)})^2}{\varepsilon^2}\right)\right)$.*

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

Discussion about Differential Privacy

Local and Central Differential Privacy Differential privacy is a definition that can be realized in many ways. Two common modes of DP are centralized differential privacy [DR+14] and local differential privacy [DN03].

In centralized DP, all data is stored centrally and managed by a trusted curator before the differentially private mechanism is applied. As Fig. 2.5 shows, the raw data from clients is first collected in a centralized database, then, the curator applies the privacy mechanism and answers the queries $f(x)$ with $f'(x)$. The local DP mode is, as Fig. 2.6 shows, where the clients first apply a privacy mechanism on their data, and send the perturbed data to the curator. An advantage of local DP mode is that no trusted central curator is needed since the data is perturbed independently before sending to the curator. However, the disadvantage is that the collected data contains redundant noise and may decrease the utility.

TODO: reproduce following figures

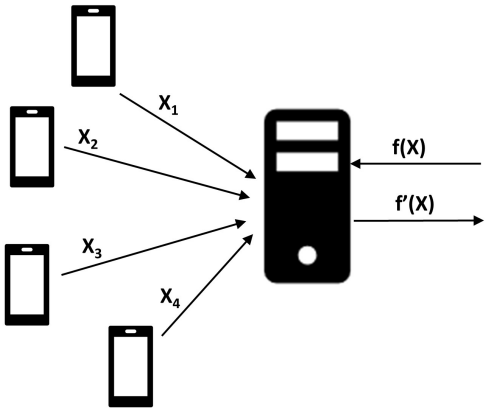

Figure 2.5: Centralized DP mode.

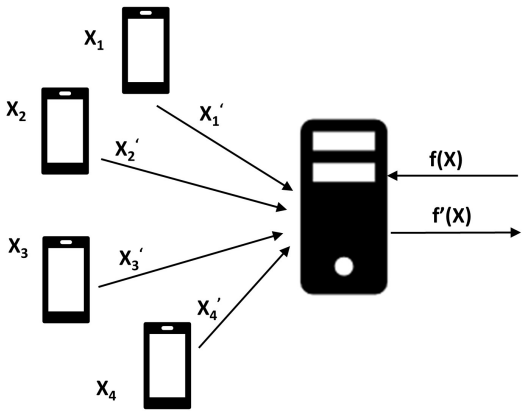

Figure 2.6: Local DP mode.

Advantages of Differential Privacy From the example § 2.3.2, we found that DP can still protect privacy even if the adversary has the knowledge of the database. Generally speaking, DP ensures privacy protection by making no assumption about the adversary's auxiliary information (even when the adversary is the data provider) or computational strategy (regarding the complexity of modern cryptography) [Vad17]. In addition, DP provides a quantitive theory about safely releasing data and maintaining certain level of accuracy.

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

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

Implementation of DP Mechanisms. The theory of DP is built upon the real number arithmetic. The practical implementation of differentially private mechanisms relies on floating-point or fixed-point arithmetic only provides an approximation of the mathematical abstractions. Mironov [Mir12] showed that the irregularities of floating-point implementations and porouse distribution of the Laplace mechanism with textbook sampling algorithm lead to the breach of differential privacy. Further, Gazeau et al. [GMP16] proved that any

differentially private mechanism that perturbs data by adding noise with a finite precision can resulting secret disclosure regardless of the actual implementation.

3 Secure Differentially Private Mechanisms On Finite Computer

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 based on floating-point or fixed-point arithmetic that only provides finite order of accuracy. Mironov [Mir12] showed that the porous distribution of the Laplace random noise sampled with textbook algorithm under floating-point implementaion couild lead to violation of differential privacy. In this chapter, we describe four types of existing differentially private mechanisms [Mir12; Tea20b; GRS12; CKS20] and modify corresponding sampling algorithms for generating secure noise on the finite computer. We construct MPC protocols for these differentially private mechanisms and sampling algorithms in § 4.

3.1 Snapping Mechanism

Recall that Laplace mechanism (cf. § 2.3.2) guarantees ε-DP by adding Laplace random variable $Y \sim Lap(\lambda)$ to the query function $f(D)$ with database D:

$$M(D, \lambda) = f(D) + Y. \tag{3.21}$$

As discussed in paragraph 2.3, we can generated a Laplace random variable Y by transforming the 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 (small) probability of generating exact 1) as follows:

$$Y \leftarrow S \cdot \lambda \ln(U) \tag{3.22}$$

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

The snapping mechanism is defined as follows:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)\right\rceil_\Lambda\right). \tag{3.23}$$

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 the query function of database D, and $S \otimes \lambda \otimes \text{LN}(U^*)$ is the noise term. S is the sign of the noise that is uniformly distributed over $\{-1, 1\}$. U^* is a *uniform* distribution over $\mathbb{D} \cap (0, 1)$, and generates floating-point numbers with probability proportional to its *unit in the last palce* (ulp), i.e., spacing between two consecutive floating-point numbers. LN(x) is the natural logarithm under floating-point implementation with exact rounding, i.e., 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. 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 input x exactly to the nearest multiple of Λ by manipulating the binary representation of floating-point number x.

Note that 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 6 ([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 snapping mechanism consists of the following steps:

1. $\text{clamp}_B(\cdot)$.
2. Generation of U^* and S.
3. Floating-point arithmetic operations such as $\text{LN}(\cdot)$, $\oplus$, and $\otimes$.
4. Operation $\lfloor x \rceil_\Lambda$ that rounds input x to the nearest multiple of Λ.

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

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

$$U^* = (1.d_1 \ldots d_{52})_2 \times 2^{e-1023}. \tag{3.24}$$

As discussed above, each floating-point U^* should be output with a probability proportional to its ulp. We sample a floating-point number from U^* using $Algo^{RandFloat1}$ [Wal74; Mir12], i.e., independently sampling a geometric random variable $x \sim Geo(0.5)$, and the significant bits $(d_1, \ldots, d_{52}) \in \{0,1\}^{52}$. Then, we set 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 : $x \leftarrow Algo^{Geo}(0.5)$

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

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

Algorithm 3.1: Algorithm for sampling uniform random floating-point number $U^* \in \mathbb{D} \cap (a,b)$.

As 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 U^*. Further, $x \sim Geo(0.5)$ guarantees that the probability of sampling a floating-point number from U^* is proportional to its ulp. Intuitively, *uniformly* sampling a floating-point number can be thought of 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} representable reals in the interval $[.5,1)$ and 2^{52} reals in the interval $[.25,.5)$. If we only sample the floating-point numbers with equal distance to each other in the interval $(0,1)$, a large amount of floating-point numbers would be ignored. As discussed in [Wal74; Mir12], a better approach is to sample floating-point numbers with probability proportional to its ulp (i.e., spacing to its consecutive neighbor). With $x \sim Geo(0.5)$, 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, we have a total sampling probability for the floating-point numbers in in the interval $(0,1)$ is $\sum_{i=1}^{\infty} \frac{1}{2^i} \approx 1$.

3.2 Integer-Scaling Mechanism

Google Differential Privacy Team [Tea20b] proposed a differentially private algorithm that could achieve a similar DP protection effect as the Laplace mechanism (cf. 2.3.16) or Gaussian mechanism (cf. 2.3.17) by re-scaling a discrete random variable to simulate the continuous random variable, which is defined as follows:

$$M_{IS}(f(D), r, \varepsilon, \delta) = f_r(D) + ir. \tag{3.25}$$

where 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 $f_r(D) \in \mathbb{D}$ rounds the output of query function $f(D) \in \mathbb{R}$ to the nearest multiple of r. 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 differential privacy guarantee.

Main Idea. Let us assume $f_r(D) + ir$ satisfy (ε, δ)-DP under real number arithmetic, if $f_r(D) + ir$ can be computed *precisely* under floating-point arithmetic, then we can conclude that $f_r(D) + ir$ also satisfies (ε, δ)-DP under floating-point arithmetic. *Precisely* indicates that the real numbers are represented as floating numbers without precision loss, and the arithmetic operations of these real numbers yield the exactly same result as when these real numbers are represented as floating-point numbers. If integer $|i| \leq 2^{52}$ (or $\Pr\left(\left|i > 2^{52}\right|\right)$ is small enough to be ignored), then we have $|ir| \leq 2^{1022}$. Therefore, ir can be represented precisely as a floating-point number by adding k to the exponent of i. Further, by choosing r and limiting $|f(D)| < 2^{52}r$, we can guarantee that $f_r(D)$ and $f_r(D) + ir$ can be represented precisely as a floating-point number. Finally, it proved that $f_r(D) + ir$ satisfies (ε, δ)-DP under floating-point implementation.

3.2.1 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:

$$M_{ISLap}(f(D), r, \Delta_r, \varepsilon) = f_r(D) + ir, \tag{3.26}$$

where r is the resolution parameter that controls the scale of the simulated continuous Laplace random variable ir, integer $i \sim DLap\left(t = \frac{\Delta_r}{r\varepsilon}\right)$ (cf. 2.3.8), $\Delta_r = r + \Delta_1^{(f)}$, and $\Delta_1^{(f)}$ is the ℓ_1-sensitivity (cf. 2.3.15) of $f(D)$.

Theorem 7 ([Tea20b]). *The Integer-Scaling Laplace mechanism $M_{ISLap}(f(D), r, \Delta_r, \varepsilon)$ satisfies ε-DP for query function f.*

We found three discrete Laplace sampling algorithms [EKM+14; Tea20b; CKS20] that can be used in the Integer-Scaling Laplace mechanism. We only describe and modify the discrete Laplace sampling algorithm in work [Tea20b], that first generates a geometric random variable x with a binary search-based geometric sampling algorithm and converts x to a discrete Laplace random variable i.

Binary Search Based Geometric Sampling Algorithm $Algo^{GeoExpBinarySearch}$ 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 $Geo\left(x \mid p = 1-e^{-\lambda}\right) = (1-p)^{x-1} \cdot p$.

Sampling Interval of x. Let us first define the sampling interval of geometric random variable x and λ. We sample random variable x in the interval $\left[1, 2^{52}\right]$. In original work [Tea20a], the sampling interval for x is $\left[1, 2^{63}-1\right]$. We limit the sampling interval of x to $\left[1, 2^{52}\right]$ because each integer in $\left[1, 2^{52}\right]$ can be represented exactly as a floating-point number.

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

$$\text{CDF} : \Pr\left(x \leq 2^{52}\right) = 1 - e^{-\lambda \cdot 2^{52}} = \begin{cases} 0.\overline{9}_{(27)}8396\ldots & \text{for } \lambda = 2^{-48} \\ 0.\overline{9}_{(13)}8734\ldots & \text{for } \lambda = 2^{-49} \end{cases} \tag{3.27}$$

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

Algorithm Description. $Algo^{GeoExpBinarySearch}$ samples a geometric random variable x by splitting the sampling interval into two subintervals with the almost equal cumulative probability. Then, one subinterval is chosen at random, and the other is discarded. This sampling interval splitting and choosing process is repeated until the remaining sampling interval only contains one value: the geometric random variable x we are looking for. In original work [Tea20b], the sampling algorithm additional checks if the generated random variable exceed the sampling interval $\left[0, 2^{63}-1\right]$, that is not necessary in our case because we set $\lambda > 2^{-48}$ such that it happens with 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 $(L \ldots R]$ is first 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., $Pr\left(Geo\left(L < x \leq M\right)\right) \approx Pr\left(Geo\left(M \leq x < R\right)\right)$). Lines $4-7$ ensure that the middle point M lies in the interval $(L \ldots R]$ (or relocates M to interval $(L \ldots R]$ otherwise). Line 8 calculates the cumulative probability proportion Q between $(L \ldots M]$ and $(M \ldots R]$ 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 the generated uniform variable U (cf. Algorithm 3.1) and Q. If $U \leq Q$, we choose $(L \ldots M]$ as the next sampling interval, $(M \ldots R]$ 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 2is not satisfied). Finally, we set $x \leftarrow R$, 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 : $\quad U \leftarrow Algo^{RandFloat1}$
10 : $\quad$ **IF** $U \leq Q$
11 : $\quad\quad R \leftarrow M$
12 : $\quad$ **ELSE**
13 : $\quad\quad L \leftarrow M$
14 : **RETURN** $x \leftarrow R$

Algorithm 3.2: Algorithm for sampling geometric random variable $Geo\left(p = 1 - e^{-\lambda}\right)$.

Two-Side Geometric Sampling Algorithm In this part, we describe how to transform $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ to $i \sim DLap\left(t = \frac{1}{\lambda}\right)$ based on the work [Tea20b].

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). $Algo^{TwoSideGeo}$ generates a discrete Laplace random variable $i = s \cdot g$ with this method. We replace the $WHILE$ loop (line 1) in the original work with a FOR loop because the MPC computation needs to determine the number of FOR loop iterations ahead of time. In line 4, we discard the case $s \cdot g = -0$. Otherwise, 0 would be returned with twice the probability as in the discrete Laplace distribution. In line 5, we output the correct discrete Laplace random variable i. In line 6, the algorithm fails to a generate discrete random variable in $ITER$ loops and output 0.

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

Input: t

Output: $i \sim DLap(t)$

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

  $g \leftarrow Algo^{GeoExpBinarySearch}\left(\lambda = \frac{1}{t}\right)$

  $s \leftarrow 2 \cdot Bern(0.5) - 1$

  IF $\neg(s == -1 \wedge g == 0)$

    RETURN $i \leftarrow s \cdot g$ // success

RETURN $i \leftarrow 0$ // failure

Algorithm 3.3: Algorithm for sampling discrete Laplace random variable $i \sim DLap(t)$.

Algorithm Fail Probability Estimation. Suppose A_i is an event that $Algo^{TwoSideGeo}$ fails when $ITER = i$. Since each iteration is independent, we estimate the $Pr(A_{ITER})$ as follows:

$$\begin{aligned} Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}(Pr(A_1)) \\ &= \Pi_{i=1}^{ITER}(Pr(s == -1) \cdot Pr(g == 0)) \\ &= \Pi_{i=1}^{ITER}\left(Pr(0 \leftarrow Bern(0.5)) \cdot Pr\left(0 \leftarrow Geo\left(\frac{1}{t}\right)\right)\right) \\ &= \Pi_{i=1}^{ITER}\frac{1}{2}\left(1 - e^{-\lambda}\right) \\ &= \frac{1}{2^{ITER}}\left(1 - e^{-\lambda}\right)^{ITER}. \end{aligned} \tag{3.28}$$

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

3.2.2 Integer-Scaling Gaussian Mechanism

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

$$M_{ISGauss}(f(D), r, \Delta_r, \varepsilon, \delta) = f_r(D) + ir \tag{3.29}$$

Specifically, $M_{ISGauss}(f(D), r, \Delta_r, \varepsilon, \delta)$ uses a symmetrical binomial random variable $i \sim SymmBino(n, p = 0.5)$ (cf. 2.3.6) to simulate the continuous Gaussian random variable $i_{Gau} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{2\ln\left(1.25/\sigma \cdot \left(\Delta_1^f\right)^2\right)}{\varepsilon^2}\right)$. The closeness between the symmetrical and Guassian distribution depends on n, i.e., a larger n indicates a better approximation effect. n can be estimate with ε and δ [BW18] and $\sqrt{n}$ is guaranteed to be in the interval $\left[2^{56}, 2^{57}\right]$. Note that in the following part, we use the square root of n, $\sqrt{n}$, as n is too large to be represented as a 64-bit floating-point number.

Theorem 8 ([Tea20b]). *The Integer-Scaling Gaussian mechanism* $M_{ISGauss}(f(D), r, \Delta_r, \varepsilon, \delta)$ *satisfies* (ε, δ)*-DP for query function* f.

Symmetrical Binomial Sampling Algorithm $Algo^{SymmetricBinomial}$ is a modification of the symmetrical binomial sampling algorithm [Tea20b], that samples $i \sim SymmBino(n, p = 0.5)$ with input $\sqrt{n} \approx 2^{56}$. 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. We also found that when $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2}$, $\tilde{p}(x) > 0$ for $x \in [2^{56}, 2^{57}]$. Therefore, we replace the IF condition from $\tilde{p}(x) > 0$ (in orignal work) to $\left(-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2} \wedge c == 1\right)$ (line 14) that simplifies the construction of MPC protocols.

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

Input: $\sqrt{n}$

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

1: $m \leftarrow \left\lfloor \sqrt{2} * \sqrt{n} + 1 \right\rfloor$
2: **FOR** $j \leftarrow 1$ **TO** $ITER$
3: $\quad s \leftarrow Algo^{Geo(0.5)}$
4: $\quad b \leftarrow RandBit(1)$
5: $\quad$ **IF** $U^* < 0.5$
6: $\quad\quad k \leftarrow s$
7: $\quad$ **ELSE**
8: $\quad\quad k \leftarrow -s - 1$
9: $\quad l \leftarrow Algo^{RandInt}(m)$
10: $\quad x \leftarrow km + l$
11: $\quad \tilde{p}(x) = \sqrt{\frac{2}{\pi n}} \cdot e^{-\frac{2x^2}{n}} \cdot \left(1 - \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}\right)$
12: $\quad f \leftarrow \frac{4}{m \cdot 2^s}$
13: $\quad c \leftarrow Algo^{Bern}\left(\frac{\tilde{p}(x)}{f}\right)$
14: $\quad$ **IF** $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2} \wedge c == 1$
15: $\quad\quad$ **RETURN** $i \leftarrow x$ // success
16: $\quad$ **ELSE**
17: $\quad\quad$ **RETURN** $i \leftarrow 0$ // failure

Algorithm 3.4: Algorithm for sampling symmetric binomial random variable $i \sim SymmBino\left(\sqrt{n}, p = 0.5\right)$.

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

$$\begin{aligned} Pr(A_{ITER}) &= \Pi_{i=1}^{ITER}(Pr(A_1)) \\ &= \left(\frac{15}{16}\right)^{ITER} \end{aligned} \tag{3.30}$$

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

3.3 Discrete Laplace Mechanism

In this section, we describe the discrete Laplace mechanism [CSS12; GRS12; EKM+14; CKS20] and present the modified sampling algorithm $Algo^{GeoExp}$.

The discrete Laplace mechanism is define as:

$$M_{DLap}(f(D), r, \varepsilon) = f(D) + Y, \tag{3.31}$$

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

Theorem 9 ([CSS12; GRS12; EKM+14; CKS20]). *The discrete Laplace mechanism $M_{DLap}(f(D), r, \varepsilon)$ satisfies ε-DP for query function $f(D) \in \mathbb{Z}$.*

Except the previous introduced discrete Laplace sampling algorithm $Algo^{DLap\_EKMPP}$ (cf. Algorithm 2.3), Canonne et al. [CKS20] proposed a discrete Laplace sampling algorithm that is based on rejection sampling [CRW04] method. 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 $Algo^{GeoExp}$ is a modificaiton of the geometric sampling algorithm from the work [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 $WHILE$ loop in the original work with two FOR loops (line $4-8$,$9-14$) and add the checking condition (line 1, 15) in case the algorithm failure. $Algo^{GeoExp}$ consists of two FOR loops and check if these FOR loops terminate in $ITER_1$ and $ITER_2$ iterations. If they both terminate, the sampling algorithm succeeds, and fails otherwise. $Algo^{RandInt}(d)$ generate a random integer u in the interval $\in [0, d-1]$. One special case is when $d = 1$, the 1th FOR loop directly terminates with $ITER_1 = 0$.

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 : $u \leftarrow Algo^{RandInt}(d)$
6 : $b_1 \leftarrow Algo^{Bern}\left(e^{-\frac{u}{d}}\right)$
7 : **IF** $b_1 == 1$
8 : $k \leftarrow 0$, **BREAK** // 1th loop terminates
9 : **FOR** $j \leftarrow 1$ **TO** $ITER_2$
10 : $b_2 \leftarrow Algo^{Bern}\left(e^{-1}\right)$
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 3.5: Algorithm for sampling geometric random variable $x \sim Geo\left(p = 1 - e^{\frac{n}{d}}\right)$ [CKS20].

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

$$\begin{aligned} Pr(A_1) &= \sum_{i=0}^{d-1} Pr(u=i) \cdot Pr(b_1=0) \\ &= \sum_{i=0}^{d-1} \frac{1}{d} \cdot \left(1 - e^{-\frac{i}{d}}\right) \\ &= 1 - \frac{1}{d} \sum_{i=0}^{d-1} e^{-\frac{i}{d}} \\ &= 1 - \frac{1}{d} \frac{1-e^{-1}}{1-e^{-\frac{1}{d}}} \end{aligned} \tag{3.32}$$

$$\begin{aligned} Pr\left(A_{ITER_1}\right) &= \prod_{i=1}^{ITER_2} Pr(A_1) \\ &= \left(1 - \frac{1}{d} \frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1} \end{aligned} \tag{3.33}$$

$$\begin{aligned} Pr\left(B_{ITER_2}\right) &= \prod_{i=1}^{ITER_2} Pr(B_1) \\ &= \prod_{i=1}^{ITER_2} Pr(b_2=1) \\ &= e^{-ITER_2} \end{aligned} \tag{3.34}$$

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

$$\begin{aligned} Pr\left(A_{ITER_1} \vee B_{ITER_2}\right) &= Pr\left(A_{ITER_1}\right) + Pr\left(B_{ITER_2}\right) - Pr\left(A_{ITER_1}\right) \cdot Pr\left(B_{ITER_2}\right) \\ &= \left(1 - \frac{1}{d} \frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1} + e^{-ITER_2} - \left(1 - \frac{1}{d} \frac{1-e^{-1}}{1-e^{-\frac{1}{d}}}\right)^{ITER_1} \cdot e^{-ITER_2} \end{aligned} \tag{3.35}$$

Finally, to guarantee $Pr\left(A_{ITER_1} \vee B_{ITER_2}\right) < 2^{-40}$: (i) when $n = 3$ and $d = 2$, we have $ITER_1 = 27$, $ITER_2 = 28$, (ii) when $n = 3$ and $d = 1$, we have $ITER_1 = 0$, $ITER_2 = 28$.

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

Algorithm Description. We replace the $WHILE$ loop in the original work with the FOR loops (line 1).

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

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

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

1 : **FOR** $j=1$ **TO** $ITER$
2 : $\quad s \leftarrow Algo^{Bern}(0.5)$
3 : $\quad m \leftarrow Algo^{GeoExp}(n,d)$
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 3.6: Algorithm for sampling discrete Laplace random variable $Y \sim DLap\left(\frac{n}{d}\right)$.

Algorithm Fail Probability Estimation. Note that we need to guarantee not only $Algo^{DLap}\left(t=\frac{d}{n}\right)$, but that all subprotocols (e.g. $Algo^{GeoExp}(n,d)$) fail with probability less than 2^{40}. Suppose A_i is an event that $Algo^{DLap}$ fails in $ITER_1 = i$ iterations. As $Pr(s==1)$ and $Pr(m==1)$ are independent of each other, we have:

$$\begin{aligned} Pr(A_1) &= Pr(s==1)\cdot Pr\left(m==0 \wedge Algo^{GeoExp(n,d)} \text{ successes}\right) + Pr\left(m==0 \wedge Algo^{GeoExp(n,d)} \text{ fails}\right) \\ &= \frac{1}{2}\cdot Pr\left(m==0 \wedge Algo^{GeoExp(n,d)} \text{ successes}\right) + Pr\left(Algo^{GeoExp(n,d)} \text{ fails}\right) \\ &= \frac{1}{2}\cdot\left(1-e^{-\frac{n}{d}}\right) + Pr\left(Algo^{GeoExp(n,d)} \text{ fails}\right) \end{aligned} \tag{3.36}$$

Finally, we have

$$\begin{aligned} Pr\left(Algo^{DLap\left(t=\frac{d}{n}\right)} \text{ fails}\right) &= Pr(A_{ITER}) \\ &= \prod_{i=1}^{ITER} Pr(A_1) \\ &= \left(\frac{1}{2}\cdot\left(1-e^{-\frac{n}{d}}\right) + Pr\left(Algo^{GeoExp(n,d)} \text{ fails}\right)\right)^{ITER} \end{aligned} \tag{3.37}$$

When $n=3$, $d=2$, we have $ITER=30$, $Pr\left(Algo^{DLap(d,n)} \text{ fails}\right) < 2^{-40}$ and $Pr\left(Algo^{GeoExp(n,d)} \text{ fails}\right) < 2^{-40}$.

3.4 Discrete Gaussian Mechanism

In this section, we describe the discrete Gaussian mechanism [CKS20] and the modified sampling algorithm Algorithm 3.7. The discrete Laplace mechanism is defined as:

$$M_{DGauss}(D) = f(D) + Y, \tag{3.38}$$

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

Theorem 10 ([CKS20]). *The discrete Gaussian mechanism $M_{DGauss}(D) = f(D) + Y$ satisfies (ε, δ)-DP for*

$$\delta = Pr\left(Y > \frac{\varepsilon\sigma^2}{\Delta_1^f} - \frac{\Delta_1^f}{2} \,|\, Y \leftarrow DGau(\mu = 0, \sigma)\right) - e^{\varepsilon} \cdot Pr\left(Y > \frac{\varepsilon\sigma^2}{\Delta_1^f} + \frac{\Delta_1^f}{2} \,|\, Y \leftarrow DGau(\mu = 0, \sigma)\right), \tag{3.39}$$

where query function $f(D) \in \mathbb{Z}$ and $\Delta_1^{(f)}$ is the sensitivity of $f(D)$.

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 $Algo^{DGau}$ 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).

Algorithm Description. We replace the $WHILE$ loop in the original work with FOR loop (line 2). We can replace $Algo^{DLap}(t)$ with other discrete Laplace sampling algorithms.

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

Input: σ

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

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

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

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

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

  IF $B == 1$

    RETURN $Y \leftarrow L$ // success

RETURN $Y \leftarrow 0$ // failure

Algorithm 3.7: Algorithm for sampling discrete Gaussian random variable $Y \sim DGau(\mu = 0, \sigma)$.

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

$$\begin{aligned} Pr(A_1) &= \sum_{i=0}^{\infty} Pr\left(B = 0 \wedge L = i \wedge Algo^{DLap}(t) \text{ successes}\right) + Pr\left(Algo^{DLap}(t) \text{ fails}\right) \\ &= \sum_{i=0}^{\infty} \left(Pr(B = 0) \cdot Pr(L = i) \cdot Pr\left(Algo^{DLap}(t) \text{ successes}\right)\right) + Pr\left(Algo^{DLap}(t) \text{ fails}\right) \\ &= \sum_{i=0}^{\infty} \left(1 - e^{-\frac{\left(|i| - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}}\right) \cdot \frac{\left(e^{\frac{1}{t}} - 1\right) \cdot e^{-\frac{|i|}{d}}}{e^{\frac{1}{t}} + 1} \cdot Pr\left(Algo^{DLap} \text{ successes}\right) + Pr\left(Algo^{DLap} \text{ fails}\right) \end{aligned} \tag{3.40}$$

For $\sigma = 1.5$, we have $ITER = 23$, $Pr\left(Algo^{DGau(t)} \text{ fails}\right) < 2^{-40}$, and $Pr(sub - protocols \text{ fails}) < 2^{-40}$.

4 MPC Protocols for Differentially Private Mechanisms

In this chapter, we present our MPC-DP procedure that combines MPC protocols and differentially private mechanisms. We first give a general overview of the settings and a description of our MPC-DP procedure. Then, we construct the MPC protocols of the previously introduced differentially private mechanisms and sampling algorithms (cf. § 3). Generally, we face two technical challenges. The first is to identify the most efficient MPC protocols for arithmetic operations. The second is determining the most efficient sampling algorithms for differentially private mechanisms. For the first challenge, we implement MPC protocols that support (signed) integer arithmetic, fixed-point arithmetic, floating-point arithmetic, and data type conversions in the MOTION frameowrk [BDST22]. For the second challenge, we construct the MPC protocols for all the introduced sampling algorithms and evaluate the performance in § 5.

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

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

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

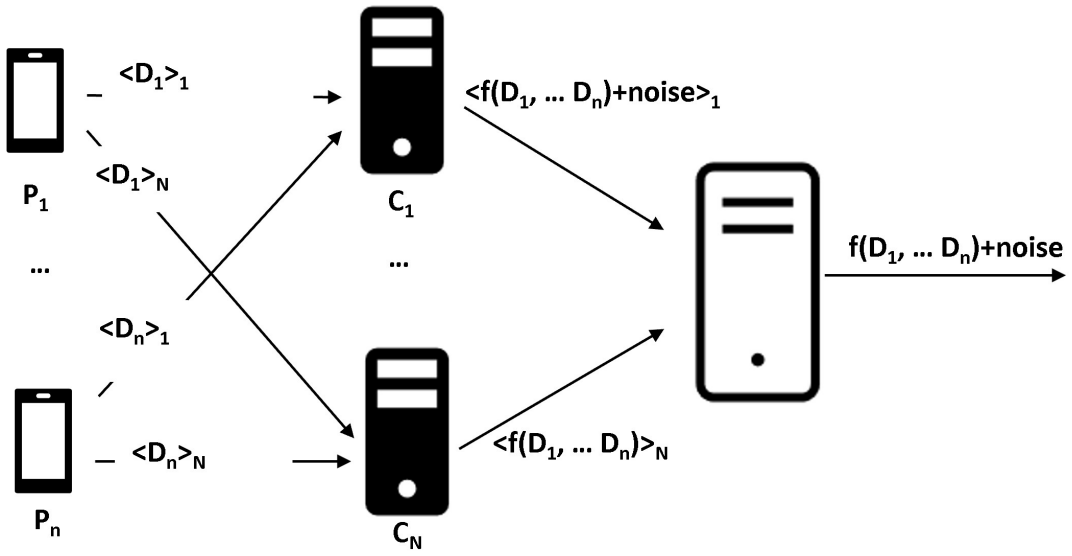

Figure 4.1: MPC-DP Procedure

MPC-DP Procedure Overview. MPC-DP procedure consists of three steps as Fig. 4.1 shows: (i) The users send the secret shared data $\langle D_i \rangle$ to the computation parties. (ii) The computation parties execute the MPC protocols to compute $\langle f(D_1, \ldots, D_n) \rangle$ (we assume this step is finished), generate secret shared noise $\langle Y \rangle$, perturb the result with differentially privacy mechanisms. (iii) The computation parties send their share of the perturbed result to the reconstruction party, that reconstructs the perturbed result.

4.1 Building Blocks

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

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

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

5. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{HW}(x_0, \ldots, x_\ell)$ [BP08] computes the Hamming weight (i.e., the number of 1 bits) in a ℓ-bit string $x_0, \ldots, x_\ell$.
6. $\langle \boldsymbol{y_i} \rangle^B \leftarrow \Pi^{ObliviousSelection}\left(\langle \boldsymbol{y_0} \rangle^B, \ldots, \langle \boldsymbol{y_{\ell-1}} \rangle^B, \langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$ outputs bit-string $\boldsymbol{y_i}$ where i is the first index such that $c_i == 1$ for $i \in [0, \ell-1]$.
7. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{RandInt}(m)$ generate shares of a random integer y in the interval $[0, m-1]$.
8. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{COTMult}\left(\langle \boldsymbol{x} \rangle^B, \langle b \rangle^B\right)$ [AHLR18; ST19] computes the multiplication of a bit string $\langle \boldsymbol{x} \rangle^B$ and a single bit $\langle b \rangle^B$ with correlated-OT (cf. § 2.2.1).
9. $\langle \boldsymbol{y} \rangle^B \leftarrow \Pi^{Mux}\left(\langle a \rangle^B, \langle \boldsymbol{x_0} \rangle^B, \langle \boldsymbol{x_1} \rangle^B\right)$ outputs bit-string $\langle \boldsymbol{x_0} \rangle^B$ if $a = 1$, $\langle \boldsymbol{x_0} \rangle^B$ otherwise.

4.2 Number Representations and Arithmetic Operations

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

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

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

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

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

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

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

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

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

Random Number Generation As discussed in (cf. § 3), the differentially private mechanisms rely heavily on generating of random numbers. Generally, four types of random numbers are used in our protocols: (i) random BGMW bits, random unsigned integers in a specific range, random fixed/floating-point numbers in the range $(0,1)$. To generate a random Boolean bit-string of length ℓ, each computation party P_i locally generate a random ℓ-bit string $\boldsymbol{r_i^B}$ and set $\boldsymbol{r_i^B}$ as a Boolean sharing with GMW protocol of a publicly unknown random bit string $\boldsymbol{r_0^B} \oplus \ldots \oplus \boldsymbol{r_{N-1}^B}$. Let $\Pi^{RandBits}(\ell)$ denote the MPC protocol for generating random bits of length ℓ.

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

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

To generate a uniform random floating point in range $(0,1)$, we follow the methods discussed in Algorithm 3.1 and construct corresponding MPC protocol $\Pi^{RandFloat1}$.

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

Protocol: $\Pi^{RandFloat1}$

Input: None

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

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

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

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

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

Protocol 4.1: MPC protocol for sampling uniform random floating-point $U^* \in \mathbb{D} \cap (0,1)$.

4.3 MPC Protocols for Snapping Mechanism

Recall that the snapping mechanism (cf. § 3.1) is defined as follows:

$$M_S(f(D), \lambda, B) = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(f(D)) \oplus S \otimes \lambda \otimes \text{LN}(U^*)\right\rceil_\Lambda\right). \tag{4.41}$$

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

$$\langle M_S(f(D), \lambda, B)\rangle = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(\langle f(D)\rangle) \oplus \langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)\right\rceil_\Lambda\right). \tag{4.42}$$

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

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

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

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

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

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

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

Protocol 4.2: MPC protocol for $\text{clamp}_B(x)$.

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

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

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

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

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

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

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

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

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

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

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

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

Protocol 4.3: MPC protocol for snapping mechanism.

4.4 MPC 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. § 3.2) use a discrete Laplace random variable $i \sim DLap\left(t = \frac{\Delta_r}{r\varepsilon}\right)$ to simulate a continuous Laplace random variable.

As discussed in § 3.2.1, one way to generate discrete Laplace random variable i is to use $Algo^{GeoExpBinarySearch}$ and $Algo^{TwoSideGeo}$. We provide the MPC protocols for both algorithms and the MPC protocols for the Integer-Scaling Laplace mechanism.

MPC Protocols for Binary Search Based Geometric Sampling Algorithm $\Pi^{GeoExpBinarySearch}(\lambda)$ samples a geometric random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ based on $Algo^{GeoExpBinarySearch}$ (cf. Algorithm 3.2).

In line $1-7$, each party locally calculates L_0, R_0, M_0 and Q_0 in plaintext to save MPC computations. In line $8-12$, the parties generate a floating-point random variable U_0 and use the comparison result of U_0 and Q_0 to choose one subinterval (either $(L_0 \dots M_0]$ or $(M_0 \dots R_0]$) and compute a flag fg_0 that indicates whether the termination condition of **WHILE** (cf. Algorithm 3.2) is satisfied. In line $16-26$, the parties execute the binary search for $ITER-1$ times. In line 19, the parties choose the correct split-point M_j using $\Pi^{COTMult}$ as follows:

$$\begin{aligned}\left\langle M_j \right\rangle^{B,UINT} = \quad & \Pi^{COTMult}\left(\left\langle cond_{M_j \le L_{j-1}} \right\rangle^{B}, \Pi^{UINT\_Add}\left(\left\langle L_{j-1} \right\rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\left\langle cond_{M_j \ge R_{j-1}} \right\rangle^{B}, \Pi^{UINT\_Sub}\left(\left\langle R_{j-1} \right\rangle^{B,UINT}, 1\right)\right) \\ & \oplus \Pi^{COTMult}\left(\left\langle cond_{L_{j-1} < M_j < R_{j-1}} \right\rangle^{B}, \left\langle M_j \right\rangle^{B,UINT}\right)\end{aligned} \tag{4.43}$$

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

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

Input: λ

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

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

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

IF $M_0 \leq L_0$

  $M_0 \leftarrow L_0 + 1$

ELSE IF $M_0 \geq R_0$

  $M_0 \leftarrow R_0 - 1$

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

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

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

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

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

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

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

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

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

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

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

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

  $\left\langle M_j\right\rangle^{B,UINT} \leftarrow$ Eq. (4.43)

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

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

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

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

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

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

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

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

Protocol 4.4: MPC protocol for sampling geometric random variable $x \sim Geo\left(p = 1-e^{-\lambda}\right)$.

MPC Protocols for Two-Side Geometric Sampling Algorithm Next, we construct $\Pi^{TwoSideGeometric}$ that converts a geometric random variable $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ to a discrete Laplace random variable $i \sim DLap\left(\frac{1}{\lambda}\right)$ based on $Algo^{TwoSideGeo}$ (cf. Algorithm 3.3).

In line 2, 3, the parties generate the sign s_j and integer part g_j of discrete Laplace random variable $i = (-1)^{s_j} \cdot g_j$. In line 4, the parties compute the flag $f\,g_j$ to record if the termination condition of **WHILE** loop (cf. Algorithm 3.3) is satisfied. In line 5, we extract the correct sign s and the number part g based on flags $\langle f\,g_0\rangle^B, \ldots, \langle f\,g_{iter-1}\rangle^B$.

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

Input: t

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

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

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

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

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

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

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

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

Protocol 4.5: MPC Protocol for sampling two-side geometric random variable $x \sim DLap(t)$.

Implementaion with SIMD. As discussed in paragraph 3.2.1, $Algo^{TwoSideGeo}$ iterates for $ITER$ times to increase the success probability ($> 1 - 2^{-40}$) of outputting the correct result. Since each iteration (line $2-4$) is independent, we uses SIMD to eliminate the FOR loop and increase the efficiency of $\Pi^{TwoSideGeometric}$. Specifically, we first generate $\langle s_{simd}\rangle^B = \left(\langle s_0\rangle^B, \ldots, \left(\langle s_{ITER-1}\rangle^B\right)\right)$ and $\langle g_{simd}\rangle^{B,UINT} = \left(\langle g_0\rangle^{B,UINT}, \ldots, \langle g_{ITER-1}\rangle^{B,UINT}\right)$ by running $\Pi^{RandBits}$ and $\Pi^{GeoExpBinarySearch}$ for single time but with $ITER$ bits in each wire. Note that $\langle s_{simd}\rangle^B$ and $\langle g_{simd}\rangle^{B,UINT}$ has the same number of Boolean GMW wires as $\left\langle s_j \right\rangle^B$ and $\left\langle g_j \right\rangle^{B,UINT}$ but $ITER$ number of bits in each wire. Then, we run the rest operations in the FOR loop with $\langle s_{SIMD}\rangle^B$ and $\langle g_{SIMD}\rangle^{B,UINT}$. We also applid this FOR loop elimination method in other MPC protocols that has independent operations in their FOR loops.

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

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

$$\left\langle M_{ISLap}\left(f_r(D), r, \Delta_r, \varepsilon\right)\right\rangle = \langle f_r(D)\rangle + \langle i\rangle \cdot r \tag{4.44}$$

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

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

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

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

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

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

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

Protocol 4.6: MPC Protocol for Integer-Scaling Laplacian mechanism.

4.5 MPC 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. § 3.2.2) use a symmetric binomial random variable $i \sim SymmBino(n, p = 0.5)$ to simulate a Gaussian random variable.

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

MPC Protocol for Symmetrical Binomial Sampling Algorithm $\Pi^{SymmBino}(n, p = 0.5)$ samples a symmetric binomial random variable $x \sim SymmBino(n, p = 0.5)$ based on $Algo^{SymmetricBinomial}$ (cf. Algorithm 3.4).

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

$$\begin{aligned} m &= \left\lfloor \sqrt{2}\cdot\sqrt{n}+1 \right\rfloor, \\ x_{min} &= -\frac{\sqrt{n}\cdot\sqrt{\ln\sqrt{n}}}{\sqrt{2}}, \\ x_{max} &= -x_{min}, \\ v_n &= \frac{0.4\cdot 2^{1.5}\cdot \ln^{1.5}(\sqrt{n})}{\sqrt{n}}, \\ \tilde{p}_{coe} &= \sqrt{\frac{2}{\pi}}\cdot(1-v_n)\cdot\frac{1}{\sqrt{n}}. \end{aligned} \tag{4.45}$$

Note that in line 15, the parties compute the condition $\left\langle cond_{x_{min}\leq x_j\leq x_{max}}\right\rangle^B \wedge \left\langle cond_{c_j==1}\right\rangle^B$ to record if the **WHILE** loop condition (cf. Algorithm 3.4) is satisfied. In line 16, the parties extract the correct $\langle i\rangle^{B,INT}$ based on flags $\langle fg_0\rangle^B,\ldots,\langle fg_{ITER-1}\rangle^B$.

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

Input: $\sqrt{n}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Protocol 4.7: MPC protocol for sampling symmetrical binomial random variable $i\sim SymmBino\left(n,p=0.5\right)$.

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

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

$$\langle M_{ISGauss}\left(f_r\left(D\right),r,\Delta_r,\varepsilon,\delta\right)\rangle=\langle f_r\left(D\right)\rangle+\langle i\rangle\cdot r, \tag{4.46}$$

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 4.8: MPC protocol for Integer-Scaling Gaussian mechanism.

4.6 MPC Protocols for Discrete Laplace Mechanism

Recall that the discrete Laplace mechanism $M_{DLap}(f(D), r, \varepsilon) = f(D) + Y$ (cf. § 3.3) use discrete Laplace random variable Y to perturb $f(D)$, where Y can be generated with $Algo^{GeoExp}$.

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

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

Input: n, d

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

1 : IF $d == 1$
2 : GO**TO** Line 7 // 1th loop terminates
3 : **FOR** $j \leftarrow 0$ **TO** $ITER_1 - 1$
4 : $\left\langle u_j\right\rangle^{B,UINT} \leftarrow \Pi^{RandInt}(d)$
5 : $\left\langle b_j^1\right\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(\Pi^{FL\_Exp}\left(\Pi^{FL\_Div}\left(\Pi^{UINT2FL}\left(\langle u\rangle^{B,UINT}\right),-d\right)\right)\right)$
6 : $\left\langle c_j\right\rangle^{B} = \langle b_1\rangle^{B}$
7 : **FOR** $j \leftarrow 0$ **TO** $ITER_2 - 1$
8 : $\langle b_2\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(e^{-1}\right)$
9 : $b_j^2 \leftarrow \neg\left(\langle b_2\rangle^{B}\right)$
10 : $\langle u\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(\langle u_0\rangle^{B,UINT},\ldots,\left\langle u_{ITER_1-1}\right\rangle^{B,UINT},\left\langle b_j^1\right\rangle^{B},\ldots,\left\langle b_{ITER_1-1}^1\right\rangle^{B}\right)$
11 : $\langle k\rangle^{B,UINT} \leftarrow \Pi^{ObliviousSelection}\left(0,\ldots,ITER_2-1,\left\langle b_0^2\right\rangle^{B},\ldots,\left\langle b_{ITER_1-1}^2\right\rangle^{B}\right)$
12 : $\langle x\rangle^{B,UINT} \leftarrow \Pi^{UINT\_Div}\left(\Pi^{UINT\_Add}\left(\langle u\rangle^{B,UINT},\Pi^{UINT\_Mul}\left(\langle k\rangle^{B,UINT},d\right)\right),n\right)$

Protocol 4.9: MPC Protocol for sampling geometric random variable $x \sim Geo\left(1-e^{-\frac{n}{d}}\right)$.

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

Input: n, d

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

1 : **FOR** $j \leftarrow 1$ **TO** $ITER-1$
2 : $\left\langle s_j\right\rangle^{B} \leftarrow \Pi^{RandBits}(1)$
3 : $\left\langle m_j\right\rangle^{B,UINT} \leftarrow \Pi^{GeometricEXP}(n,d)$
4 : $\left\langle f\,g_j\right\rangle^{B} = \neg\left(\left(\left\langle s_j\right\rangle^{B} == 1\right)\wedge\left(\left\langle m_j\right\rangle^{B,UINT} == 0\right)\right)$
5 : $\langle m\rangle^{B,UINT} \,\|\, \langle s\rangle^{B} \leftarrow \Pi^{ObliviousSelection}\left(\langle m_0\rangle^{B,UINT} \,\|\, \langle s_0\rangle^{B},\ldots,\langle m_{ITER-1}\rangle^{B,UINT} \,\|\, \langle s_{ITER-1}\rangle^{B},\langle f\,g_0\rangle^{B},\ldots,\langle f\,g_{ITER-1}\rangle^{B}\right)$
6 : Set $\langle s\rangle^{B}$ as the sign bit of $\langle m\rangle^{B,UINT}$
7 : $\langle Y\rangle^{B,INT} \leftarrow \langle m\rangle^{B,UINT}$

Protocol 4.10: MPC Protocol for sampling discrete Laplace random variable $x \sim DLap(t)$.

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

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

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

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

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

Protocol 4.11: MPC protocols for discrete Laplace mechanism.

4.7 MPC Protocol for Discrete Gaussian Mechanism

We provide the MPC protocols for sampling discrete Gaussian random variable and the discrete Gaussian mechanisms based on (cf. § 3.4).

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

Input: σ

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

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

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

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

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

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

Protocol 4.12: MPC Protocol for sampling discrete Gaussian random variable $Y \sim DGauss(\mu = 0, \sigma)$.

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

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

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

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

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

Protocol 4.13: MPC protocol for discrete Gaussian mechanism.

4.8 Security Discussion

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

5 Evaluation

Recall that the two challenges that we proposed in § 4:

1. What are the most efficient MPC protocols for arithmetic operations?
2. What are the most efficient sampling algorithms for differentially private mechanisms in MPC?

To answer the above questions, we implement and measure the performance of all the protocols we have discussed in § 4 in a semi-honest scenario. First, we benchmark the performance of integer, fixed-point and floating-point operations in different MPC protocols (BGMW, AGMW, and BMR). Next, we choose the most efficient MPC protocols for arithmetic operations and implement the differentially private mechanisms and corresponding sampling algorithms. The code can be found at [Zha22];

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 MPC-DP protocols.

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

5.1 Arithmetic Operations Performance Evaluation

In this section, we provide extensive benchmarks for arithmetic protocols (cf. § 4.2). We choose the most efficient MPC protocols for arithmetic operations based on the benchmark result.

Fixed-Point Benchmarks TODO: add benchmark result of BMR protocols, BMR protocol is only 1x-1.5x faster than BGMW in division and slower in all other operations. Therefore, we choose BGMW as main protocols

Tab. 5.1 shows that division is the slowest operation for BGMW fixed-point arithmetic. The reason is that the circuit we generated using HyCC [BDK+18] have a deeper depth of 6 432. In the LAN setting, the 5-party division is 2.2× slower than the 3-party division, whereas, in the WAN setting, the factor was 1.1.

In Tab. 5.2, natural exponentiation has the longest online run-times. In the LAN setting, as the number of parties increases from 3 to 5, the online run-times of operation subtraction and square root increase at most by a factor 15× and 3.8×.

Comparing the online run-times of fixed-point arithmetic from Tab. 5.1 and Tab. 5.2, we can see that the BGMW-based fixed-point arithmetic (with $SIMD = 1$) is more efficient than the AGMW-based fixed-point arithmetic only in operations such as exp2, exp and conversion operation with integer, but slower in all other operations. However, when we apply the SIMD technique, BGMW-based fixed-point arithmetic (with $SIMD = 1000$) is $4 \times -4000\times$ faster than the AGMW-based fixed-point arithmetic for all the operations in the LAN setting. In the WAN setting, the BGMW-based fixed-point arithmetic is $35 \times -2800\times$ faster than the AGMW-based fixed-point arithmetic except for addition and multiplication operations. That means if the functionalities (e.g., *FOR* loop in $\Pi^{prot:TwoSideGeometric}$) can be parallelized into independent and identical operations, the BGMW-based fixed-point arithmetic is a better option than AGMW-based fixed-point arithmetic in the LAN setting. Therefore, we choose BGMW-based fixed-point arithmetic to implement the MPC protocols.

Table 5.1: Online run-times in milliseconds (ms) of fixed-point (k=64, f=16) operations for the GMW (B) and TODO: add BMR protocol (Y). For the entries with $SIMD = 1\,000$, we specify the run-time of a single operation amortized over 1 000 SIMD values. We take the average over 10 protocol runs in the LAN and WAN environments.

		LAN			WAN		
Parties N	SIMD	N=2	N=3	N=5	N=2	N=3	N=5
FX_Add^B	1	27.49	29.22	39.75	507.80	516.25	556.10
FX_Sub^B	1	23.74	60.90	39.36	527.04	565.47	562.13
FX_Mul^B	1	197.14	199.17	219.78	1 487.11	1 772.54	1 829.94
FX_Div^B	1	25 690.22	26 924.64	59 848.43	336 749.59	354 113.67	391 287.07
FX_Lt^B	1	28.63	96.86	183.17	885.07	867.33	995.09
FX_Exp2^B	1	239.48	259.27	255.51	4 924.61	5 330.73	5 566.90
FX_Log2^B	1	2 880.34	2 898.00	2 887.24	17 599.92	19 747.61	22 030.32
FX_Exp^B	1	297.63	283.95	256.11	5 482.55	5 971.34	6 326.46
FX_Ln^B	1	2 910.26	2 995.85	2 985.79	18 903.29	19 957.76	22 555.63
FX_Sqrt^B	1	1 277.98	1 749.13	2 022.90	19 957.41	20 955.41	23 665.47
$Fx2Int^B$	1	14.35	15.53	17.53	675.69	795.63	948.14
$Fx2FL^B$	1	683.13	722.25	1 271.52	10 781.96	11 794.16	12 818.66
FX_Add^B	1 000	0.03	0.04	0.04	0.54	0.48	0.79
FX_Sub^B	1 000	0.01	0.01	0.04	0.51	0.56	0.52
FX_Mul^B	1 000	0.10	0.13	0.17	1.12	2.47	2.24
FX_Div^B	1 000	24.71	25.25	58.90	337.73	357.79	391.94
FX_Lt^B	1 000	0.05	0.15	0.14	0.79	0.80	1.00
FX_Exp2^B	1 000	0.21	0.41	0.47	4.75	5.68	6.14
FX_Log2^B	1 000	2.61	1.80	1.57	16.35	20.13	26.91
FX_Exp^B	1 000	0.24	0.45	0.55	5.66	6.37	6.65
FX_Ln^B	1 000	4.35	3.93	1.70	17.50	20.41	33.14
FX_Sqrt^B	1 000	4.62	4.33	5.11	19.44	21.60	32.91
$Fx2Int^B$	1 000	0.01	0.02	0.05	0.62	0.56	0.80
$Fx2FL^B$	1 000	0.25	0.40	0.55	7.36	9.64	10.38

Table 5.2: Online run-times in milliseconds (ms) of fixed-point (k=41, f=20) operations for the GMW (A). We take the average over 10 protocol runs in the LAN and WAN environments.

	LAN			WAN		
Parties N	N=2	N=3	N=5	N=2	N=3	N=5
FX_Add^A	0.10	0.23	0.24	0.19	0.25	0.28
FX_Sub^A	0.14	0.22	3.45	0.27	0.24	0.46
FX_Mul^A	16.05	38.24	25.20	531.76	437.77	448.93
FX_Div^A	219.16	755.79	1 650.56	8 968.31	10 198.43	11 210.29
FX_Lt^A	19.32	13.00	18.26	350.21	334.68	372.35
FX_Exp2^A	387.46	752.12	1 891.61	13 458.62	14 164.27	16 461.11
FX_Log2^A	181.10	215.64	292.53	4 371.94	4 716.72	5 565.20
FX_Exp^A	444.02	1 043.37	1 924.93	13 931.69	14 527.76	16 205.35
FX_Ln^A	223.29	280.02	308.78	4 864.74	5 049.57	5 652.10
FX_Sqrt^A	343.14	438.70	1 682.41	10 792.22	11 573.25	13 250.91
FX_Fx2FL^A	40.91	50.22	187.62	1 653.18	1 733.67	1 866.73
FX_Fx2Int^A	13.81	17.26	26.25	336.89	338.50	341.04

Floating-Point Benchmarks In Tab. 5.3, natural logarithm takes the longest online run-times in both LAN and WAN settings. As the number of parties grows from 3 to 5, the run-times of conversion operations to integer and fixed-point increase at most by a factor 3.08× and 3.66× ($SIMD = 1$), factor 2.2× and 1.9× ($SIMD = 1000$). Comparing the online run-times of floating-point arithmetic from Tab. 5.3 and Tab. 5.4, we can see that without applying SIMD, the BGMW-based floating-point arithmetic is faster ($1.3\times - 3\times$) than AGMW-based floating-point arithmetic in operations such as addition, subtraction, $<$, log2 and square root, but slower in all other operations. After applying the SIMD technique, the BGMW-based floating-point arithmetic is $50\times - 10911\times$ faster than AGMW-based floating-point arithmetic for all the operations in the LAN setting. Therefore, we choose BGMW-based floating-point arithmetic to implement MPC protocols for sampling algorithms.

Table 5.3: Online run-times in milliseconds (ms) of floating-point operations for the GMW (B) TODO: and BMR protocol (Y). For the entries with $SIMD = 1\,000$, we specify the run-time of a single operation amortized over 1 000 SIMD values. We take the average over 10 protocol runs in the LAN and WAN environments.

		LAN			WAN		
Parties N	SIMD	N=2	N=3	N=5	N=2	N=3	N=5
FL_Add^B	1	158.90	175.47	266.47	4 976.43	5 665.71	5 780.94
FL_Sub^B	1	151.38	167.19	252.31	5 053.17	5 599.27	5 970.80
FL_Mul^B	1	166.46	178.80	222.60	6 759.47	7 278.22	7 671.80
FL_Div^B	1	1 418.62	1 659.29	1 882.86	74 821.42	89 112.86	100 660.43
FL_Lt^B	1	47.08	52.42	132.53	1 638.13	1 751.99	1 840.56
FL_Exp2^B	1	1 958.28	2 391.79	3 392.22	112 465.43	129 661.24	140 737.75
FL_Log2^B	1	1 271.88	1 599.22	2 581.09	49 886.75	58 236.84	64 266.20
FL_Exp^B	1	2 122.94	2 564.97	3 461.87	118 774.84	137 115.63	150 439.77
FL_Ln^B	1	3 725.02	4 765.81	8 773.41	151 622.29	177 746.42	200 747.65
FL_Sqrt^B	1	887.72	1 081.30	1 514.39	44 584.20	51 846.96	57 450.22
$FL2Int^B$	1	327.69	451.41	1 390.65	12 943.05	14 093.49	16 199.04
$FL2Fx^B$	1	453.89	481.33	1 763.47	19 657.52	21 243.10	24 070.59
FL_Add^B	1000	0.23	0.34	0.61	4.73	5.18	5.81
FL_Sub^B	1000	0.23	0.31	0.55	4.85	5.37	5.47
FL_Mul^B	1000	0.23	0.29	0.27	6.67	7.30	7.85
FL_Div^B	1000	2.45	2.94	3.82	63.26	70.05	74.88
FL_Lt^B	1000	0.05	0.06	0.17	1.42	1.53	1.53
FL_Exp2^B	1000	1.78	2.05	2.66	84.15	96.90	105.37
FL_Log2^B	1000	1.19	1.59	2.23	42.75	48.91	52.82
FL_Exp^B	1000	1.94	2.28	2.57	90.90	104.04	113.61
FL_Ln^B	1000	4.12	5.57	7.54	128.44	146.59	163.34
FL_Sqrt^B	1000	0.14	0.30	0.59	2.48	2.89	3.30
$FL2Int^B$	1000	0.21	0.32	0.74	10.26	11.36	12.31
$FL2Fx^B$	1000	0.33	0.53	1.03	16.64	18.06	19.56

Table 5.4: Online run-times in milliseconds (ms) of floating-point operations for the GMW (A). We take the average over 10 protocol runs in the LAN and WAN environments. TODO: implement FL-FL2Fx and FL-FL2Int

	LAN			WAN		
Parties N	N=2	N=3	N=5	N=2	N=3	N=5
FL_Add^A	216.99	253.25	474.08	4 877.91	5 440.09	6 067.92
FL_Sub^A	216.63	314.82	491.58	5 000.17	5 356.96	6 113.30
FL_Mul^A	53.35	75.07	84.27	1 248.12	1 263.70	1 553.22
FL_Div^A	123.93	168.83	301.37	3 173.60	3 344.13	3 485.83
FL_Lt^A	66.03	84.35	197.75	1 361.24	1 566.96	1 779.53
FL_Exp2^A	569.54	953.94	1 784.02	9 105.86	10 054.34	13 147.79
FL_Log2^A	3 871.35	5 395.07	8 623.15	90 173.12	99 853.84	107 905.28
FL_Exp^A	620.11	966.76	1 684.20	9 748.59	10 695.65	11 560.34
FL_Ln^A	3 903.48	5 588.28	8 516.19	90 034.64	99 444.66	108 395.14
FL_Sqrt^A	1 575.21	2 372.36	3 603.36	31 263.47	33 778.76	36 776.32
FL_FL2Fx^A						
FL_FL2Int^A						

5.2 Evaluation of MPC Protocols for Differentially Private Mechanisms

Tab. 5.5 presents the combinations of different sampling algorithms to generate discrete Laplace random variables and discrete Gaussian random variables as discussed in § 4. We implement corresponding protocols in BGMW-based fixed/floating-point arithmetic. The benchmark results are listed in Tab. 5.6. We found that memory overflow (> 128 GB) happends for $DLap_1^B$, $DGau_1^B$ and $DGau_2^B$ when we set the failure probability $p < 2^{-40}$. The reason is that for failure probability $p < 2^{-40}$, discrete random variables need a larger number of iterations. TODO: Therefore, we need to reimplement the MPC protocols for $DLap_1$, $DGau_1$ and $DGau_2$ with AGMW-based fixed/floating protocols.

Table 5.5: Overview of sampling algorithms for generating discrete Laplace/Gaussian random variables.

	Sampling Algorithms		
Random Variable	$Algo^{TwoSideGeo}$ [Tea20a]	$Algo^{DLap}$ [CKS20]	$Algo^{DGau}$ [CKS20]
$DLap_1$	•		
$DLap_2$		•	
$DGau_1$	•		•
$DGau_2$		•	•

Table 5.6: Online run-times in milliseconds (ms) of fixed/floating-point Laplace/Gaussian sampling protocols for the GMW (B). We take the average over 10 protocol runs in the LAN and WAN environments.

	LAN			WAN		
Parties N	N=2	N=3	N=5	N=2	N=3	N=5
$DLap_1^{B,FX}$	—	—	—	—	—	—
$DLap_1^{B,FL}$	—	—	—	—	—	—
$DLap_2^{B,FX}$	60 946.33	62 613.41	144 253.44	893 460.44	947 818.24	1 040 754.52
$DLap_2^{B,FL}$	48 874.17	49 236.48	105 650.76	730 002.77	782 549.19	861 691.71
$DGau_1^{B,FX}$	—	—	—	—	—	—
$DGau_1^{B,FL}$	—	—	—	—	—	—
$DGau_2^{B,FX}$	—	—	—	—	—	—
$DGau_2^{B,FL}$	—	—	—	—	—	—

We present the performance evaluation of the MPC protocols for differentially private mechanisms in Tab. 5.7.

Laplace Mechanisms. Recall that Integer-Scaling Laplace mechanism M_{ISLap} achieves differential privacy protection effect as Laplace mechanism textitsecurely by re-scaling a discrete Laplace random variable. In our work, we generate the discrete Laplace random variable with sampling protocol Π^{DLap} (cf. Prot. 4.10) and set the failure probability as $p < 2^{-40}$. We can see that the M_{ISLap} with fixed-point implementation of Π^{DLap} is $1.09\times - 1.29\times$ slower than that with floating-point implementation. The reason is that the division operation in fixed-point is more expensive than floating-point for BGMW protocols.

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 online run-times performance and it is $5380\times - 11940\times$ 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 errors introduced by the Integer-Scaling Laplace mechanism M_{ISLap} by setting appropriate resolution parameter r (cf. § 3.2).

Gaussian Mechanisms. Recall that the Integer-Scaling Gaussian mechanism M_{ISLap} deploys the sampling protocol $\Pi^{SymmBinomial}$ under floating-point arithmetic to simulate the continuous Gaussian random variable. We implement $\Pi^{SymmBinomial}$ with BGMW-based floating-point arithmetic. We can see that the Integer-Scaling Gaussian mechanism M_{ISGau} is $2.08\times - 3.06\times$ faster than the Integer-Scaling Laplace mechanism M_{ISLap}.

Discrete Laplace Mechanisms. The discrete Laplace mechanism M_{DLap} deploys the same sampling protocol Π^{DLap} (cf. Prot. 4.10) as M_{ISLap}. We also implement M_{DLap} [EKM+14] for comparison. It can be seen, that M_{DLap} [EKM+14] is at least $1068\times$ faster than our implementation of M_{DLap} in BGMW-based fixed/floating-point arithmetic 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. TODO: We encounter memory overflow when implement BGMW-based M_{DGau}, try to implement it in AGMW-based floating-point arithmetic

Table 5.7: 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 SIMD values. We take the average over 10 protocol runs in the LAN and WAN environments.

			LAN			WAN		
Parties N	SIMD	Protocol	N=2	N=3	N=5	N=2	N=3	N=5
M_{Lap} [EKM+14] (insecure)	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**)	1	B, FX	72 296.53	71 352.89	160 443.40	949 907.44	994 913.88	1 091 406.72
M_{ISLap} (**this work**)	1	B, FL	55 858.07	47 186.90	111 232.66	846 435.93	908 322.81	1 001 687.22
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	134.97	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**)	1	B, FX	60 946.33	62 613.41	144 253.44	893 460.44	947 818.24	1 040 754.52
M_{DLap} (**this work**)	1	B, FL	48 874.17	49 236.48	105 650.76	730 002.77	782 549.19	861 691.71
M_{Dau} (**this work**)	1	B, Fx	—	—	—	—	—	—
M_{Dau} (**this work**)	1	B, FL	—	—	—	—	—	—

6 Related Work

Combining MPC and DP

To the best of our knowledge, Dwork et al. [DKM+06] was the first to consider deploying malicious secure MPC to aggregate and perturb data by generating the noise shares in a distributed setting. Dwork et al. [DKM+06] proposed methods to generate two types of noise: approximating Gaussian distribution with Binomial distribution, approximating scaled symmetric exponential distribution with Possion distribution. For binomial distribution, their protocol requires generating n uniform random bits that would be inefficient for a very large n (as discussed in § 3.2.2). For discrete Laplace distribution, the protocol requires securely evaluating a circuit to generating biased bits, that fails with non-zero probability and requires multiple iterations to make the failure probability negligible.

Eigner et al. [EKM+14] proposed an architecture called PrivaDA, that combined DP and MPC by generating Laplace and discrete Laplace noise using MPC protocols in a distributed manner. However, the Laplace noise suffers from the floating-point attack [Mir12] that is caused by the irregularities of floating-point and porous of Laplace distribution.

Wu et al. [WHWX16] proposed methods for generating Laplace and Gaussian noise in MPC using the central limit theory [AL06], i.e., the aggregation of n of Bernoulli random variable approximates a normal random variable ($\sqrt{n}\left(\frac{\sum_{i=1}^{n} Bern(0.5)}{n} - \mu\right) \approx \mathcal{N}\left(0, \frac{1}{4}\right)$). However, the central limit theory holds when $n \to \infty$, and there is no discussion about the relation between n and differential privacy.

Eriguchi et al. [EIKN21] provided two MPC-based differential privacy protocols for generating shares of finite-range discrete Laplace (FDL) noise and binomial noise. In contrast to discrete Laplace distribution, that can sample arbitrarily large integers, FDL can only generate integers in a specific range. The second protocol for generating binomial noise deploys pseudorandom secret-sharing [CDI05] for generating shares of uniform random variables non-interactively and use the binomial mechanism [ASY+18]. However, the binomial mechanism only satisfy computational differential privacy [MPRV09] (an relaxation of differential privacy definition that is only secure against computational bounded adversary).

TODO: literature about floating-point and fixed-point MPC protocols

7 Final Remarks

List of Figures

List of Tables

List of Abbreviations

Bibliography

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

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

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

[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.

[ASY+18] N. AGARWAL, A. T. SURESH, F. X. X. YU, S. KUMAR, B. MCMAHAN. **"cpSGD: Communication-efficient and differentially-private distributed SGD"**. In: *Advances in Neural Information Processing Systems* 31 (2018).

[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.

[BGW19] M. BEN-OR, S. GOLDWASSER, A. WIGDERSON. "Completeness theorems for non-cryptographic fault-tolerant distributed computation". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 351–371.

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

[BK15] E. BARKER, J. KELSEY. **"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"**. en. 2015.

[BKP+14] K. BRINGMANN, F. KUHN, K. PANAGIOTOU, U. PETER, H. THOMAS. **"Internal DLA: Efficient simulation of a physical growth model"**. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2014, pp. 247–258.

[BLO16] A. BEN-EFRAIM, Y. LINDELL, E. OMRI. **"Optimizing semi-honest secure multiparty computation for the internet"**. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 578–590.

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

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

[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.

[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.

[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.

[CHK+12] S. G. CHOI, K.-W. HWANG, J. KATZ, T. MALKIN, D. RUBENSTEIN. **"Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces"**. In: *Cryptographers' Track at the RSA Conference*. Springer. 2012, pp. 416–432.

[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.

[Cla15] J. CLARK. **"Why 2015 was a breakthrough year in artificial intelligence"**. 2015.

[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.

[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.

[EGL85] S. EVEN, O. GOLDREICH, A. LEMPEL. **"A randomized protocol for signing contracts"**. In: *Communications of the ACM* 28.6 (1985), pp. 637–647.

[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).

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

[GMW19] O. GOLDREICH, S. MICALI, A. WIGDERSON. "How to play any mental game, or a completeness theorem for protocols with honest majority". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, pp. 307–328.

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

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

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

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

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

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

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

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

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

[Knu14] D. E. KNUTH. **"Art of computer programming, volume 2: Seminumerical algorithms"**. Addison-Wesley Professional, 2014.

[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.

[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.

[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.

[PL15] M. PETTAI, P. LAUD. **"Combining differential privacy and secure multiparty computation"**. In: *Proceedings of the 31st Annual Computer Security Applications Conference*. 2015, pp. 421–430.

[Rab05] M. O. RABIN. **"How to exchange secrets with oblivious transfer"**. In: *Cryptology ePrint Archive* (2005).

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

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

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

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

[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.

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

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

[Vad17] S. VADHAN. "The complexity of differential privacy". In: *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 347–450.

[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.

[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.

[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.

[Zha22] L. ZHAO. **"Securely Realizing Output Privacy in MPC"**. In: GitHub, 2022.

[Zoh17] M. ZOHNER. **"Faster Oblivious Transfer Extension and Its Impact on Secure Computation"**. PhD thesis. Dissertation, Darmstadt, Technische Universität Darmstadt, 2016, 2017.

[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.

[Zum15] N. ZUMEL. **"A Simpler Explanation of Differential Privacy"**. 2015.

A Appendix

A.1 MPC Protocols

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

$$\left(\left\langle c_{a,b} \right\rangle, \left\langle \boldsymbol{y_{a,b}} \right\rangle^B\right) = \begin{cases} \left(\langle c_a \rangle, \langle \boldsymbol{y_a} \rangle^B\right), & \text{if } \langle c_a \rangle == 1 \\ \left(\langle c_b \rangle, \langle \boldsymbol{y_b} \rangle^B\right), & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 1 \\ (0,0) & \text{if } \langle c_a \rangle == 0 \wedge \langle c_b \rangle == 0, \end{cases} \tag{A.47}$$

which is equivalent to

$$\left\langle c_{a,b} \right\rangle = \langle c_a \rangle \oplus \langle c_b \rangle \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \tag{A.48}$$

$$\begin{aligned} \left\langle \boldsymbol{y_{a,b}} \right\rangle = \quad & (\langle c_a \rangle \oplus \langle c_b \rangle) \cdot \left(\langle \boldsymbol{y_a} \rangle^{B,UINT} \cdot \langle c_a \rangle \oplus \langle \boldsymbol{y_b} \rangle^{B,UINT} \cdot \langle c_b \rangle\right) \\ & \oplus (\langle c_a \rangle \wedge \langle c_b \rangle) \cdot \langle \boldsymbol{y_a} \rangle^{B,UINT} \end{aligned} \tag{A.49}$$

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

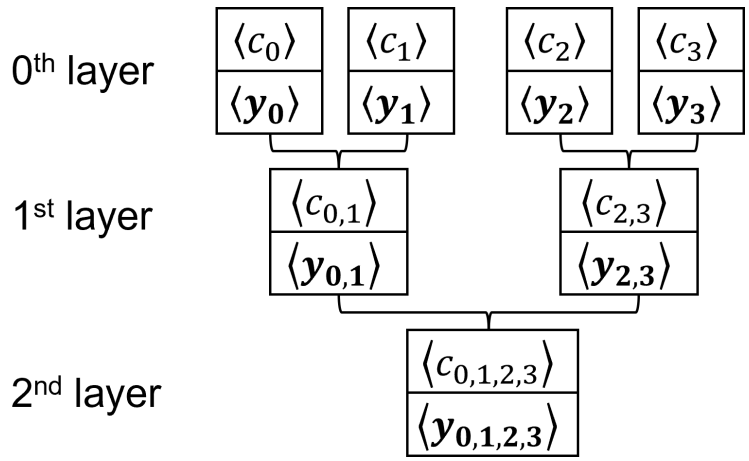

Figure A.1: Example inverted binary tree for $\Pi^{ObliviousSelection}$.