TECHNISCHE
UNIVERSITÄT
DARMSTADT

Thesis Type

This is the Title

Student Name

January 25, 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, January 25, 2022

Student Name

Abstract

Nowadays, the world has turned into an information-driven society where the distribution and processing of information is the most significant 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 new techniques that combine MPC and DP to guarantee privacy under floating-point arithmetic. We convert existing secure differentially private mechanisms for floating-point numbers and integer sampling methods into MPC protocols. In addition, we implement our MPC protocols and evaluate the computation and communication cost of different optimization techniques.

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 highly centralized database may contain sensitive information and be misused. Therefore, it is crucial to provide privacy protections for the 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's consider a typical scenario of PPML: Alice wishes to investigate if she has the genetic disease while keeping her genomic data secret. As a service provider, Bob has trained an ML model that can predict genetic diseases given genomic data. However, Bob also wants to keep his ML model private as it is his intellectual property that he aims to maintain. One unrealistic solution would be to rely on a trusted third party to do the analysis after being provided with Alice's genomic data and Bob's ML model. However, since such a trusted third party does rarely exist 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] show 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. As Mironov [Mir12] shows, the textbook noise generation methods can 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 [BDST20].

Research Goal In this thesis, we investigate how to combine differentially private algorithms and MPC techniques securely under floating-point arithmetics. Specifically, we design novel MPC protocols for secure differentially private algorithms and noise generation methods from [Mir12; Tea20; CKS20]. Besides, we aim at achieving DP and maintaining the accuracy of the computation result. We also aim at efficient MPC protocols by evaluating various MPC optimization techniques [BDST20] and their practical performance.

Contributions A major part of this thesis deals with the construction and optimizations of MPC protocols for noise generation methods. Specifically, we consider the outsourcing scenario [KR11], i.e., the data owners first secret share their private input to multiple ($N \geq 2$) non-colluding computing parties, and the computing parties execute the MPC protocols to securely compute the desired functionality. In our protocols, the computing parties jointly generate shares of an unknown public noise that provides a more accurate result than the scenario, where the parties generate a noise independently and aggregate the noise together.

TODO: describe novel MPC techniques...

Thesis Outline TODO: add after revision

2 Preliminaries

TODO: change after revision In this chapter, we present the essential background for this thesis. 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 $a, b, c \in \mathbb{N}$, we use $[a]$ to denote $\{x \in \mathbb{Z} \,|\, 1 \leq x \leq a\}$, (a, b) to denote $\{x \in \mathbb{R} \,|\, a < x < b\}$, and $[a, b]$ to denote $\{x \in \mathbb{R} \,|\, a \leq x \leq b\}$. $\{a, b, c\}$ is a set containing the three elements, and $(a_i)_{i\in[n]} = (a_1, \ldots, a_n)$ is a sequence of n elements.

Let $\mathbb{D}$ denote the set of floating-point numbers, and $\mathbb{D} \cap (a, b)$ contain floating-point numbers in the interval (a, b).The notation $\overline{b}_{(l)}$ indicates that bit $b \in \{0, 1\}$ is repeated l times. $(\cdot)_2$ is the binary representation of an integer.

Let $(P_i)_{i\in[N]}$ denote N computing parties. The shares of value x among N parties are denoted by $\langle x \rangle^S = \left(\langle x \rangle^S_1, \ldots, \langle x \rangle^S_N\right)$. For $i \in [N]$, $\langle x \rangle^S_i$ is hold by party P_i. $S \in \{A, B, Y\}$ denotes the sharing type: A for arithmetic sharing, B for Boolean sharing, Y for Yao sharing. B2A denotes the share conversion from Boolean sharing to arithmetic sharing, and other share conversions are defined similarly as discussed in [DSZ15]. For $\boldsymbol{x} \in \{0, 1\}^\ell$, $\langle \boldsymbol{x} \rangle^{B,D}$ is a vector of ℓ shared bits which is interpreted as an ℓ-bit value of data type D. For example, $\langle \mathbf{011} \rangle^{B,UI}$ is interpreted as a share of 3-bit unsigned integer with value 3. $D \in \{UI, SI, FP, FL\}$ indicates the data type: UI for unsigned integer, SI for signed integer, FP 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 $\langle \boldsymbol{a} \rangle^{B,D} \odot \langle \boldsymbol{b} \rangle^{B,D}$ be the arithmetic operations on two bit vectors $\langle \boldsymbol{a} \rangle^{B,D}$ and $\langle \boldsymbol{b} \rangle^{B,D}$ shared with Boolean sharing, where $\odot \in \{+, -, \cdot, \div, >, ==\}$.

Let $\ln\left(\langle \boldsymbol{a} \rangle^{B,D}\right)$, $2^{\langle \boldsymbol{a} \rangle^{B,D}}$, $e^{\langle \boldsymbol{a} \rangle^{B,D}}$, $\left|\langle \boldsymbol{a} \rangle^{B,D}\right|$, $\left\lfloor \langle \boldsymbol{a} \rangle^{B,D} \right\rfloor$, $\left\lceil \langle \boldsymbol{a} \rangle^{B,D} \right\rceil$, $\langle \boldsymbol{a} \rangle^{B,D} \mod \langle \boldsymbol{b} \rangle^{B,D}$ denote the arithmetic operation of bit vector $\langle \boldsymbol{a} \rangle^{B,D}$ and $\langle \boldsymbol{b} \rangle^{B,D}$.

$\langle a \rangle^B \cdot \langle \boldsymbol{b} \rangle^{B,D}$ between a Boolean sharing bit $\langle a \rangle^B$ and a vector of Boolean sharing bits $\langle \boldsymbol{b} \rangle^{B,D}$ represents bitwise $\wedge$ operations between $\langle a \rangle^B$ and every Boolean sharing bit $\langle b \rangle^B \in \langle \boldsymbol{b} \rangle^{B,D}$.

For data type conversions, we denote $\langle \boldsymbol{a} \rangle^{B,FL} = \text{UI2FL}\left(\langle \boldsymbol{a} \rangle^{B,UI}\right)$ the conversion from an unsigned integer to a floating-point number. Other data type conversion operations are defined in a similar manner.

2.2 Secure Multi-Party Computation

Yao [Yao86] first introduces the secure two-party computation with Yao's Millionaires' problem (two millionaires wish to know who is richer without revealing their actual wealth) and proposes the garbled circuit protocol as a solution. Afterwards, Beaver, Micali and Rogaway (BMR) [BMR90] generalize Yao's garbled circuit protocol to multi-party settings. Goldreich, Micali and Wigderson (GMW) [Gol87] use Boolean sharing to extend secure two-party computation to multi-party settings. Generally, the execution of MPC protocols is divided into the offline phase (no private inputs are involved) and the online phase (private inputs is 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 was first defined by Rabin [Rab05] as a mode of transferring information, where the receiver P_R receives the message from the sender P_S successfully with probability $p = 0.5$, and the sender is oblivious of whether the message was received. Later Even et al. [EGL85] redefine 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 P_S and a choice bit c from P_R, and outputs $\perp$ to P_S, only x_c to P_R. $\binom{2}{1}$-OT guarantees that P_R does not learn anything about c and that P_R does not learn about x_{1-c}. Impagliazzo and Rudich [IR90] show 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] propose 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] propose special OT functionalities for optimization of MPC protocols such as correlated OT (C-OT) and random OT (R-OT). In C-OT, the sender P_S inputs a correlation function f_Δ (e.g., $f_\Delta(x) = x \oplus \Delta$) and receives random

values x_0 and $x_1 = f_\Delta(x_0)$, the receiver P_R inputs a choice bit c and receives x_c. In R-OT, P_S has no inputs and receives random values (x_0, x_1), P_R 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 online phase of MPC protocols. For a multiplication triple $\left(\langle a\rangle^S, \langle b\rangle^S, \langle c\rangle^S\right)$ with $S \in \{B,A\}$, $c = a \wedge b$ is for Boolean sharing (cf. paragraph 2.2.5) and $c = a \times b$ is for arithmetic sharing (cf. paragraph 2.2.5). Multiplication triple can be generated using C-OT (cf. § 2.2.1) as Braun et al. [DSZ15; BDST20] show. 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 Yao's Garbled Circuit Protocol

Yao [Yao86] introduces 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.

Execution of Garbled Circuit Protocol

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 (with input wire:a and b, 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 permute 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 $\widetilde{C}$, P_E also needs the wire keys corresponding to its input. For each of P_G's input wire i with correspond 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) guarantess 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 $\widetilde{C}$. 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 (cf. § 2.2.2). 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 output wire. Point and permute [BMR90] 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 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 of P_E. Three-halves garbling [RR21] requires less 25% communication bits than two-halves garbling at the cost of more computation.

2.2.3 Beaver-Micali-Rogaway (BMR)

BMR protocol [BMR90] extends Yao's garbled circuit protocol [Yao86] to the multi-party setting. Recall that in Yao's garbled circuit protocol (cf. § 2.2.2), the circuit is first garbled by one party and evaluated by another party. BMR protocol enables the multi-party computation by garbling the circuit jointly by all parties, and then, each party evaluates the garbled circuit locally. Ben-Efraim et al. [BLO16] propose several optimization techniques for BMR protocol, e.g., OT-based garbling, BGW [WOG88]-based garbling.

2.2.4 Goldreich-Micali-Wigderson (GMW)

GMW protocol [Gol87] 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 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 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, 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 $z = z_0 \oplus z_1$.

2.2.5 Secret Sharing and Sharing Conversions

In this section, we detail the sharing types and conversions used in this work based on works by [DSZ15].

Secret Sharing

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 (B) A bit x is shared among N parties as $\left(\langle x\rangle_1^A, \ldots, \langle x\rangle_N^A\right) \in \{0,1\}^N$. 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 (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 [**Braun**] shows.

Sharing Conversions

Different sharing types and corresponding MPC protocols have advantages in specific operations, e.g., arithmetic GMW (cf. § 2.2.4) allows parties to execute linear operations locally, and BMR (cf. § 2.2.3) is often more efficient for comparison. Therefore, we use the sharing conversion methods introduced in works [DSZ15; BDST20].

2.2.6 MOTION Framework

We build upon the MOTION framework [BDST20] that provides the following novel features:

1. Support for MPC with N parties (tolerating up to $N-1$ passive corruptions) and sharing conversions.
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 examines differential privacy in a formal mathematical view. We first briefly introduce basic knowledge about the probabilistic distribution and random variable generation methods. Then, we recap traditional privacy preservation methods and their limitations. Next, we present and formalize the definition of differential privacy. Finally, we present the differentially private mechanisms for realizing differential privacy.

2.3.1 Probabilistic Distribution and Random Variable Generation

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 (PDF):*

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

$Uni(a,b)$ denotes the continuous uniform distribution with parameters a and b. $x \sim Uni(a,b)$ is a continuous uniform random variable.

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

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

$Expon(\lambda)$ denotes the exponential distribution with parameter λ. $x \sim Expon(b)$ is a exponential random variable. The cumulative distribution function (CDF) of exponential distribution is defined as follows:

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

Definition 2.3.3 (Laplace distribution [GRS12])**.** *The Laplace distribution with location parameter $\mu = 0$ and scale parameter b, has the following probability density function (PDF):*

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

$Lap(b)$ denotes the Laplace distribution with scale parameter b. $x \sim Lap(b)$ is a Laplace random variable. The Laplace distribution is also called the double exponential distribution because it can be thought of as the exponential distribution assigned a randomly chosen sign.

The cumulative distribution function (CDF) of Laplace distribution is defined as follows:

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

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

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

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

Discrete Probability Distribution

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

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

$Bern(p)$ denotes the Bernoulli distribution with parameters 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 $x \in [0,n]$:*

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

$Bino(n,p)$ 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 symmetrical about the y-axis, which we denote as $SymmBino(n, p = 0.5)$.

Definition 2.3.7 (Geometric distribution)**.** *The geometric distribution with parameter $p \in [0,1]$, has the following probability mass function for $x \in \mathbb{Z}^+$:*

$$\begin{aligned} Geo(p) &= \Pr(x \mid p) \\ &= (1-p)^{x-1} p \end{aligned} \tag{2.10}$$

$Geo(p)$ 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 trials up to and including the first success (with probability p). The cumulative distribution function of geometric distribution is $\Pr(x \leq X) = 1 - (1-p)^X$.

Definition 2.3.8 (Two-Side geo Geometric distribution)**.** *The two-side geometric distribution with parameter $p = 1 - e^{-\lambda}$ with $p \in [0,1]$, has the following probability mass function for $x \in \mathbb{Z}$:*

$$\begin{aligned} DGeo(p) &= \Pr\left(x \mid p = 1 - e^{-\lambda}\right) \\ &= \frac{(1-p)^{|x|} p}{2-p} \\ &= \frac{e^{-\lambda|x|} \cdot \left(1 - e^{-\lambda}\right)}{1 + e^{\lambda}} \end{aligned} \tag{2.11}$$

The two-side geometric distribution is also called discrete Laplace distribution $DLap(p)$. $DGeo(p)$ denotes the two-side geometric distribution with parameter $p = 1 - e^{-\lambda}$. $x \sim DGeo(p)$ is a two-side geometric random variable. $DGeo(p)$ can be generated by left-shifting $Geo(p)$ along x-axis by 1, mirroring along the y-axis and getting scaled such that for $DGeo(p)$'s PMF: $\sum_x \Pr(x \mid p) = 1$.

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 $x \in \mathbb{Z}$:*

$$\begin{aligned} DGauss(\mu, \sigma) &= \Pr(x \mid \mu, \sigma) \\ &= \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sum_{y \in \mathbb{Z}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}} \end{aligned} \tag{2.12}$$

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

Floating-Point Numbers

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

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

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

Probability Sampling Methods

Theorem 1 (Inverse Sampling Method [Dev86, 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 $F^{-1}(U)$ has a distribution function F. Also, if X has distribution function F , then $F(X)$ is uniformly distributed in $[0,1]$.

Inverse sampling method is a common sampling method when the inverse function of the desired probabilistic distribution's CDF is available. For example, we can sample a Laplace random variable $Y \sim Lap(b)$ from an exponential distribuion with CDF: $F(x \mid b) = 1 - e^{-\frac{x}{b}}$ as follows [Knu14, Chapter 3.4]:

1. Sample $U \sim Uni(0,1) \setminus 1$ and $Z \sim Bern(0.5)$
2. $F^{-1}(U) = -b \cdot \ln(1-U)$
3. $Y \leftarrow (2Z-1) \cdot b \ln(1-U)$

Sampling from a Bernoulli Distribution Algorithm 2.1 samples $x \sim Bern(p)$ based on the comparison result between the generated uniform random variable u and probability p.

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

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

Algorithm 2.1: Algorithm for sampling $x \sim Bern(p)$.

Sampling from a Geometric Distribution Algorithm 2.2 [Wal74; Tea20] generates a geometric random variable $x \sim Geo(0.5)$ by first generating an 8-bit random string $r \in \{0,1\}^8$ (i.e., eight Bernoulli trials) and counting its leading zeros (i.e., number of trials before the first success) into x. If all the bits in r are zeros, an new 8-bit random string is generated and its leading zeros is counted into x again. This process repeats until the generated random strings contain one (i.e., the first success trial). In practical implementation, we can increase the length of random string r to reduce the iteration times of the **WHILE** loop (cf. Line 2).

Algorithm: $Algo^{Geometric}$

Input: **None**
Output: $x \sim Geo(0.5)$
1 : $x \leftarrow 1$
2 : **WHILE** $r == 0$
3 : $r \leftarrow Random8bits()$
4 : $x \leftarrow x + LeadingZeros(r)$
5 : **RETURN** x

Algorithm 2.2: Algorithm for sampling $x \sim Geo(0.5)$.

revised based on feedback

2.3.2 Traditional Methods for Privacy Preservation

Suppose a fictitious hospital has collected massive data from thousands of patiens and wants to make the data available to academic researchers such as data analysts. However, the data contains sensitive information of 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), to preserve the patients' privacy, it must take specific privacy perservation measures before releasing the data.

Let us assume that the released data is already anonymized by removing the identifying features such as the patients' name and social security number (SSN). 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 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] introduces two techniques to achieve $k-anonymity$: data generalization and suppression. The former method makes the data less informative by mapping specific attribute values to a broader value range, and the latter method removes specific attribute

values. As [**tabular:4-anonymousinpatientmicrodata**] 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., Zip Code: $130**$, 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 so-called homogeneity attacks and background knowledge attacks [MKGV07]. One example for a background knowledge attack is that, suppose we know one patient who is about thirty years old, has visited the hospital and is in the records of Tab. 2.4, then we could conclude that he has cancer. Afterward, $l-Diversity$ [MKGV07] is proposed to overcome the shortcoming of $k-anonymity$ by preventing the homogeneity of sensitive attributes in the equivalent classes. 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 $l-diversity$. $t-closeness$ requires that 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 less than the given threshold t. However, $t-closeness$ is later showed to significantly affect the quantity of valuable information the released data contains [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 for the hostipal 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 result, releasing approximate statistics can prevent the above attack. However, Dinur et al. [DN03] shows that the adversary can reconstruct the entire database when sufficient queries are allowed and the approximate statistics error is bound to a certain level. Therefore, there are fundamental limits between what privacy protection can achieve and what useful statistical information 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.

2.3.3 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, there are n students, and each student has a sensitive information bit $X_i \in \{0,1\}$, where 0 denotes *never cheated* and 1 denotes *have cheated*. Every student want to keep their 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

$$Y_i = \begin{cases} X_i & \text{with probability } p \\ 1 - X_i & \text{with probability } 1-p \end{cases} \tag{2.14}$$

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

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

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

$$Var\left[\tilde{C}\right] = Var\left[\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2\gamma}\left(Y_i - \frac{1}{2} + \gamma\right)\right]\right] = \frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var[Y_i]. \tag{2.15}$$

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

$$\begin{aligned}\frac{1}{4\gamma^2 n^2}\sum_{i=1}^{n} Var\left[Y_i\right] &= \frac{1}{4\gamma^2 n} Var\left[Y_i\right] \\ &\leq \frac{1}{16\gamma^2 n}.\end{aligned} \tag{2.16}$$

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

$$\Pr(|X-\mu| \geq t) \leq \frac{\sigma^2}{t^2}, \tag{2.17}$$

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

$$\begin{aligned}\Pr\left(\left|\tilde{C} - CheatFraction\right| \geq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\leq O(1) \\ \Pr\left(\left|\tilde{C} - CheatFraction\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)\right) &\geq O(1),\end{aligned} \tag{2.18}$$

and $\left|\tilde{C} - CheatFraction\right| \leq O\left(\frac{1}{\gamma\sqrt{n}}\right)$ with high probability. The error term $\left|\tilde{C} - CheatFraction\right| \rightarrow 0$ as $n \rightarrow \infty$ with high probability. The conclusion is that the error increases as the privacy protection level increases $\gamma \rightarrow 0$. To maintain accuracy, more data $n \rightarrow \infty$ 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 as $D \in \mathcal{X}^n$. 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 [**goldreich2019play**].

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 can be 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 $\vec{X}$ contains 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 § 2.3.3 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 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 set such that when the adversary has $T < M(D) < 1$, he knows M has input $D = D_1$ (or $D = D_0$, when $0 < M(S) \leq T$) .

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

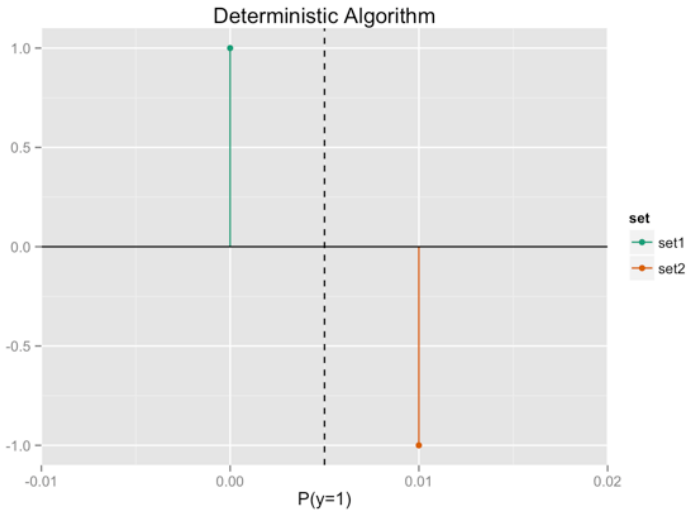

Figure 2.2: Deterministic algorithm (need reproduce).

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

TODO: need reproduce following figures

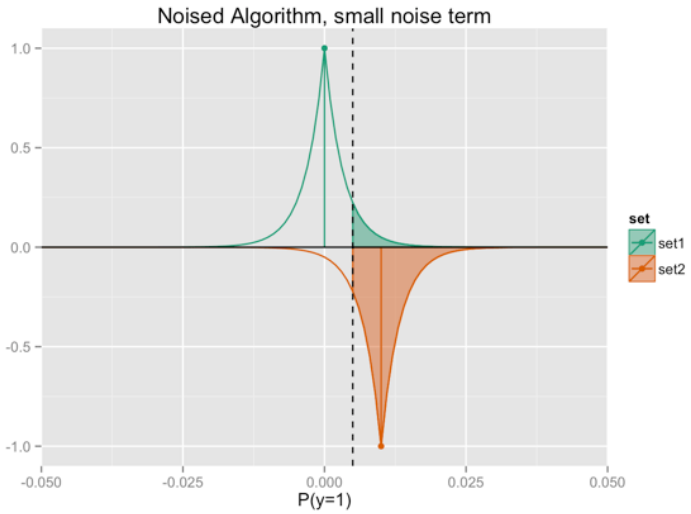

Figure 2.3: Indeterministic algorithm with small noise ($b = 0.005$) (need reproduce).

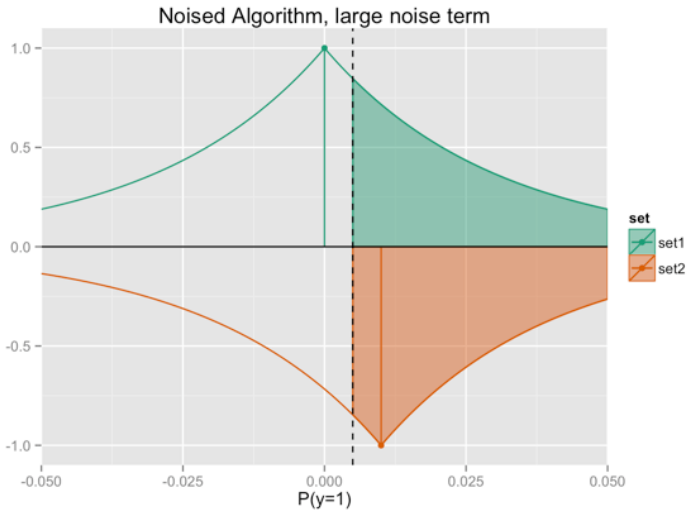

Figure 2.4: Indeterministic algorithm with large noise ($b = 0.05$) (need reproduce).

Properties of Differential Privacy

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

Post-Processing

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

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

Group Privacy

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

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

Basic Composition

Theorem 4. *Suppose $M = (M_1 \dots M_k)$ is a sequence of $\left(\varepsilon_[i], \delta_i\right)$-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.

Discussion about Differential Privacy

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

In centralized DP, all data is stored centrally and managed by a trusted curator before the differentially private mechanism is applied. As Fig. 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 the 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. However, the disadvantage is that the collected data may contain too much noise and decrease the utility.

TODO: reproduce following figures

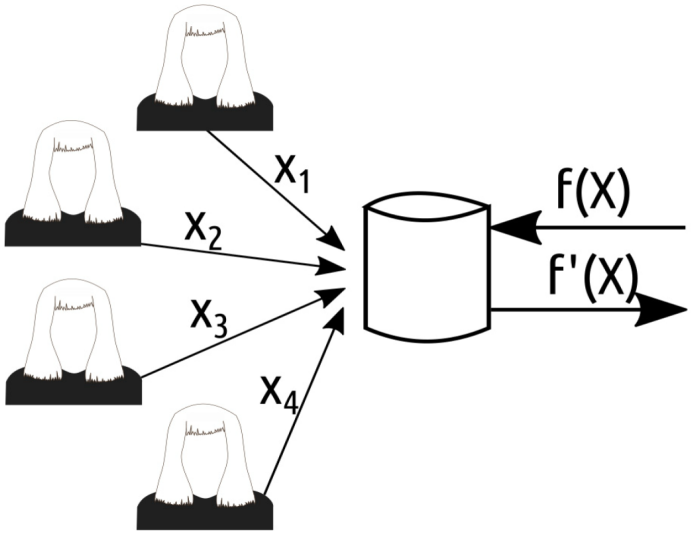

Figure 2.5: Centralized DP mode (need reproduce).

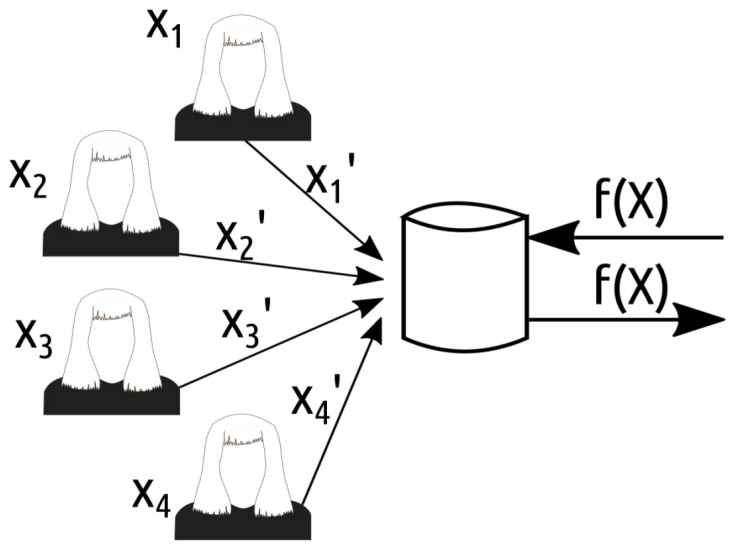

Figure 2.6: Local DP mode (need reproduce).

Advantages of Differential Privacy From the example § 2.3.3, we found that DP can still protect privacy even if the adversary knows the database. Generally speaking, DP ensures privacy protection by making no assumption about the adversary's auxiliary information (even when the adversary is the data provider) or computational strategy (regarding the complexity of modern cryptography) [Vad17]. In addition, DP provides a quantitive theory about safely releasing data and maintaining certain level of accuracy.

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

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

Implementation of DP mechanisms. The theory of DP operates on the real number field. However, because of the actual machine's limitation, the implementation of differentially private mechanisms based on floating-point or fixed-point number only provides an approximation

of the mathematical abstractions. [Mir12] shows the irregularities of floating-point implementations of the Laplace mechanism results in a porous distribution over double-precision number and leads to the breach of differential privacy. Further, [GMP16] prove that any differentially private mechanism that perturbs data by adding noise can break the DP when implemented with a finite precision regardless of the actual implementation.

2.3.4 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 mechanisms to realize DP.

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 $M(X) = f(X) + (Y_1, \ldots, Y_k)$, where the Y_i are independent Laplace random variables drawn from distribution $Laplace(Y_i \mid b) = \frac{1}{2b} e^{\left(-\frac{|Y_i|}{b}\right)}$ with $b = \frac{\Delta_1^{(f)}}{\varepsilon}$.*

Theorem 5. *Laplace Mechanism preservers ε-DP [DR$^+$14].*

Definition 2.3.17 (Privacy Loss [DR$^+$14]). *Let X and Y be two random variables. The privacy loss random variable $\mathcal{L}_{X||Y}$ is distributed by drawing $t \sim Y$, and outputting $\ln(\frac{\Pr[X=t]}{\Pr[Y=t]})$.*

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

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

(ε, δ)-Differential Privacy

ε-DP has 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 2.3.18 (Gaussian Mechanism [DR$^+$14]). *Let $f : \mathcal{X}^n \to \mathbb{R}^k$. The Gaussian mechanism is defined as $M(X) = f(X) + (Y_1, \ldots, Y_k)$, where the Y_i are independent Gaussian random variables drawn from distribution $Gauss\left(Y_i|\mu, \sigma^2\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{Y_i-\mu}{\sigma}\right)^2}$ with $\mu = 0$, $\sigma^2 = 2\ln\left(\frac{1.25}{\delta} \cdot \left(\frac{\Delta_2^{(f)})^2}{\varepsilon^2}\right)\right)$.*

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

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

3 Secure Differentially Private Mechanisms under Floating-Point Arithmetic

In this chapter, we present three existing differentially private mechanisms [Mir12; Tea20; CKS20] that is secure under floating-point arithmetic. We construct corresponding MPC protocols for those differentially private mechanisms in § 4.

Recall that differentially private mechanisms (cf. § 2.3.4) guarantee differential privacy by adding appropriately chosen random noise to a query function $f(D)$, which can be expressed as follows:

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

where D is the database, Y is the noise term.

Generally, the security analysis of differentially private mechanisms is based on the following assumptions:

1. Computations are performed on real numbers, requiring machines with infinite precision.
2. The noise is sampled with a probability close to the theoretically correct probability.

However, for the practical implementations of differentially private mechanisms, we only have machines with finite precision and typically use floating-point arithmetic (cf. § 2.3.1) to approximate calculations of real numbers. Mironov [Mir12] shows that the porous distribution of the Laplace Mechanism implemented with the textbook noise sampling methods under floating-point arithmetic can lead to severe differential privacy breaching, and proposes the snapping mechanism to avoid such security issues by rounding and smoothing the output of the Laplace Mechanism (cf. 2.3.16). Team [Tea20] introduces an alternative secure approach that scales integer noise under floating-point arithmetic to approximate continuous noise and yield better accuracy than the snapping mechanism. Canonne et al. [CKS20] provide algorithms to directly sample discrete Laplace and Gaussian noise for the query function $f(D) \in \mathbb{Z}$ in the integer field.

3.1 Snapping Mechanism

In this section, we introduce the snapping mechanism [Mir12] and its implementations under floating-point arithmetic based on [Cov19].

Recall that random variable Y from Laplace distribution $Lap(\lambda)$ (cf. 2.3.3) can be generated with the inverse sampling method (cf. Theorem 1) and reformulated as follows:

$$Y = S \cdot \lambda \ln(U), \tag{3.1}$$

where $S \in \{-1, +1\}$ is the sign, λ controls the magnitude of Y, and U is a uniform random variable in interval $(0, 1]$.

Recall that in paragraph 2.3.3, the porous distribution and rounding effects of floating-point arithmetic can lead to privacy breach of the Laplace Mechanism (cf. 2.3.16). To mitigate the attack, Mironov [Mir12] proposes the snapping mechanism to guarantee DP under floating-point arithmetic with rounding and clamping operations.

The snapping mechansim is defined as follows:

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

TODO: explain exact rounding

Let $\mathbb{D}$ denote the set of floating-point numbers, and $\mathbb{D} \cap (a, b)$ contain 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. Function $\text{clamp}_B(x)$ limits the output to the interval $[-B, B]$ by outputing B if $x > B$, $-B$ if $x < -B$, and x otherwise. $\oplus$ and $\otimes$ are the floating-point implementations of addition and multiplication. S is the sign of the noise and uniformly distributed over $\{-1, 1\}$. U^* is a *uniform* distribution over $\mathbb{D} \cap (0, 1)$, and generates floating-point number with probability proportional to its *unit in the last palce* (ulp), i.e., spacing between two consecutive floating-point numbers. $\text{LN}(\cdot)$ is the natural logarithm implementation under floating-point with exact rounding. Λ is the smallest power of two greater than or equal to λ, and we have $\Lambda = 2^n$ such that $2^{n-1} < \lambda \leq 2^n$ for $n \in \mathbb{Z}$. $\lfloor \cdot \rceil_\Lambda$ exactly rounds its input to the nearest multiple of Λ by manipulating the binary floating-point representation of the input. Note that the snapping mechanism assumes that the *sensitivity* Δ_1^f of query function f is 1, which can be extended to an arbitrary query function f' with *sensitivity* (cf. 2.3.15) $\Delta_1^{(f')} \neq 1$ by scaling the output of f' with $f = \frac{f'}{\Delta_1^{(f')}}$.

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 snapping mechanism consists of the following steps:

1. Calculation of $\text{clamp}_B(\cdot)$.
2. Generation of U^* and S.
3. Floating-point arithmetic operations: $\text{LN}(\cdot)$, $\oplus$, $\otimes$.
4. Calculation of Λ.
5. Rounding to the nearest multiple of Λ.

We briefly introduce how to generate U^*, S and round a floating-point number to the nearest multiple of Λ. The rest steps are trival to realize under floating-point arithmetic.

Generation of U^* and S

Sign $S \in \{-1, 1\}$ is a random variable that can be generated by tossing an unbiased coin.

U^* is the *uniform* distribution (cf. 2.3.1) over $\mathbb{D} \cap (0, 1)$ and can be represented in IEEE 754 floating-point (cf. § 2.3.1) as follows:

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

As stated above, each floating-point number sampled from U^* should be output with a probability proportional to its ulp. We sample a floating-point number from U^* using Algorithm 3.1 [Wal74; Mir12], i.e., independently sampling a geometric distribution random variable $x \sim Geo(0.5)$ (see Algorithm 2.2 for details), and setting U^*'s biased exponent $e = 1023 + x$, then sampling U^*'s significant bits $(d_1, \dots, d_{52})$ at random from $\{0, 1\}^{52}$.

Algorithm: $Algo^{RandFloat1}$

Input: **None**
Output: $U^* \in \mathbb{D} \cap (0, 1)$
1 : $(d_1, \dots, d_{52}) \xleftarrow{\$} \{0, 1\}^{52}$
2 : $x \leftarrow Algo^{Geometric}$
3 : $e \leftarrow 1023 + x$
4 : **RETURN** $U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$

Algorithm 3.1: Algorithm for random floating-point number from U^*.

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

Intuitively, *uniformly* sampling a floating point number can be thought as first drawing a real number in interval $(0, 1)$ at random and rounding it to the nearest floating-point number.

However, the floating-point numbers are discrete and not equidistant. For example, there are exactly 2^{52} representable reals in interval $[.5, 1)$ and 2^{52} reals in interval $[.25, .5)$. If we only use the floating-point numbers with equal distance to each other in interval $(0, 1)$, a large part of floating-point numbers would be ignored. As discussed in [Mir12], a better approach is to sample a floating-point number with probability proportional to its ulp (i.e., spacing to its consecutive neighbor). With $x \sim Geo(0.5)$, we have a total probability $p = 0.5$ for the floating-point numbers with exponent $e - 1023 = -1$ between $(0, 1)$, a total probability, $p = 0.25$ for the floating-point numbers with exponent $e - 1023 = -2$ between $[0, 0.5)$, etc. The total probability of the floating-point numbers that can be sampled in interval $\mathbb{D} \cap (0, 1)$ is $\sum_{i=1}^{\infty} \frac{1}{2^i} \approx 1$.

TODO: image about uniform sampling of floating point number

Rounding x to the nearest multiple of $\Lambda = 2^n$

We represent x in IEEE 754 floating-point (cf. § 2.3.1) as follows:

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

$\lfloor x \rceil_\Lambda$ is done in three steps:

1. $x' = \frac{x}{\Lambda} = x \cdot 2^{-n}$,
2. $x'' = \lfloor x' \rceil$,
3. $\lfloor x \rceil_\Lambda = x'' \cdot \Lambda = x \cdot 2^n$,

where $\lfloor \cdot \rceil$ rounds the input to the nearest integer.

The first step ($x' = \frac{x}{\Lambda}$) and last step ($\lfloor x \rceil_\Lambda = \Lambda \cdot x''$) are calculated by manipulating (substraction or addition) the exponent of x and x''. Note that one exception is when $x = 0$ or $x'' = 0$, because for floating-point numbers $0 = (-1)^0 (1.\overline{0})_2 \times 2^{(0000000000)_2 - 1023}$, $0 \times 2^n \neq (-1)^0 (1.\overline{0})_2 \times 2^{(0000000000)_2 - 1023 + n}$ as discussed in [Com19].

Calculate $x'' = \lfloor x' \rceil$ Suppose $x' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^{(e_1 \dots e_{11})_2 - 1023 - n}$. Let $m = (e_1 \dots e_{11})_2 - n$, $y = m - 1023$ and we have

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

The calculation of $x'' = \lfloor x' \rceil$ is categorized in five cases depending on the value of unbiased exponent y.

Case 1: $y \geq 52$
According to [Com19], when the biased exponent y is greater than or equal to 52, x' is an integer. Then, we have $x'' = (-1)^S (1.d_1 \dots d_{52})_2 \times 2^y$.

Case 2: $y = 0$
When $y = 0$, we have $x' = (-1)^S(1.d_1d_2\dots d_{52})_2 \times 2^0$. The rounding result x'' depends on d_1: $x'' = (-1)^S \times 2^0$ if $d_1 = 0$ or $x'' = (-1)^S \times 2^1$ if $d_1 = 1$. Therefore, we have $x'' = (-1)^S \left(1.\overline{0}\right)_2 \times 2^{d_1}$.

Case 3: $y \in \{1, \dots, 51\}$
We represent x' by right-shifting the radix point y times,and removing the biased exponent y:

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

Note that bits $\left(1d_1 \dots d_y\right)_2$ are the integer part and bits $\left(.d_{y+1} \dots d_{52}\right)_2$ are the fractional part. We have $\left(.d_{y+1}\right)_2 = 0.5$ when $d_{y+1} = 1$, and $\left(.d_{y+1}\right)_2 = 0$ when $d_{y+1} = 0$. Therefore, rounding x' to the nearest integer means rounding up if $d_{y+1} = 1$, or keeping the integer part unchanged if $d_{y+1} = 0$. In both cases, all bits in the fractional part are set to zeros. An edge case is when $(d_i)_{i\in[y]} = 1$ and $d_{y+1} = 1$, as $(d_i)_{i\in[y]} = 0$ after rounding up. Therefore, we have to round x' by increasing the exponent y by one and setting all bits of the significant to zero.

In summary we have three subcases (Case 3a, Case 3b, Case 3c) that are summarized below:

$$x'' = \begin{cases} (-1)^S \left(1.d'_1 \dots d'_y \overline{0}\right)_2 \times 2^y, & \text{if } d_{y+1} = 1 \text{ and for } i \in [y],\ \exists i : d_i = 0 \\ (-1)^S \left(1.\overline{0}\right)_2 \times 2^{y+1}, & \text{if } d_{y+1} = 1 \text{ and for } i \in [y],\ \forall i : d_i = 1 \\ (-1)^S \left(1.d_1 \dots d_y \overline{0}\right)_2 \times 2^y, & \text{if } d_{y+1} = 0 \end{cases} \tag{3.2}$$

where $\left(d'_1 \dots d'_y\right)_2 = \left(d_1 \dots d_y\right)_2 + 1$.

Case 4: $y = -1$
When $y = -1$, we have $x' = (-1)^S(0.1d_1d_2\dots d_{51})_2$. Since the digit after the radix point is always 1, x' is round to $(-1)^S \times 2^0$. Therefore, x'' can be represented in IEEE 754 floating-point (cf. § 2.3.1) as follows:

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

Case 5: $y < -1$
When $y < -1$, we have $x' = (-1)^S(0.01d_1d_2\dots d_{50})_2$. Since the digit after the radix point is 0, x' is always rounded to 0. We set $x'' \leftarrow \pm 0$ which can be represented in IEEE 754 floating-point (cf. § 2.3.1) as follows:

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

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

3.2 Integer-Scaling Mechanism

In this section, we introduce the differentially private mechanism proposed in [Tea20].

Mironov [Mir12] showed that the implementation of Laplace mechanism (cf. 2.3.16) with textbook noise generation under floating-point arithmetic can lead to a violation of the DP guarantee. In [Tea20], a team from Goolgle presents a secure differentially private mechanism that address the problem using scaled integers to simulate continuous noise, which is defined as follow:

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

where i is an integer sampled from a discrete probability distribution that is appropriately chosen and scaled by resolution parameter $r2^k$ to simulate continuous noise for $k \in [-1074, 971]$. $f_r\left(D\right) \in \mathbb{D}$ is calculated by rounding the output of query function $f\left(D\right) \in \mathbb{R}$ to the nearest multiple of r.

The key challenge thereby is to calculate $M_{IS}\left(f\left(D\right), r\right)$ under floating-point arithmetic *precisely* as real numbers. *Precisely* indicates to represent real numbers as floating numbers without precision loss, and the arithmetic operations of those real numbers yield the same result as if these real numbers are represented as floating-point numbers. In this way, $M_{IS}\left(f\left(D\right), r\right)$ is immune to the attack described in [Mir12], that is caused by the porous distribution and rounding effect of floating-point arithmetic.

3.2.1 Approximating Laplace Mechanism

In this section, we describe how the integer-scaling Laplace mechanism $M_{ISLap}\left(f\left(D\right), r, \varepsilon, \Delta_r\right) = f_r\left(D\right) + ir$ [Tea20] approximates Laplace mechanism, where integer i is sampled from a two-side geometric distribution (cf. 2.3.8).

Since the two-side geometric distribution is a discrete variate of the Laplace distribution, it is natural to use a scaled two-side geometric random variable ir to simulate the Laplace random variable Y. Specifically, integer i is sampled from a two-side geometric distribution $DGeo\left(p = 1 - e^{-\lambda}\right)$, where $\lambda = \frac{r\varepsilon}{\Delta_r}$. r is the smallest power of 2 exceeding $\frac{\Delta}{2^c\varepsilon}$, ε controls the differential privacy protection level (cf. 2.3.16), and c is a predefined parameter that controls the degree of discretization and accuracy of $M_{ISLap}\left(f\left(D\right), r, \varepsilon, \Delta_r\right)$.

TODO: explain, how to choose c in practice, influence on accuracy...

Theorem 7 ([Tea20]). *The integer-scaling Laplace mechanism $M_{ISLap}\left(f\left(D\right), r, \varepsilon, \Delta_r\right)$ satisfies ε-DP for query function f.*

In the following part, we first introduce the sampling algorithm for geometric random variables and two-side geometric random variables based on work [Tea20].

Geometric Distribution Sampling

Algorithm 3.2 [Tea20] samples a positive integer x from a geometric distribution $Geo\left(p = 1 - e^{-\lambda}\right)$ (cf. 2.3.7).

Before introducing the details about Algorithm 3.2, let's first define the sampling interval of geometric random variable x and value range for λ.

We choose $x \in [1, Int_{max53} - 1]$, where $Int_{max53} - 1 = \left(\overline{1}_{(53)}\right)_2 - 1 = 9.007199254740991 \times 10^{15}$. In original work [Tea20], the sampling interval for x is $\left[1, Int_{max64}\right]$, where $Int_{max64} = \left(\overline{1}_{(63)}\right)_2 = 9.223372036854775807 \times 10^{18}$. We limit the sampling interval to $[1, Int_{max53} - 1]$ such that xr can be calculate precisely under floating-point number without additional operations as follows:

$$\begin{aligned} xr &= x \cdot 2^k \\ &= (x_1 \dots x_{53})_2 \cdot 2^k \\ &= (1.x_2 x_3 \dots x_{53})_2 \cdot 2^{k+52}, \end{aligned} \tag{3.3}$$

where $x = (1.x_2 x_3 \dots x_{53})_2$, $x_1 = 1$ and $x_i \in \{0, 1\}$ for $i \in [2, 53]$. As $k \in [-1074, 971]$, we have $k + 52 \in [-1022, 1023]$ which indicates that $xr = (1.x_2 x_3 \dots x_{53})_2 \cdot 2^{k+52}$ is a valid IEEE 754 double-precision floating-point number (cf. § 2.3.1) if $x \in [1, Int_{max53} - 1]$.

Then, we set a reasonable value range for parameter λ (original work [Tea20] requires $\lambda > 2^{-59}$). For the geometric distribution's CDF ($\Pr(x \leq X) = 1 - (1-p)^X$ with $X = Int_{max53} - 1$), we have

$$\text{CDF} : \Pr(x \leq Int_{max53} - 1) = 1 - e^{-\lambda(Int_{max52} - 1)} = \begin{cases} 0.\overline{9}_{(13)}8834\dots & \text{for } \lambda = 2^{-48} \\ 0.\overline{9}_{(6)}8875\dots & \text{for } \lambda = 2^{-49} \end{cases} \tag{3.4}$$

The support of the geometric distribution $Geo(p)$ is $supp(Geo(p)) = \{z \in \mathbb{Z} \mid z > 0\}$. However, since we can only sample integers from interval $[1, Int_{max52} - 1]$ as discussed, $\Pr(x \leq Int_{max53} - 1)$ should be close to 1. THus, we set $\lambda \geq 2^{-49}$ to ensure $\Pr(x \leq Int_{max53} - 1) \geq 0.\overline{9}_{(6)}8875\dots$.

Roughly speaking, Algorithm 3.2 samples an integer $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$ by splitting the sampling interval into two almost even subintervals, then one subinterval is chosen at random and the other is discard. This splitting and choosing process is repeated until the remaining interval only contains one value which is the the geometric random variable x. In original work [Tea20], the sampling algorithm additional checks if the generated random variable

exceed the sampling interval $\left[0, Int_{64}\right]$, that is not necessary in our case because we always generate value in the interval $[1, Int_{53}-1]$.

In Line 1, we set the initial sampling interval to $[1, Int_{53}-1]$. In Line $2-7$, the interval $(L \ldots R]$ is first splitted with $\text{Split}(L,R,\lambda)$ into two subintervals (i.e., $(L \ldots M]$ and $(M \ldots R]$), such that they have approximately equal probability ($\approx \frac{1}{2}$). Function $\text{Split}(L,R,\lambda) = L - \text{int}\left(\frac{\ln(0.5)+\ln\left(1+e^{-\lambda(R-L)}\right)}{\lambda}\right)$ calculates the middle point M of interval $(L \ldots R]$ such that for $Geo\left(p = 1-e^{-\lambda}\right)$'s PMF

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

Line $4-7$ ensure that the middle point M lies in interval $(L \ldots R]$ (relocates M to interval $(L \ldots R]$). Line 8 calculates the proportion of probability mass 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^* and Q. Specifically, suppose $\Pr(L < x \leq M) = l$ and $\Pr(M < x \leq R) = r$, we have $Q = \frac{l}{l+r}$. If $U^* \leq Q$, we choose $(L \ldots M]$. Otherwise, we choose $(M \ldots R]$. In other words, the probability that an interval is chosen is proportional to its probability mass.

The whole process is repeated until the remaining interval contains only one value (cf. Line 2). Finally, we set $x \leftarrow R$, where $x \sim Geo\left(p = 1-e^{-\lambda}\right)$.

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

Input: λ

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

```
1 :  L ← 0, R ← Int_{max53} − 1
2 :  FOR L + 1 < R
3 :      M ← Split(L, R, λ)
4 :      IF M ≤ L
5 :          M = L + 1
6 :      ELSE IF M ≥ R
7 :          M = R − 1
8 :      Q = Proportion(L, R, M, λ)
9 :      U* ← Algo^{RandFloat1}
10 :     IF U* ≤ Q
11 :         R ← M
12 :     ELSE
13 :         L ← M
14 :  RETURN x ← R
```

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

Two-side geo Geometric Distribution Sampling

In this part, we describe how to sample $i \sim DGeo\left(p = 1 - e^{-\lambda}\right)$ based on the work by [Tea20].

Recall that $DGeo(p)$ can be generated by left-shifting geometric distribution $Geo(p)$ along the x-axis by 1, mirroring along the y-axis and scale it such that for $DGeo(p)$'s PMF: $\Pr(-\infty < i < +\infty \mid p) = 1$ (cf. 2.3.8).

As Algorithm 3.3 [Tea20] shows, we generate a two-side geometric random variable $i = s \cdot g$ by initializing its sign $s \leftarrow -1$ and $g \leftarrow 0$. Then, in the **WHILE** loop of Line $2-4$, we keep regenerating the sign s and the number part g as long as $s == -1 \wedge g == 0$. In Line 3, $Algo^{GeometricExpBinarySearch}(\lambda) - 1$ is equivalent to left-shifting the geometric distribution $Geo_{left-shift}(p)$ such that it has support $supp\left(Geo_{left-shift}(p)\right) = \{z \in \mathbb{N}\}$. In Line 2, for $g == 0$, we reject the random variable $i = s \cdot g = -1 \cdot 0$ and accept only $i = s \cdot g = +1 \cdot 0$, otherwise, we would generate value $i = s \cdot g = 0$ twice.

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

Input: λ

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

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

Algorithm 3.3: Algorithm for sampling $i \sim DGeo\left(p = 1 - e^{-\lambda}\right)$.

3.2.2 Approximating Gaussian Mechanism

In this section, we describe how the Gaussian mechanism (cf. 2.3.18) can be approximated with integer-scaling Gaussian mechanism $M_{ISGauss}(f(D), r, \varepsilon, \sigma, \Delta_r) = f_r(D) + ir$ [Tea20], where i is sampled from a symmetrical binomial distribution.

Recall that $SymmBino(n, p = 0.5) = Bino(n, p = 0.5) - \frac{n}{2}$ (cf. 2.3.6). We first generate a binomial random variable, then, substract it by np to get a symmetrical binomial random variable. It can be proven that the closeness between a symmetrical binomial random variable and Guassian random variable depends on n, i.e., a larger n indicates a better approximation effect [Tea20].

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

Suppose $x \sim Bino(n_x, p)$ and $y \sim Bino(n_y, p)$. We have that $x+y \sim Bino(n_x + n_y, p)$ [Dev86, Lemma 4.3]. Note that x and y can be generated by independently flipping a coin for n_x and n_y times and counting the numbers of $head$. However, for a very large n (e.g., $n \approx 2^{96}$), this combination method is infeasible. Therefore, a team from Google [Tea20] proposes a rejection based binomial sampling algorithm (cf. Algorithm 3.4) that is efficient for large n.

Algorithm 3.4 samples $i \sim SymmBino(n, p = 0.5)$ with input $\sqrt{n}$ for very large n, e.g., $n = 2^{96}$. Note that even for $n = 2^{96}$, $\sqrt{n} = 2^{48}$ can still be precisely represented as a 64-bit integer under IEEE 754 floating-point (cf. § 2.3.1) (see Algorithm 2.2, Algorithm 3.1), Algorithm A.4) and Algorithm 2.1 for details).

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

Input: $n \approx 2^{48}$
Output: $i \sim SymmBino(n, p = 0.5)$
1 : $m \leftarrow \lfloor \sqrt{2} * \sqrt{n} + 1 \rfloor$
2 : **WHILE TRUE**
3 : $s \leftarrow Algo^{Geometric}$
4 : $U^* \leftarrow Algo^{RandFloat1}$
5 : **IF** $U^* < 0.5$
6 : $k \leftarrow s$
7 : **ELSE**
8 : $k \leftarrow -s - 1$
9 : $l \leftarrow Algo^{RandInt}(m)$
10 : $x \leftarrow km + l$
11 : **IF** $-\frac{\sqrt{n \ln n}}{2} \leq x \leq \frac{\sqrt{n \ln n}}{2}$
12 : $\tilde{p}(x) = \sqrt{\frac{2}{\pi n}} \cdot e^{-\frac{2x^2}{n}} \cdot (1 - \nu_n)$, where $\nu_n = \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}$
13 : **IF** $\tilde{p}(x) > 0$
14 : $f \leftarrow \frac{4}{m \cdot 2^s}$
15 : $c \leftarrow Algo^{Bernoulli}\left(\frac{\tilde{p}(x)}{f}\right)$
16 : **IF** $c == 0$
17 : **RETURN** $i \leftarrow x$

Algorithm 3.4: Algorithm for sampling $i \sim SymmBino\left(n \approx 2^{96}, p = 0.5\right)$.

3.3 Discrete Gaussian Mechanism

In this section, we introduce the discrete Gaussian mechanism proposed by Canonne et al. [CKS20] that is defined as follows:

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

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

We briefly introduce the two major algorithms for discrete Gaussian mechanism proposed by Canonne et al. [CKS20] : Algorithm 3.6 and Algorithm 3.5 (see Algorithm 2.1, Algorithm A.3), and Algorithm A.1 for details).

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

Input: n **and** d

Output: $x \sim DLap\left(\frac{n}{d}\right)$

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

Algorithm 3.5: Algorithm for sampling $x \sim DLap\left(\frac{n}{d}\right)$.

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

Input: σ

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

1 : $t \leftarrow \lfloor \sigma \rfloor + 1$
2 : **WHILE TRUE**
3 : $L \leftarrow Algo^{DiscreteLap}(t, 1)$
4 : $B \leftarrow Algo^{BernoulliEXP}\left(\frac{\left(|L| - \frac{\sigma^2}{t}\right)^2}{2\sigma^2}\right)$
5 : **IF** $B == 1$
6 : **RETURN** $Y \leftarrow L$

Algorithm 3.6: Algorithm for sampling $Y \sim DGauss(\mu = 0, \sigma)$.

4 MPC Protocols for Secure Differentially Private Mechanisms

In this chapter, we first define the investigated research problem formally and introduce our MPC-DP procedure for combining MPC protocols and secure differentially private mechanisms. Then, we construct the MPC protocols for the secure differentially private mechanisms (cf. § 3) integrate them into our MPC-DP procedure (cf. **??**). In the MPC-DP procedure, we assume that the parties has already computed $\langle f(D) \rangle$ and focus on the distributed noise generation and output perturbation.

4.1 General Procedure for Combining MPC and DP

Let dataset $D = (D_1, \ldots, D_n)$ be distributed among n parties $(P_i)_{i \in [n]}$ who do not trust each other and party P_i owns data D_i. The parties $(P_i)_{i \in [n]}$ wish to jointly compute a public function f on their private inputs $(pre(D_i))_{i \in [n]}$. After the computation, either all parties or designated parties receive the result of $f(pre(D_1), \ldots, pre(D_n))$ as output, where pre is a pre-processing function executed on their local data $(D_i)_{i \in [n]}$. The parties want to achieve (1) computional privacy (i.e., each party's input is kept secret), (2) output privacy (i.e., the output satisfies the DP), and (3) obtain a result with reasonable accuracy.

To achieve computional privacy, the parties deploy a MPC protocol Π^f which takes $(pre(D_i))_{i \in [n]}$ as input and reveals only the computation result. For requirements (2) and (3), we design a MPC-DP procedure which combines Π^f with a secure differentially private mechanism. Note that in following, we omit the data pre-processing procedure $pre(\cdot)$ for simplicity and use $f(D)$ to represent $f(D_1, \ldots, D_n)$.

Before describing our MPC-DP protocol, let's consider a trival example that combines MPC protocols with a DP mechanism. Suppose the parties $(P_i)_{i \in [n]}$ wish to calculate the sum of their local data $(D_i)_{i \in [n]}$ with a public known function $f(D_1, \ldots, D_n) = \sum_{j=1}^{n} D_j$, and f has ℓ_2-*sensitivity* $\Delta_2^{(f)} = 1$ (cf. 2.3.15). Meanwhile, the parties require that only the computation result of f should be revealed and differential privacy must be satisfied. To achieve this, each party first adds noise r_i to its local data D_i and determines $y_i = D_i + r_i$. Recall that in 2.3.18, we have showed that $y_i =$ satisfies (ε, δ)-DP for $r_i \sim \mathcal{N}\left(0, \sigma^2\right)$, where $\sigma^2 = \frac{2}{\varepsilon^2} \cdot \ln\left(\frac{1.25}{\delta}\right)$ and $i \in [n]$. Then, the parties jointly run the MPC protocol Π^f with their private inputs y_i to compute function f. After reconstruction they get:

$$\begin{aligned} y &= \sum_{j=1}^{n} y_j \\ &= \sum_{j=1}^{n} \left(D_j + r_j \right). \end{aligned} \tag{4.1}$$

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

In general, the parties in above example can achieve $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-DP assuming no corrupted parties. However, in realistic scenarios, corrupted parties may try to extract information of certain parties by weaking the achieved differential privacy protection level. After executing aforementioned steps honestly and obtaining the computation result y, the corrupted parties subtract the noise they have added from y. In the worst case, i.e., $n-1$ parties are corrupted, the differential privacy guarantee reduces from $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-DP to (ε, δ)-DP. If, however, having $\left(\frac{\varepsilon}{\sqrt{n}}, \delta\right)$-DP is required, a potential naive solution is that each party adds *more* noise, .e.g, $r_i' \sim \mathcal{N}\left(0, n \cdot \sigma^2\right)$, locally before running Π^f. However, this can lead to a severely reduced accuracy.

To summarize, a well-designed MPC-DP protocol requires a more careful integration of MPC protocols and DP mechanisms. To guarantee the both accuracy and privacy of the MPC-DP protocol, we define several requirements, that is based on the "ideal scenario" with a centralized data server, i.e., a trusted server that has access to all the parties' data D locally computes $f(D)$ before adding an essential amount of noise to guarantee DP:

1. The MPC-DP protocol should achieve the same privacy protection level as in the centralized data server scenario, and the amount of noise r (in above example: $r = \sum_{j=1}^{n} r_j$) should match that of the centralized data server scenario.
2. The differential privacy guarantee of the MPC-DP protocol should not degenerate in the presence of corrupted parties.

MPC-Scenario The MPC-DP procedure can be used among n parties, or in an outsourcing scenario, i.e., an arbitrary amount of parties secret share their private inputs to N ($N \ll n$) non-colluding computing parties. Then, the computing parties execute the N-party MPC-DP protocol, where they compute f and add noise shares to get the noisy secret-shared results. Finally, the noisy secret-shared results are sent back to the parties for reconstruction. We assume semi-honest adversaries that follow the protocol specifications but wish to infer additional information[EKR17, Chapter 2].

Prot. 4.1 describes the general procedure of our MPC-DP protocol. Note that the parties first calculate $\langle f(D) \rangle = \Pi^f\left(\langle D_1 \rangle, \ldots, \langle D_n \rangle\right)$ and add the distributed generated noise $\langle r \rangle$ (output

perturbation) to guarantee differential privacy. In this work, we assume that each party has computed $\langle f(D)\rangle$ and focus on the process of Π^{DNG}, i.e., distributed noise generation (DNG).

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

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

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

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

2 : $\langle r\rangle \leftarrow \Pi^{DNG}$

3 : $\langle M(D)\rangle \leftarrow \Pi^{Perturb}(\langle f(D)\rangle, \langle r\rangle)$

Protocol 4.1: General framework for MPC-DP protocol.

4.2 Building Blocks

Our MPC protocols for differentially private mechanisms relies on the following building blocks:

1. $\langle \boldsymbol{y}\rangle^B \leftarrow \Pi^{RandBits}(\ell)$ generates a share of a public unknown ℓ-bit random string $\langle \boldsymbol{y}\rangle^B = \left(\langle y_0\rangle^B, \ldots, \langle y_{\ell-1}\rangle^B\right)$ without communications between parties. To achieve this, each party locally generate a random ℓ-bit string $\boldsymbol{x} \in \{0,1\}^\ell$ and set $\langle \boldsymbol{y}\rangle^B = \boldsymbol{x}$.
2. $\langle \boldsymbol{y}\rangle^B \leftarrow \Pi^{PreOr}\left(\langle \boldsymbol{x}\rangle^B\right)$ calculates the prefix-OR of a l-bit string $\langle \boldsymbol{x}\rangle^B = \left(\langle x_0\rangle^B, \ldots, \langle x_{l-1}\rangle^B\right)$ and output $\langle \boldsymbol{y}\rangle^B = \left(\langle y_0\rangle^B, \ldots, \langle y_{l-1}\rangle^B\right)$ such that $\left\langle y_j\right\rangle^B = \vee_{k=0}^{j} \langle x_k\rangle^B$ for $j \in [0, l)$, where $\langle y_0\rangle^B = \langle x_0\rangle^B$.
3. $\langle \boldsymbol{y}\rangle^{B,UI} \leftarrow \Pi^{Geometric}$ (cf. § A.2.2) generates a share of geometric random variable $y \sim Geo(p = 0.5)$ (cf. 2.3.7).
4. $\langle \boldsymbol{y}\rangle^B \leftarrow \Pi^{OTMult}\left(\langle \boldsymbol{x}\rangle^B, \langle b\rangle^B\right)$ computes $\langle \boldsymbol{y}\rangle^B = \langle \boldsymbol{x}\rangle^B \cdot \langle b\rangle^B$, which is inspired by the OT-based multiplication algorithms [AHLR18; ST19]. For example, in two-party computation, $\langle \boldsymbol{x}\rangle^B \cdot \langle b\rangle^B$ can be reformulated as follows:

$$\begin{aligned}\langle \boldsymbol{x}\rangle^B \cdot \langle b\rangle^B &= \left(\langle \boldsymbol{x}\rangle_0^B \oplus \langle \boldsymbol{x}\rangle_1^B\right) \cdot \left(\langle b\rangle_0^B \oplus \langle b\rangle_1^B\right) \\ &= \langle \boldsymbol{x}\rangle_0^B \cdot \langle b\rangle_0^B \oplus \langle \boldsymbol{x}\rangle_0^B \cdot \langle b\rangle_1^B \oplus \langle \boldsymbol{x}\rangle_1^B \cdot \langle b\rangle_0^B \oplus \langle \boldsymbol{x}\rangle_1^B \cdot \langle b\rangle_1^B,\end{aligned} \tag{4.2}$$

   where $\langle \boldsymbol{x}\rangle_0^B \cdot \langle b\rangle_0^B$ and $\langle \boldsymbol{x}\rangle_1^B \cdot \langle b\rangle_1^B$ can be calculated by each party locally. For the remaining two mulitplications, we utilize two C-OTs (cf. § 2.2.1). For $i \in \{0,1\}$, P_i inputs $\langle b\rangle_i^B$, and P_{1-i} inputs $\left(r_i, r_i \oplus \langle \boldsymbol{x}\rangle_{1-i}^B\right)$ and receives $r_i \oplus \left(\langle b\rangle_i^B \cdot \langle \boldsymbol{x}\rangle_{1-i}^B\right)$, where r_i is a random bit string. Finally, P_i set $r_i \oplus r_{1-i} \oplus \left(\langle b\rangle_{1-i}^B \cdot \langle \boldsymbol{x}\rangle_i^B\right)$ as its share of $\langle \boldsymbol{x}\rangle_0^B \cdot \langle b\rangle_1^B \oplus \langle \boldsymbol{x}\rangle_1^B \cdot \langle b\rangle_0^B$. We also extend Π^{OTMult} to multiple parties using similar C-OT based

approach. We set Π^{OTMult} as default MPC protocol to compute bit-vector multiplication. TODO: check if MOTION has implemented bit-vector multiplication

5. $\left(\langle y_1\rangle^B, \ldots, \langle y_\ell\rangle^B\right) \leftarrow \Pi^{Binary2Unary}\left(\langle \boldsymbol{x}\rangle^{B,UI}, \ell\right)$ [ABZS12] converts 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 others to 0. We use HyCC compiler (cf. § 2.2.6) to generate both the size-optimal and depth-optimal circuit for AND gate.
6. For signed (SI) and unsigned integer (UI) operations (e.g., comparision, addition, substraction), we use depth-optimized circuits generated with HyCC (cf. § 2.2.6).
7. For floating-point operations, we use the Boolean circuits of work [DSZ15].
8. $\langle \boldsymbol{y}\rangle^{B,UI} \leftarrow \Pi^{HW}(x_0, \ldots, x_\ell)$ counts the number of ones in string $x_0, \ldots, x_\ell$ using the algorithm proposed by Boyar and Peralta [BP08].
9. $\langle \boldsymbol{y_i}\rangle^{B,UI} \leftarrow \Pi^{OAA}\left(\langle \boldsymbol{y_0}\rangle^{B,UI}, \ldots, \langle \boldsymbol{y_{\ell-1}}\rangle^{B,UI}, \langle f\, g_0\rangle^B, \ldots, \langle f\, g_{\ell-1}\rangle^B\right)$ outputs y_i where i is the first index such that $f\, g_i == 1$ for $i \in [0, \ell-1]$. We provide various methods realizing Π^{OAA} in § A.2.1.
10. $\langle y\rangle^{B,UI} \leftarrow \Pi^{RandInt}(m)$ (cf. Prot. A.7) generate an unsigned integer $y \in [0, m)$.

4.3 MPC Protocols for Snapping Mechanism

remove duplicate content

We describe the MPC protocols based on the snapping mechanism implementations (cf. **??**).

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

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

According to our MPC-DP procedure (cf. **??**), the parties generate a noise share $\langle r\rangle$ and add it to $\langle f(D)\rangle$ to achieve differential privacy (cf. § 2.3). The snapping mechanism is reformulated in secret shares as follows:

$$\langle M_S(f(D), \lambda, B)\rangle = \text{clamp}_B\left(\left\lfloor \text{clamp}_B(\langle f(D)\rangle) \oplus \langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)\right\rceil_\Lambda\right), \tag{4.3}$$

where $\langle f(D)\rangle$ is assumed to be already computed and $\langle S\rangle \otimes \lambda \otimes \text{LN}(\langle U^*\rangle)$ is the noise share, i.e., the parties generate the noise starting from the generation of random variable $S \in \{-1, 1\}$ and $U^* \in \mathbb{D} \cap (0, 1)$. Finally, the parties compute $\lfloor\cdot\rceil_\Lambda$ and $\text{clamp}_B(\cdot)$.

In our MPC protocols, we mainly operate on IEEE 754 floating-point (cf. § 2.3.1): $\langle \boldsymbol{x}\rangle^{B,FL} = (\langle S\rangle, \langle d_1\rangle, \ldots, \langle d_{52}\rangle, \langle e_1\rangle, \ldots, \langle e_{11}\rangle)$, where $x = (-1)^S (1.d_1 \ldots d_{52})_2 \times 2^{(e_1 \ldots e_{11})_2 - 1023}$.

4.3.1 Generation of U^* and S

Each party can generate sign $\langle S \rangle^B$ by sampling a random bit locally.

As discussed in § 3.1, U^* is the uniform distribution over $\mathbb{D} \cap (0,1)$, which can be represented in IEEE-754 double-precision floating-point (cf. § 2.3.1) as follows:

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

where significant bits $(d_1, \dots, d_{52})$ are sampled uniformly from $\{0,1\}^{52}$ and biased exponent $x = e - 1023$ is sampled from a geometric distribution (cf. 2.3.7).

As Prot. 4.2 shows, we first generate significand bits share $\langle d_1 \rangle^B, \dots, \langle d_{52} \rangle^B$ with $\Pi^{RandBits}(52)$ (cf. § 4.2), then generate a geometric random variable share $\langle \boldsymbol{x} \rangle^{B,UI}$ with $\Pi^{Geometric}$ (cf. § A.2.2) to build the biased exponent $\langle \boldsymbol{x} \rangle^{B,UI}$, where $x = e - 1023$. Finally, the parties compute and obtain shares of the floating-point number $U^* = (1.d_1 \dots d_{52})_2 \times 2^{e-1023}$.

Protocol: $\Pi^{RandFloat1}$

Input: None

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

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

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

3 : $\langle \boldsymbol{e} \rangle^{B,UI} \leftarrow 1023 + \langle \boldsymbol{x} \rangle^{B,UI}$

Protocol 4.2: MPC protocol for uniform random variable $U^* \in \mathbb{D} \cap (0,1)$.

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

Function $\text{clamp}_B(x)$ (cf. **??**) outputs B if $x > B$, $-B$ if $x < -B$ and x otherwise.

$\Pi^{Clamp}\left(\langle \boldsymbol{x} \rangle^{B,FL}, B\right)$ (cf. Prot. 4.3) first identifies the location of x in line $1-3$, and output result based on its conditions.

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

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

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

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

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

3 : $\left\langle cond_{-B\leq x\leq B} \right\rangle^{B} \leftarrow \mathrm{NOT}\left(\langle cond_{x<-B} \rangle^{B} \vee \langle cond_{x>B} \rangle^{B}\right)$.

4 : $\langle \boldsymbol{a} \rangle^{B,FL} \leftarrow \langle cond_{x<-B} \rangle^{B} \cdot (-B) \oplus \langle cond_{x>B} \rangle^{B} \cdot B$.

5 : $\langle \boldsymbol{b} \rangle^{B,FL} \leftarrow \Pi^{OTMult}\left(\langle \boldsymbol{x} \rangle^{B,FL}, \left\langle cond_{-B\leq x\leq B} \right\rangle^{B}\right)$.

6 : $\left\langle \boldsymbol{x_{clampB}} \right\rangle^{B,FL} \leftarrow \langle \boldsymbol{a} \rangle^{B,FL} \oplus \langle \boldsymbol{b} \rangle^{B,FL}$.

Protocol 4.3: MPC protocol for $\mathrm{clamp}_B(\cdot)$.

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

Prot. 4.4 rounds x to the nearest multiply of $\Lambda = 2^n$. As discussed in § 3.1, $\lfloor x \rceil_\Lambda$ with input x can be calculated in three steps:

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

In line 1, the parties extended biased exponent to $(0, \langle e_1 \rangle, \ldots, \langle e_{11} \rangle)$ such that $\langle \boldsymbol{e} \rangle^{B,UI}$ has enough bits to represent a 12-bit signed integer. In line 2, the parties convert biased exponent $(e_1, \ldots, e_{11})_2$ to signed integer because $(e_1, \ldots, e_{11})_2 - n - 1023$ (line 3) can be smaller than 0 but unsigned integer cannot represent a negative value. In line 3, 4, the parties compute $(e_1, \ldots, e_{11})_2 - n - 1023$ that is equivalent to $x' = \frac{x}{\Lambda}$. In line 5, 6, 8, 16, 17, 19, 20, 22, we compute the conditions for five cases as discussed in paragraph 3.1. In line 9, the parties run $\Pi^{Binary2Unary}\left(\langle \boldsymbol{y} \rangle^{B,UI}, 52\right)$ to get $p_1, \ldots, p_{52}$, where $p_j = 1$ for $j \in [y]$ and other bits equal to zero. In line 10, the parties extract the first y bits from significant field $(d_1, \ldots, d_{52})$, set the rest bits to zero, and obtain $\left(d_1, \ldots, d_y, \overline{0}_{52-y}\right)$. In line $11-13$, the parties extract d_{y+1} by setting all bits in bit string c to zero except $c_{y+1} = 1$. In line 14, the parties verify if $\left(d_1, \ldots, d_y\right) = (0, \ldots, 0)$, and set is as the subcondition for Case 3. In line 15, the parties calculate the new significant bits for Case 3a. In line 18, the parties compute the biased exponent for Case 3b. In line 21, the parties set the biased exponent for Case 4.

In line 23, the parties compute the significant bits $\langle \boldsymbol{t} \rangle^{B,UI}$ and biased exponent $\langle \boldsymbol{m} \rangle^{B,UI}$ with Π^{OTMult} (cf. § 4.2) as follows:

$$\begin{aligned}\langle \boldsymbol{t} \rangle^{B,UI} = \quad & \langle cond_{case1} \rangle^{B} \cdot \left(\langle d_1 \rangle^{B}, \ldots, \langle d_{52} \rangle^{B} \right) \\ & \oplus \langle cond_{case3a} \rangle^{B} \cdot \langle \boldsymbol{d_{case3a}} \rangle^{B,UI} \\ & \oplus \langle cond_{case3c} \rangle^{B} \cdot \langle \boldsymbol{d_{case3c}} \rangle^{B,UI},\end{aligned} \tag{4.4}$$

$$\begin{aligned}\langle \boldsymbol{m} \rangle^{B,UI} = \quad & \left(\langle cond_{case1} \rangle^{B} \oplus \langle cond_{case3a} \rangle^{B} \oplus \langle cond_{case3c} \rangle^{B} \right) \cdot \left(\langle e_1 \rangle^{B}, \ldots, \langle e_{11} \rangle^{B} \right) \\ & \oplus \langle cond_{case2} \rangle^{B} \cdot \langle \boldsymbol{m_{case2}} \rangle^{B,UI} \\ & \oplus \langle cond_{case3b} \rangle^{B} \cdot \langle \boldsymbol{m_{case3b}} \rangle^{B,UI} \\ & \oplus \left\langle cond_{case4} \right\rangle^{B} \cdot \left\langle \boldsymbol{m_{case4}} \right\rangle^{B,UI},\end{aligned} \tag{4.5}$$

Note that Eq. (4.4) ignore Case 2, Case 3b, Case 4 and Case 5, because the significant bits of those cases are all zeros. Similarly, we ignore Case 5 in Eq. (4.5).

In line 24, 25, the parties first check if x'' equals to zero, output $\lfloor x \rceil_{\Lambda} = \Lambda \cdot x''$ if not, x'' otherwise.

Protocol: $\Pi^{Round2Lambda}\left(\langle \boldsymbol{x} \rangle^{B,FL}, n\right)$

Input: $\langle \boldsymbol{x} \rangle^{B,FL} = \left(\langle S \rangle^{B}, \langle d_1 \rangle^{B}, \ldots, \langle d_{52} \rangle^{B}, \langle e_1 \rangle^{B}, \ldots, \langle e_{11} \rangle^{B}\right)$, n, **where** $\Lambda = 2^n$

Output: $\langle \lfloor \boldsymbol{x} \rceil_{\Lambda} \rangle^{B,FL} = \left(\langle S \rangle^{B}, \langle \boldsymbol{t} \rangle^{B,UI}, \langle \boldsymbol{m} \rangle^{B,UI}\right)$, **where** $\langle \boldsymbol{t} \rangle^{B,UI} = \left(\langle t_1 \rangle^{B}, \ldots, \langle t_{11} \rangle^{B}\right)$ **and** $\langle \boldsymbol{m} \rangle^{B,UI} = \left(\langle m_1 \rangle^{B}, \ldots, \langle m_{11}$

1 : $\langle \boldsymbol{e} \rangle^{B,UI} \leftarrow \left(0, \langle e_1 \rangle^{B}, \ldots, \langle e_{11} \rangle^{B}\right)$

2 : $\langle \boldsymbol{e} \rangle^{B,SI} \leftarrow \mathrm{UI2SI}\left(\langle \boldsymbol{e} \rangle^{B,UI}\right)$

3 : $\langle \boldsymbol{y} \rangle^{B,UI} \leftarrow \langle \boldsymbol{e} \rangle^{B,UI} - n - 1023$

4 : $\langle \boldsymbol{y} \rangle^{B,SI} \leftarrow \langle \boldsymbol{e} \rangle^{B,SI} - n - 1023 \qquad //x' = \dfrac{x}{\Lambda}$

5 : $\langle cond_{case1} \rangle^{B} \leftarrow \left(\langle \boldsymbol{y} \rangle^{B,SI} > 51\right)$ // Case 1: $y \geq 52$

6 : $\langle cond_{case2} \rangle^{B} \leftarrow \left(\langle \boldsymbol{y} \rangle^{B,SI} == 0\right)$ // Case 2: y==0

7 : $\langle \boldsymbol{m}_{\boldsymbol{case2}} \rangle^{B,UI} \leftarrow \mathrm{NOT}\left(\langle d_1 \rangle^{B}\right) \cdot 1023 + \langle d_1 \rangle^{B} \cdot 1024$ // Case 2: $m = 1023 + d_1$

8 : $\langle cond_{case3} \rangle^{B} \leftarrow \left(\langle \boldsymbol{y} \rangle^{B,SI} > 0\right) \wedge \mathrm{NOT}\left(\langle cond_{case1} \rangle^{B}\right)$ // Case 3: $y \in [1, 51]$

9 : $\left(\langle p_1 \rangle^{B}, \ldots, \langle p_{52} \rangle^{B}\right) \leftarrow \Pi^{Binary2Unary}\left(\langle \boldsymbol{y} \rangle^{B,UI}, 52\right)$

10 : $\langle \boldsymbol{d}_{\boldsymbol{case3c}} \rangle^{B,UI} = \left(\langle p_1 \rangle^{B} \wedge \langle d_1 \rangle^{B}, \ldots, \langle p_{52} \rangle^{B} \wedge \langle d_{52} \rangle^{B}\right)$

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

12 : $\quad \left\langle c_{j+1} \right\rangle^{B} = \left\langle p_j \right\rangle^{B} \oplus \left\langle p_{j+1} \right\rangle^{B}$, where $\langle c_1 \rangle^{B} = 0$

13 : $\left\langle d_{y+1} \right\rangle^{B} = \oplus_{j=1}^{52}\left(\left\langle c_j \right\rangle^{B} \wedge \left\langle d_j \right\rangle^{B}\right)$

14 : $\left\langle cond_{d_1 \wedge \ldots \wedge d_y == 1} \right\rangle^{B} \leftarrow \wedge_{j=1}^{52}\left(\left\langle p_j \right\rangle^{B} \wedge \left\langle d_j \right\rangle^{B}\right)$ // Case 3: For $i \in [y]$, $\forall i : d_i = 1$

15 : $\langle \boldsymbol{d}_{\boldsymbol{case3a}} \rangle^{B,UI} \leftarrow \langle \boldsymbol{d}_{\boldsymbol{case3c}} \rangle^{B,UI} + \mathrm{POW2}\left(52 - \langle \boldsymbol{y} \rangle^{B,UI}\right)$ // Case 3a: $\left(d'_1 \ldots d'_y\right)_2 = \left(d_1 \ldots d_y\right)_2 + 1$

16 : $\langle cond_{case3a} \rangle^{B} \leftarrow \left\langle d_{y+1} \right\rangle^{B} \wedge \mathrm{NOT}\left(\left\langle cond_{d_1 \wedge \ldots \wedge d_y == 1} \right\rangle^{B}\right)$

17 : $\langle cond_{case3b} \rangle^{B} \leftarrow \left\langle d_{y+1} \right\rangle^{B} \wedge \left\langle cond_{d_1 \wedge \ldots \wedge d_y == 1} \right\rangle^{B}$

18 : $\langle \boldsymbol{m}_{\boldsymbol{case3b}} \rangle^{B,UI} \leftarrow \langle \boldsymbol{y} \rangle^{B,UI} + 1 + 1023$ // Case 3b: $m = y + 1 + 1023$

19 : $\langle cond_{case3c} \rangle^{B} \leftarrow \mathrm{NOT}\left(\left\langle d_{y+1} \right\rangle^{B}\right)$

20 : $\left\langle cond_{case4} \right\rangle^{B} \leftarrow \left(\langle \boldsymbol{y} \rangle^{B,SI} == (-1)\right)$ // Case 4: $y == -1$

21 : $\left\langle \boldsymbol{m}_{\boldsymbol{case4}} \right\rangle^{B,UI} \leftarrow \left(0, \overline{1}_{(10)}\right)$ // Case 4: $m = (01111111111)_2$

22 : $\langle cond_{case5} \rangle^{B} \leftarrow \left(\langle \boldsymbol{y} \rangle^{B,SI} < (-1)\right)$ // Case 5: $y < -1$

23 : $\langle \boldsymbol{t} \rangle^{B,UI}, \langle \boldsymbol{m} \rangle^{B,UI} \leftarrow \Pi^{OTMult}(\cdot)$ // $x'' \leftarrow$ round x' to nearest integer

24 : $\left\langle cond_{x'' \neq \pm 0} \right\rangle^{B} \leftarrow \left(\vee_{j=1}^{52}\left\langle t_j \right\rangle^{B}\right) \vee \left(\vee_{j=1}^{11}\left\langle m_j \right\rangle^{B}\right)$ // check **if** $x'' = \pm 0$

25 : $\langle \boldsymbol{m} \rangle^{B,UI} \leftarrow \left\langle cond_{\neq \pm 0} \right\rangle^{B} \wedge \left(\langle \boldsymbol{m} \rangle^{B,UI} + n\right)$ // $\lfloor x \rceil_{\Lambda} = \Lambda \cdot x''$

Protocol 4.4: MPC Protocol for $\lfloor x \rceil_{\Lambda}$

4.3.4 MPC-DP Protocol for Snapping Mechanism

Prot. 4.5 integrates our MPC protocols for snapping mechanism in § 4.3 into MPC-DP procedure (cf. **??**).

$\Lambda = 2^n$ and can be calculated from the public known λ (cf. **??**) without MPC. In line 5, we directly manipulate sign bit of $\left\langle \boldsymbol{Y_{LapNoise}} \right\rangle^{B,FL}$.

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

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

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

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

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

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

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

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

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

7 : $\langle \lfloor \boldsymbol{x} \rceil_{\Lambda} \rangle^{B,FL} \leftarrow \Pi^{Round2Lambda}\left(\langle \boldsymbol{x} \rangle^{B,FL}, n\right)$.

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

Protocol 4.5: MPC-DP protocol for snapping mechanism.

4.4 MPC Protocols for Integer-scaling Mechanism

In this section, we construct the MPC protocols for integer-scaling Laplace mechanism (cf. § 3.2.1) and integer-scaling Gaussian mechanism (cf. § 3.2.2).

4.4.1 Approximating Laplacian Mechanism

Recall for integer-scaling Laplace Mechanism $M_{ISLap}(f(D), r, \varepsilon, \Delta_r) = f_r(D) + ir$ (cf. § 3.2.1), i is sampled from a two-side geometric distribution $DGeo\left(p = 1 - e^{-\lambda}\right)$. We first construct Prot. 4.6 for geometric distribution Algorithm 3.2 based on § 3.2.1.

In line $1-7$, each party locally calculates L_0, R_0, M_0 and Q_0 in plaintext. In line $8-10$, the parties generate a fixed-point random variable U_0 rescale it such that $U_0 \in [0, 1)$. In line $11-15$, the parties choose one subinterval (either $(L_0 \ldots M_0]$ or $(M_0 \ldots R_0]$) and compute flag fg_0 that indicates whether the termination condition of **WHILE** (cf. Algorithm 3.2) is

satisfied. In line $16-30$, the parties execute the binary search for $iter-1$ times. In line 21, the parties choose the correct split-point M_j using Π^{OTMult} as follows:

$$\begin{aligned}\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} = \quad & \left\langle cond_{M_j \leq L_{j-1}} \right\rangle^{B} \cdot \left(\left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI} + 1 \right) \\ & \oplus \left\langle cond_{M_j \geq R_{j-1}} \right\rangle^{B} \cdot \left(\left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI} - 1 \right) \\ & \oplus \left\langle cond_{L_{j-1} < M_j < R_{j-1}} \right\rangle^{B} \cdot \left\langle \boldsymbol{M_j} \right\rangle^{B,UI}\end{aligned} \tag{4.6}$$

In line 31, the parties extract the correct result from $\left(\langle \boldsymbol{R_0} \rangle^{B,UI}, \ldots, \left\langle \boldsymbol{R_{iter-1}} \right\rangle^{B,UI} \right)$ based on $\langle f\, g_0 \rangle^B, \ldots, \langle f\, g_{iter-1} \rangle^B$.

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

Input: λ

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

$L_0 \leftarrow 0, R_0 \leftarrow Int_{max52}$

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

IF $M_0 \leq L_0$

    $M_0 \leftarrow L_0 + 1$

ELSE IF $M_0 \geq R_0$

    $M_0 \leftarrow R_0 - 1$

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

$\langle \boldsymbol{U_0} \rangle^{B,UI} \leftarrow \Pi^{RandBits}(11)$

$\langle \boldsymbol{U_0} \rangle^{B,FP} \leftarrow \text{UI2FP}\left(\langle \boldsymbol{U_0} \rangle^{B,UI}\right)$

$\langle \boldsymbol{U_0} \rangle^{B,FP} \leftarrow \text{right-shift}\left(\langle \boldsymbol{U_0} \rangle^{B,FP}, 11\right)$

$\left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \leftarrow \left(\langle \boldsymbol{U_0} \rangle^{B,FP} \leq Q_0\right)$

$\left\langle cond_{U_0 > Q_0} \right\rangle^{B} \leftarrow \text{NOT}\left(\left\langle cond_{U_0 \leq Q_0} \right\rangle^{B}\right)$

$\langle \boldsymbol{R_0} \rangle^{B,UI} \leftarrow \left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \cdot M_0 \oplus \left\langle cond_{U_0 > Q_0} \right\rangle^{B} \cdot R_0$

$\langle \boldsymbol{L_0} \rangle^{B,UI} \leftarrow \left\langle cond_{U_0 > Q_0} \right\rangle^{B} \cdot M_0 \oplus \left\langle cond_{U_0 \leq Q_0} \right\rangle^{B} \cdot L_0$

$\langle f g_0 \rangle^{B} = \left(\langle \boldsymbol{L_0} \rangle^{B,UI} + 1 \geq \langle \boldsymbol{R_0} \rangle^{B,UI}\right)$

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

    $\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \leftarrow \left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI} - \text{FP2UI}\left(\frac{\ln(0.5) + \ln\left(1 + e^{-\lambda \cdot \text{UI2FP}\left(\left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI} - \left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI}\right)}\right)}{\lambda}\right)$

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

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

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

    $\left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \leftarrow$ Eq. (4.6)

    $\left\langle \boldsymbol{Q_j} \right\rangle^{B,FP} \leftarrow \frac{e^{-\lambda \cdot \text{UI2FP}\left(\langle \boldsymbol{M_0} \rangle^{B,UI} - \langle \boldsymbol{L_0} \rangle^{B,UI}\right)} - 1}{e^{-\lambda \cdot \text{UI2FP}\left(\langle \boldsymbol{R_0} \rangle^{B,UI} - \langle \boldsymbol{L_0} \rangle^{B,UI}\right)} - 1}$

    $\left\langle \boldsymbol{U_j} \right\rangle^{B,UI} \leftarrow \Pi^{RandBits}(11)$

    $\left\langle \boldsymbol{U_j} \right\rangle^{B,FP} \leftarrow \text{UI2FP}\left(\left\langle \boldsymbol{U_j} \right\rangle^{B,UI}\right)$

    $\left\langle \boldsymbol{U_j} \right\rangle^{B,FP} \leftarrow \text{right-shift}\left(\left\langle \boldsymbol{U_j} \right\rangle^{B,FP}, 11\right)$

    $\left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \leftarrow \left(\left\langle \boldsymbol{U_j} \right\rangle^{B,FP} \leq \left\langle \boldsymbol{Q_j} \right\rangle^{B,FP}\right)$

    $\left\langle cond_{U_j > Q_j} \right\rangle^{B} \leftarrow \text{NOT}\left(\left\langle cond_{U_j \leq Q_j} \right\rangle^{B}\right)$

    $\left\langle \boldsymbol{R_j} \right\rangle^{B,UI} \leftarrow \left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \cdot \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \oplus \left\langle cond_{U_j > Q_j} \right\rangle^{B} \cdot \left\langle \boldsymbol{R_{j-1}} \right\rangle^{B,UI}$

    $\left\langle \boldsymbol{L_j} \right\rangle^{B,UI} \leftarrow \left\langle cond_{U_j > Q_j} \right\rangle^{B} \cdot \left\langle \boldsymbol{M_j} \right\rangle^{B,UI} \oplus \left\langle cond_{U_j \leq Q_j} \right\rangle^{B} \cdot \left\langle \boldsymbol{L_{j-1}} \right\rangle^{B,UI}$

    $\left\langle f g_j \right\rangle^{B} \leftarrow \left(\left\langle \boldsymbol{L_j} \right\rangle^{B,UI} + 1 \geq \left\langle \boldsymbol{R_j} \right\rangle^{B,UI}\right)$

$\langle x \rangle^{B,UI} = \Pi^{OAA}\left(\langle \boldsymbol{R_0} \rangle^{B,UI}, \ldots, \left\langle \boldsymbol{R_{iter-1}} \right\rangle^{B,UI}, \langle f g_0 \rangle^{B}, \ldots, \langle f g_{iter-1} \rangle^{B}\right)$

Protocol 4.6: MPC protocol for geometric distribution $x \sim Geo\left(p = 1 - e^{-\lambda}\right)$.

Next, we construct Prot. 4.7) for two-side geometric distribution (cf. 2.3.8) based on Algorithm 3.3.

In line 2, 3, the parties generate the sign s_j and number part g_j of two-side geometric random variable $x = (-1)^{s_j} \cdot g_j$. In line 4, the parties compute the fg_j to record if the termination condition of **WHILE** loop (cf. Algorithm 3.3) is satisfied.

In line 5, we extract the sign s and number part g based on $\langle fg_0 \rangle^B, \ldots, \langle fg_{iter-1} \rangle^B$.

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

Input: λ

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

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

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

3 : $\quad \left\langle \boldsymbol{g_j} \right\rangle^{B,UI} \leftarrow \Pi^{GeometricExpBinarySearch}(\lambda) - 1$.

4 : $\quad \left\langle fg_j \right\rangle^B \leftarrow \mathrm{NOT}\left(\left(\left\langle s_j \right\rangle^B == 1\right) \wedge \left(\left\langle \boldsymbol{g_j} \right\rangle^{B,UI} == 0\right)\right)$

5 : $\langle s \rangle^B, \langle \boldsymbol{g} \rangle^{B,UI} \leftarrow \Pi^{OAA}\left(\langle \boldsymbol{g_0} \| s_0 \rangle^B, \ldots, \left\langle \boldsymbol{g_{iter-1}} \| s_{iter-1} \right\rangle^B, \langle fg_0 \rangle^B, \ldots, \langle fg_{iter-1} \rangle^B\right)$

Protocol 4.7: MPC Protocol for two-side geometric distribution $x \sim DGeo\left(p = 1 - e^{-\lambda}\right)$.

MPC-DP Protocol for Integer-scaling Laplace Mechanism

Recall integer-scaling Laplace mechanism $M_{ISLap}(f_r(D), r, \varepsilon, \Delta_r)$ (cf. § 3.2.1) that approximates the Laplace mechanism is defined as:

$$\begin{aligned} M_{ISLap}(f_r(D), r, \varepsilon, \Delta_r) &= f_r(D) + ir \\ &= f_r(D) + \left(2^{52} + i\right) \cdot r - 2^{52} \cdot r, \end{aligned} \tag{4.7}$$

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

In our MPC-DP general framework (cf. Prot. 4.1), it can be reformulated as:

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

where r, ϵ, Δ_r, λ are public known.

We present Prot. 4.8 for $\left\langle M_{ISLap}(f_r(D), r, \varepsilon, \Delta_r) \right\rangle$. We assume that the parties have already computed $\langle \boldsymbol{f_r(D)} \rangle^{B,FL}$ and focus on generating the integer share $\langle \boldsymbol{i} \rangle^{B,UI}$.

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

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

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

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

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

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

4 : $\left\langle \boldsymbol{M_{ISLap}}\right\rangle^{B,FL} \leftarrow \langle \boldsymbol{f_r}(\boldsymbol{D})\rangle^{B,FL} + \left\langle \boldsymbol{Y_{LapNoise}}\right\rangle^{B,FL}$.

Protocol 4.8: MPC-DP protocol for Integer-scaling Laplacian Mechanism.

4.4.2 Approximating Gaussian Mechanism

Recall for integer-scaling Gaussian Mechanism $M_{ISGauss}(f(D), r, \varepsilon, \sigma, \Delta_r) = f_r(D) + ir$ (cf. § 3.2.2), where i is sampled from a symmetric binomial distribution $SymmBino(n, p)$ (cf. 2.3.6).

We present Prot. 4.9 for symmetrical binomial distribution based on Algorithm 3.4.

Each party first locally compute following parameters:

$$\begin{aligned} m &= \left\lfloor \sqrt{2} \cdot \sqrt{n} + 1 \right\rfloor, \\ x_{min} &= -\frac{\sqrt{n \ln n}}{2}, \\ x_{max} &= \frac{\sqrt{n \ln n}}{2}, \\ \nu_n &= \frac{0.4 \ln^{1.5}(n)}{\sqrt{n}}, \\ \tilde{p}_{coe} &= \sqrt{\frac{2}{\pi n}} \cdot (1 - \nu_n). \end{aligned} \tag{4.8}$$

In line 1−4, the parties sample a geometric random variable s_j and s'_j, and convert it to signed integer and fixed-point number for future operations. In line 6 − 9, the parties calculate $x_j = k_j \cdot m_j + l_j$. In line 17, the parties extract the correct $\langle \boldsymbol{i}\rangle^{B,SI}$ based on $\langle f\, g_0\rangle^B, \ldots, \langle f\, g_{iter-1}\rangle^B$.

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

Input: $\sqrt{n} \approx 2^{48}$

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

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

2 : $\quad \left\langle \boldsymbol{s_j} \right\rangle^{B,UI} \leftarrow \Pi^{Geometric}$

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

4 : $\quad \left\langle \boldsymbol{s_j} \right\rangle^{B,FP} \leftarrow \mathrm{SI2FP}\left(\left\langle \boldsymbol{s_j} \right\rangle^{B,SI}\right)$

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

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

7 : $\quad \left\langle \boldsymbol{k_j} \right\rangle^{B,SI} \leftarrow \left\langle \boldsymbol{s_j} \right\rangle^{B,SI} \cdot \left\langle b_j \right\rangle^{B} \oplus \left\langle \boldsymbol{s'_j} \right\rangle^{B,SI} \cdot \mathrm{NOT}\left(\left\langle b_j \right\rangle^{B}\right)$

8 : $\quad \left\langle \boldsymbol{l_j} \right\rangle^{B,UI} \leftarrow \Pi^{RandInt}\,(m)$

9 : $\quad \left\langle \boldsymbol{x_j} \right\rangle^{B,SI} \leftarrow \left\langle \boldsymbol{k_j} \right\rangle^{B,SI} \cdot m + \mathrm{UI2SI}\left(\left\langle \boldsymbol{l_j} \right\rangle^{B,UI}\right)$

10 : $\quad \left\langle cond_{x_{min} \le x_j \le x_{max}} \right\rangle^{B} \leftarrow \left(\left\langle \boldsymbol{x_j} \right\rangle^{B,SI} \ge x_{min}\right) \wedge \left(\left\langle \boldsymbol{x_j} \right\rangle^{B,SI} \le x_{max}\right)$

11 : $\quad \left\langle \tilde{\boldsymbol{p}}_j \right\rangle^{B,FP} \leftarrow \tilde{p}_{coe} \cdot e^{\left(\frac{-2}{\sqrt{n}} \cdot \left\langle \boldsymbol{x_j} \right\rangle^{B,FP}\right)^2}.$

12 : $\quad \left\langle cond_{\tilde{p}_j > 0} \right\rangle^{B} \leftarrow \left(\tilde{p}_{coe} > 0\right)$

13 : $\quad \langle \boldsymbol{p_{Bernoulli}} \rangle^{B,FP} \leftarrow \left\langle \tilde{\boldsymbol{p}}_j \right\rangle^{B,FP} \cdot \mathrm{UI2FP}\left(2^{\left\langle \boldsymbol{s_j} \right\rangle^{B,UI}}\right) \cdot \frac{m}{4}$

14 : $\quad \left\langle c_j \right\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(\langle \boldsymbol{p_{Bernoulli}} \rangle^{B,FP}\right)$

15 : $\quad \left\langle cond_{c_j == 0} \right\rangle^{B} \leftarrow \left(\left\langle c_j \right\rangle^{B} == 0\right)$

16 : $\quad \left\langle f\,g_j \right\rangle^{B} \leftarrow \left\langle cond_{x_{min} \le x_j \le x_{max}} \right\rangle^{B} \wedge \left\langle cond_{\tilde{p}_j > 0} \right\rangle^{B} \wedge \left\langle cond_{c_j == 0} \right\rangle^{B}$

17 : $\langle \boldsymbol{i} \rangle^{B,SI} \leftarrow \Pi^{OAA}\left(\left\langle \boldsymbol{x_0} \right\rangle^{B,SI}, \ldots, \left\langle \boldsymbol{x_{iter-1}} \right\rangle^{B,SI}, \langle f\,g_0 \rangle^{B}, \ldots, \langle f\,g_{iter-1} \rangle^{B}\right)$

Protocol 4.9: MPC protocol for symmetrical binomial distribution $i \sim SymmBino\left(n \approx 2^{96}, p = 0.5\right)$.

MPC-DP Protocol for Integer-scaling Gaussian Mechanism

Recall integer-scaling Gaussian mechanism $M_{ISGauss}\left(f_r\left(D\right), r, \varepsilon, \sigma, \Delta_r\right)$ § 3.2.2 that approximates the Gaussian mechanism is defined as follows:

$$\begin{aligned} M_{ISGauss}\left(f_r\left(D\right), r, \varepsilon, \sigma, \Delta_r\right) &= f_r\left(D\right) + ir \\ &= f_r\left(D\right) + \left(2^{52} + i\right) \cdot r - 2^{52} \cdot r, \end{aligned} \tag{4.9}$$

where $i \sim SymmBino\,(n, p)$.

In our MPC-DP general framework Prot. 4.1, it can be reformulated as:

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

where r, ϵ, Δ_r, λ are public known.

We present the Prot. 4.10 for $\langle M_{ISGauss}\left(f_r\left(D\right), r, \varepsilon, \sigma, \Delta_r\right)\rangle$. We assume the parties have already computed $\langle \boldsymbol{f_r\left(D\right)}\rangle^{B,FL}$.

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

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

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

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

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

3 : $\langle \boldsymbol{Y_{BinoNoise}}\rangle^{B,FL} \leftarrow \left\langle \boldsymbol{i_{float}}\right\rangle^{B,FL} \cdot r - 2^{52} \cdot r$.

4 : $\langle \boldsymbol{M_{ISGauss}}\rangle^{B,FL} \leftarrow \langle \boldsymbol{f_r\left(D\right)}\rangle^{B,FL} + \langle \boldsymbol{Y_{BinoNoise}}\rangle^{B,FL}$.

Protocol 4.10: MPC-DP protocol for SNG Laplacian Mechanism.

4.5 MPC Protocols for Discrete Gaussian Mechanism

We present two primary MPC protocols Prot. 4.11 and Prot. 4.12 for discrete Gaussian mechanism (cf. **??**) in this section.

Protocol: $\Pi^{DiscreteLap}\left(n, d\right)$

Input: n, d

Output: $\langle s\rangle^B$ **and** $\langle \boldsymbol{m}\rangle^{B,UI}$**, where** $x \sim DLap\left(\frac{n}{d}\right)$

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

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

3 : $\left\langle \boldsymbol{m}_j\right\rangle^{B,UI} \leftarrow \Pi^{GeometricEXP}\left(d, n\right)$

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

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

Protocol 4.11: MPC Protocol for discrete Laplace $x \sim DLap\left(\frac{n}{d}\right)$.

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

Input: σ

Output: $\langle s\rangle^{B}$ **and** $\langle \boldsymbol{m}\rangle^{B,UI}$, **where** $Y=(-1)^{s}\cdot m$ **and** $Y\sim DGauss(\mu=0,\sigma)$

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

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

3 : $\quad \left\langle \boldsymbol{m_j}\right\rangle^{B,UI}, \left\langle s_j\right\rangle^{B} \leftarrow \Pi^{DiscreteLap}(t,1)$

4 : $\quad \left\langle b_j\right\rangle^{B} \leftarrow \Pi^{BernoulliEXP}\left(\frac{\left(\mathrm{UI2FP}\left(\left\langle \boldsymbol{m}_j\right\rangle^{B,UI}\right)-\frac{\sigma^2}{t}\right)^2}{2\sigma^2}\right)$

5 : $\langle \boldsymbol{m}\|s\rangle^{B} \leftarrow \Pi^{OAA}\left(\langle \boldsymbol{m_0}\|s_0\rangle^{B},\ldots,\left\langle \boldsymbol{m_{iter-1}}\|s_{iter-1}\right\rangle^{B},\langle b_0\rangle^{B},\ldots,\langle b_{iter-1}\rangle^{B}\right)$

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

4.5.1 MPC-DP Protocol for Discrete Gaussian Mechanism

Recall discrete Gaussian mechanism (cf. § 3.3) is defined as follows:

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

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

In our MPC-DP general framework Prot. 4.1, it can be reformulated as:

$$\langle M_{DGauss}(f(D),\sigma)\rangle=\langle f(D)\rangle+\langle Y\rangle.$$

In Prot. 4.13, we assume that the parties have already computed $\langle f(D)\rangle$. In line 2, we convert unsigned integer m to a signed integer with s as its sign bit.

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

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

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

1 : $\langle \boldsymbol{m}\rangle^{B,UI}, \langle s\rangle^{B} \leftarrow \Pi^{DiscreteGauss}(\sigma)$

2 : $\langle \boldsymbol{m}\rangle^{B,SI} \leftarrow \mathrm{UI2SI}\left(\langle \boldsymbol{m}\rangle^{B,UI}, \langle s\rangle^{B}\right)$

3 : $\langle \boldsymbol{M_{DGauss}}\rangle^{B,SI} \leftarrow \langle \boldsymbol{f(D)}\rangle^{B,SI}+\langle \boldsymbol{m}\rangle^{B,SI}$

Protocol 4.13: MPC-DP protocol for Discrete Gaussian Mechanism.

5 Implementaion and Evaluation

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).

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

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

[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.

[BDST20] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. **"MOTION-A Framework for Mixed-Protocol Multi-Party Computation."** In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1137.

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

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

[BK15] E. BARKER, J. KELSEY. **"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"**. Computer Security Division Information Technology Laboratory, 2015.

[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: *arXiv preprint arXiv:2010.05867* (2020).

[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: *arXiv preprint arXiv:2004.00010* (2020).

[Cla15] J. CLARK. 2015.

[Com19] M. S. COMMITTEE. **"IEEE Standard for Floating-Point Arithmetic"**. In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84.

[Cov19] C. COVINGTON. **"Snapping Mechanism Notes"**. `https://github.com/ctcovington/floating_point/blob/master/snapping_mechanism/notes/snapping_implementation_notes.pdf`. 2019.

[Dev86] L. DEVROYE. **"Non-Uniform Random Variate Generation"**. Springer New York, 1986.

[DMNS06] C. DWORK, F. MCSHERRY, K. NISSIM, A. SMITH. **"Calibrating noise to sensitivity in private data analysis"**. In: *Theory of cryptography conference*. Springer. 2006, pp. 265–284.

[DN03] I. DINUR, K. NISSIM. **"Revealing information while preserving privacy"**. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2003, pp. 202–210.

[DR+14] C. DWORK, A. ROTH. **"The algorithmic foundations of differential privacy."** In: *Found. Trends Theor. Comput. Sci.* 9.3-4 (2014), pp. 211–407.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *NDSS'15*. Internet Society, 2015.

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

[EGL85] S. EVEN, O. GOLDREICH, A. LEMPEL. **"A Randomized Protocol for Signing Contracts"**. In: *Commun. ACM* 28.6 (1985), pp. 637–647.

[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.

[Gol87] S. GOLDWASSER. **"How to play any mental game, or a completeness theorem for protocols with an honest majority"**. In: *Proc. the Nineteenth Annual ACM STOC'87* (1987), pp. 218–229.

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

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *Advances in Cryptology (CRYPTO'03)*. Vol. 2729. LNCS. Springer, 2003, pp. 145–161.

[IR90] R. IMPAGLIAZZO, S. RUDICH. **"Limits on the Provable Consequences of One-way Permutations"**. In: *Advances in Cryptology — CRYPTO' 88*. Springer New York, 1990, pp. 8–26.

[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.

[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.

[Mir12] I. MIRONOV. **"On significance of the least significant bits for differential privacy"**. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 650–661.

[MKGV07] A. MACHANAVAJJHALA, D. KIFER, J. GEHRKE, M. VENKITASUBRAMANIAM. **"l-diversity: Privacy beyond k-anonymity"**. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), 3–es.

[MRT20] P. MOHASSEL, M. ROSULEK, N. TRIEU. **"Practical Privacy-Preserving K-means Clustering."** In: *Proc. Priv. Enhancing Technol.* 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.

[PR96] J. K. PATEL, C. B. READ. **"Handbook of the normal distribution"**. Vol. 150. CRC Press, 1996, pp. 28–29.

[Rab05] M. O. RABIN. **"How To Exchange Secrets with Oblivious Transfer"**. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. 2005.

[RR21] M. ROSULEK, L. ROY. **"Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits"**. In: (2021).

[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.

[Swe97] L. SWEENEY. **"Weaving technology and policy together to maintain confidentiality"**. In: *The Journal of Law, Medicine & Ethics* 25.2-3 (1997), pp. 98–110.

[Tea20] G. D. P. TEAM. **"Secure Noise Generation"**. `https://github.com/google/differential-privacy/blob/main/common_docs/Secure_Noise_Generation.pdf`. 2020.

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

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

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

[WOG88] A. WIGDERSON, M. OR, S. GOLDWASSER. **"Completeness theorems for non-cryptographic fault-tolerant distributed computations"**. In: *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*. 1988, pp. 1–10.

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets"**. In: *Foundations of Computer Science (FOCS'86)*. IEEE, 1986, pp. 162–167.

[Zoh17] M. ZOHNER. **"Faster Oblivious Transfer Extension and Its Impact on Secure Computation"**. In: (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 Algorithms for Probability Sampling Methods

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

Input: None

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

IF $n == 0$
    RETURN 0
WHILE TRUE
    $u \leftarrow Algo^{RandInt}(d)$
    $b_1 \leftarrow Algo^{BernoulliEXP1}(u, d)$
    IF $b_1 == 1$
        BREAK
$k \leftarrow 0$
WHILE TRUE
    $b_2 \leftarrow Algo^{BernoulliEXP1}(1)$
    IF $b_2 == 1$
        $k \leftarrow k + 1$
    ELSE
        BREAK
RETURN $x \leftarrow \frac{k \cdot d + u}{n}$

Algorithm A.1: Algorithm for geometric distribution $x \sim Geo\left(p = 1 - e^{\frac{n}{d}}\right)$ [CKS20].

Algorithm: $Algo^{BernoulliEXP1}(\gamma)$

Input: γ

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

$j \leftarrow 1$
WHILE TRUE
    $b \leftarrow Algo^{Bernoulli}\left(\frac{\gamma}{j}\right)$
    IF $b == 1$
        $j \leftarrow j + 1$
    ELSE
        BREAK
RETURN $x \leftarrow \text{MOD2}(j)$

Algorithm A.2: Algorithm for Bernoulli distribution $x \sim Bern\left(p = e^{-\gamma}\right)$ with $\gamma \in [0,1]$ [CKS20].

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

Input: γ

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

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

Algorithm A.3: Algorithm for Bernoulli distribution $x \sim Bern\left(p = e^{-\gamma}\right)$ [CKS20].

A.1.1 Random Integer

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

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

Input: m

Output: x

1 : Generate random bits $b_0, \ldots, b_{l+\kappa-1}$

2 : Calculate $r = \sum_{j=0}^{l+\kappa-1} 2^j b_j$

3 : Calculate $x = r \mod m$

Algorithm A.4: Algorithm for generate random integer $x \in [0, \ldots, m)$.

A.2 MPC Protocols

A.2.1 Oblivious Array Access

Given an array of ℓ unsigned integers $y_0, \ldots, y_{\ell-1}$ and ℓ-bit string $c_0, \ldots, c_{\ell-1}$. Π^{OAA} outputs y_i where i is the first index such that $c_i == 1$ for $i \in [0, \ell-1]$. We present Π^{OAA_a} Prot. A.1 and Π^{OAA_b} to realize this functionality.

Protocol: $\Pi^{OAA_a}\left(\langle \boldsymbol{y_0} \rangle^{B,UI}, \ldots, \langle \boldsymbol{y_{\ell-1}} \rangle^{B,UI} \langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$

Input: $\left(\langle \boldsymbol{y_0} \rangle^{B,UI}, \ldots, \langle \boldsymbol{y_{\ell-1}} \rangle^{B,UI}\right)$, $\left(\langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$

Output: $\langle \boldsymbol{y_i} \rangle^{B,UI}$, **where** i **is the smallest index such that** $c_i == 1$ **for** $i \in [0, \ell-1]$

1 : $\left(\langle e_0 \rangle^B, \ldots, \langle e_{\ell-1} \rangle^B\right) \leftarrow \Pi^{PreOr}\left(\langle c_0 \rangle^B, \ldots, \langle c_{\ell-1} \rangle^B\right)$

2 : **FOR** $j = 1$ **TO** $\ell - 1$

3 : $\quad \langle p_j \rangle^B \leftarrow \langle e_j \rangle^B \oplus \langle e_{j-1} \rangle^B$, where $\langle p_0 \rangle^B \leftarrow \langle e_0 \rangle^B$

4 : $\langle \boldsymbol{y_i} \rangle^{B,UI} \leftarrow \oplus_{j=0}^{\ell-1} \langle p_j \rangle^B \cdot \langle \boldsymbol{y_j} \rangle^{B,UI}$

Protocol A.1: Protocol for OAA-a.

Note that Π^{PreOr} in Π^{OAA_a} (cf. Prot. A.1) has at least ℓ-depth for AND gate, that can be inefficient for large ℓ using GMW (cf. § 2.2.4). Π^{OAA_b} is inspired by the works[JLL+19; MRT20]. Π^{OAA_b} 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 $\langle \boldsymbol{y} \rangle^{B,UI}$. 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 $(\left(\langle c_{a,b} \rangle, \langle \boldsymbol{y_{a,b}} \rangle^{B,UI}\right))$ in the j-th layer are computed with the value of two nodes (with value $\left(\langle c_a \rangle, \langle \boldsymbol{y_a} \rangle^{B,UI}\right)$ and $\left(\langle c_b \rangle, \langle \boldsymbol{y_b} \rangle^{B,UI}\right)$) in the $j-1$-th layer as follows:

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

which is equivalent to

$$\left\langle c_{a,b}\right\rangle = \langle c_a\rangle \oplus \langle c_b\rangle \oplus (\langle c_a\rangle \wedge \langle c_b\rangle) \tag{A.2}$$

$$\begin{aligned} \left\langle \boldsymbol{y_{a,b}}\right\rangle = \quad & (\langle c_a\rangle \oplus \langle c_b\rangle) \cdot \left(\langle \boldsymbol{y_a}\rangle^{B,UI} \cdot \langle c_a\rangle \oplus \langle \boldsymbol{y_b}\rangle^{B,UI} \cdot \langle c_b\rangle\right) \\ & \oplus (\langle c_a\rangle \wedge \langle c_b\rangle) \cdot \langle \boldsymbol{y_a}\rangle^{B,UI} \end{aligned} \tag{A.3}$$

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

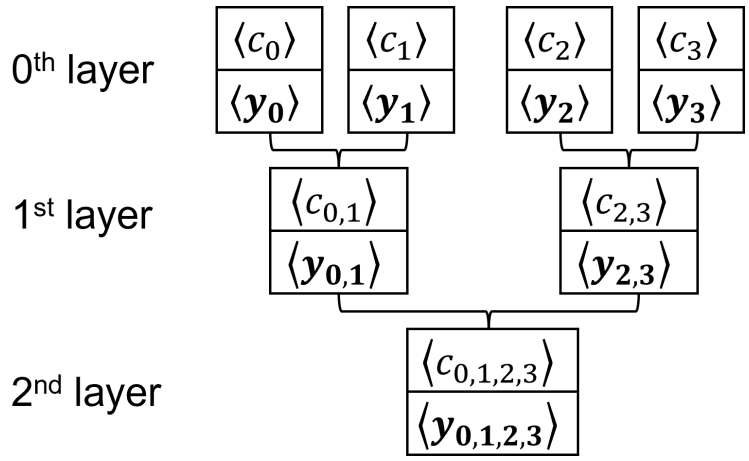

Figure A.1: Example inverted binary tree for Π^{OAA_b}.

A.2.2 Geometric Distribution

Prot. A.2 is constructed based on $Algo^{Geometric}$ (cf. Algorithm 2.2). $(u_0, \ldots, u_{iter}) \in \{0,1\}^{iter}$ is a bit string with uniform bits. In line 2, 3, we set the bits in $b_0, \ldots, b_{iter-1}$ to one when the corresponding bits in $u_0, \ldots, u_{iter}$ are the leading zero bits, and other bits in $b_0, \ldots, b_{iter-1}$ to zero. Recall that the geometric distribution (cf. 2.3.7) counts the number of Bernoulli trials up to and including the first success. Therefore, we append bit 1 and compute the Hamming Weight of string $1, b_0, \ldots, b_{iter-1}$.

Protocol: $\Pi^{Geometric}$

Input: None

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

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

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

3 : $\left(\langle b_0 \rangle^B, \ldots, \langle b_{iter-1} \rangle^B\right) = \left(\mathrm{NOT}\left(\langle p_0 \rangle^B\right), \ldots, \mathrm{NOT}\left(\langle p_{iter-1} \rangle^B\right)\right)$

4 : $\langle y \rangle^{B,UI} = \Pi^{HW}\left(1, \langle b_0 \rangle^B, \ldots, \langle b_{iter-2} \rangle^B\right)$.

Protocol A.2: MPC Protocol for geometric distribution $y \sim Geo\,(p = 0.5)$.

TODO: anaylyse about differential privacy, $iter_2$ leak information of j in second while loop
Prot. A.3 converts $Algo^{GeometricExp}\,(n, d)$ (cf. Algorithm A.1) into MPC protocol for $n \neq 0$.

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

Input: n, d

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

1 : **FOR** $j = 0$ **TO** $iter_1 - 1$

2 : $\quad \left\langle u_j \right\rangle^{B,UI} \leftarrow \Pi^{RandInt}\,(d)$

3 : $\quad \left\langle \gamma_{1,j} \right\rangle^{B,FP} \leftarrow \dfrac{\left\langle u_j \right\rangle^{B,UI}}{d}$

4 : $\quad \left\langle b_{1,j} \right\rangle^{B} = \Pi^{BernoulliEXP1}\left(\left\langle \gamma_{1,j} \right\rangle^{B,FP}\right)$

5 : **FOR** $k = 0$ **TO** $iter_2 - 1$

6 : $\quad \left\langle b_{2,k} \right\rangle^{B} = \Pi^{BernoulliEXP1}\,(1)$

7 : $\langle u \rangle^{B,UI} \leftarrow \Pi^{OAA}\left(\left\langle u_0 \right\rangle^{B,UI}, \ldots, \left\langle u_{iter_1 - 1} \right\rangle^{B,UI}, \left\langle b_{1,0} \right\rangle^{B}, \ldots, \left\langle b_{1,iter_1 - 1} \right\rangle^{B}\right)$

8 : $\langle k \rangle^{B,UI} \leftarrow \Pi^{OAA}\left(0, \ldots, iter_2 - 1, \left\langle b_{2,0} \right\rangle^{B}, \ldots, \left\langle b_{2,iter_2 - 1} \right\rangle^{B}\right)$

9 : $\langle x \rangle^{B,UI} \leftarrow \langle k \rangle^{B,UI} \cdot \dfrac{d}{n} + \dfrac{\langle u \rangle^{B,UI}}{n}$

Protocol A.3: MPC Protocol for geometric distribution $x \sim Geo\left(p = 1 - e^{\frac{n}{d}}\right)$.

A.2.3 Bernoulli Distribution

Prot. A.4 is based on Algorithm 2.1.

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

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

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

1 : $\langle \boldsymbol{U} \rangle^{B,UI} \leftarrow \Pi^{RandBits}(11)$

2 : $\langle \boldsymbol{U} \rangle^{B,FP} \leftarrow \text{UI2FP}\left(\langle \boldsymbol{U} \rangle^{B,UI}\right)$

3 : $\langle \boldsymbol{U} \rangle^{B,FP} \leftarrow \text{right-shift}\left(\langle \boldsymbol{U} \rangle^{B,FP}, 11\right)$

4 : $\langle x \rangle^{B} \leftarrow \left(\langle p \rangle^{B,FP} > \langle \boldsymbol{U} \rangle^{B,FP}\right)$

Protocol A.4: MPC Protocol for Bernoulli distribution $x \sim Bern(p)$.

Prot. A.5 convert $Algo^{BernoulliEXP1}(\gamma)$ [CKS20] into MPC protocols.

Protocol: $\Pi^{BernoulliEXP1}\left(\langle \boldsymbol{\gamma} \rangle^{B,FP}\right)$

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

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

1 : **FOR** $j \in [1 \ldots iter]$

2 : $\quad \left\langle \boldsymbol{p}_j \right\rangle^{B,FP} \leftarrow \dfrac{\langle \boldsymbol{\gamma} \rangle^{B,FP}}{j}$

3 : $\quad \left\langle b_j \right\rangle^{B} \leftarrow \Pi^{Bernoulli}\left(\left\langle \boldsymbol{p}_j \right\rangle^{B,FP}\right)$

4 : $\quad \left\langle fg_j \right\rangle^{B} \leftarrow \left(\left\langle b_j \right\rangle^{B} == 1\right)$

5 : $\langle x \rangle^{B} \leftarrow \Pi^{OAA}\left(j_{mod2}, \langle fg_1 \rangle^{B}, \ldots, \langle fg_{iter} \rangle^{B}\right)$, where $j_{mod2} = (1,0,1,0,\ldots)$

Protocol A.5: MPC Protocol for Bernoulli distribution $x \sim Bern\left(p = e^{-\gamma}\right)$, where $\gamma \in [0,1]$.

Prot. A.6 converts $Algo^{BernoulliEXP}(\gamma)$ [CKS20] into MPC protocol.

Note that in line 10, the parties compute $\langle x \rangle^{B}$ with $\langle \boldsymbol{b} \rangle^{B} = \left(\langle b_1 \rangle^{B}, \left\langle b_{2,0} \right\rangle^{B}, \ldots, \left\langle b_{2,iter-1} \right\rangle^{B}, \langle b_3 \rangle^{B}\right)$ and $\langle \boldsymbol{fg} \rangle^{B} = \left(\langle cond_{b1} \rangle^{B}, \left\langle cond_{b2,0} \right\rangle^{B}, \ldots, \left\langle cond_{b2,iter-1} \right\rangle^{B}, 1\right)$.

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

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

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

1 : $\langle cond_{b1} \rangle^{B} \leftarrow \left(\langle \gamma \rangle^{B,FP} \leq 1\right)$

2 : $\langle b_1 \rangle^{B} \leftarrow \Pi^{BernoulliEXP1}\left(\langle \gamma \rangle^{B,FP}\right)$

3 : $\langle \lfloor \gamma \rfloor \rangle^{B,FP} \leftarrow \text{Floor}\left(\langle \gamma \rangle^{B,FP}\right)$

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

5 : $\quad \left\langle b_{2,j} \right\rangle^{B} \leftarrow \Pi^{BernoulliEXP1}(1)$

6 : $\quad \left\langle cond_{b_{2,j}==0} \right\rangle^{B} \leftarrow \left(\left\langle b_{2,j} \right\rangle^{B} == 0\right)$

7 : $\quad \left\langle cond_{b2,j} \right\rangle^{B} \leftarrow \left(j < \langle \lfloor \gamma \rfloor \rangle^{B,FP}\right) \wedge \left\langle cond_{b_{2,j}==0} \right\rangle^{B}$

8 : $\langle \gamma - \lfloor \gamma \rfloor \rangle^{B,FP} \leftarrow \langle \gamma \rangle^{B,FP} - \langle \lfloor \gamma \rfloor \rangle^{B,FP}$

9 : $\langle b_3 \rangle^{B} = \Pi^{BernoulliEXP1}\left(\langle \gamma - \lfloor \gamma \rfloor \rangle^{B,FL}\right)$

10 : $\langle x \rangle^{B} \leftarrow \Pi^{OAA}\left(\langle \boldsymbol{b} \rangle^{B}, \langle fg \rangle^{B}\right)$

Protocol A.6: MPC Protocol for Bernoulli distribution $x \sim Bern\left(p = e^{-\gamma}\right)$.

A.2.4 Random Integer

Prot. A.7 convert $Algo^{RandInt}(m)$ into MPC protocol.
TODO: Problem with unsigned integer because κ is requires be greater than 64.

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

Input: m

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

1 : $\langle \boldsymbol{r} \rangle^{B,UI} \leftarrow \Pi^{Randbits}(l + kappa)$

2 : $\langle \boldsymbol{x} \rangle^{B,UI} \leftarrow \text{MOD}\left(\langle \boldsymbol{r} \rangle^{B,UI}, m\right)$

Protocol A.7: MPC Protocol for random integer $x \leftarrow\!\!\$\, [0, m)$.

A.2.5 Binary2Unary

TODO: Binary2Unary needs to be improved. $\Pi^{Binary2Unary}\left(\langle a \rangle^{A}, l\right)$ [ABZS12] converts integer a from binary to unary bitwise representation and outputs a l-bit string $\boldsymbol{p} = (p_0, \ldots, p_{l-1})$, where the a least significant bits $(p_0, \ldots, p_{a-1})$ are set to 1 and others to 0. In line 1, 2, we calculate the 2^a and convert it to boolean shares. Then in line 3 we generate $l + k$

random bits and hide 2_a by adding it with the $l+k-1$-bit integer and reconstruct the addition result c. In line 5, the plaintext value c is decomposed into binary bits. In line 6, we compute XOR of c_j and correspond bit u_j. Then in line 7, by $PreOr$ we get $\left(g_{l-1},\ldots,g_j\right)=\left(\overline{1}_{(l-j+1)}\right)$ and $\left(g_{j-1},\ldots,g_0\right)=\left(\overline{0}_{(j)}\right)$, where $j-1=a$. Finally, we calculate $(p_{l-1},\ldots,p_0)=\text{NOT}\,(g_{l-1},\ldots,g_0)$ and the number of non-zero bits in $(p_{l-1},\ldots,p_0)$ equals to a. The share conversion in line 4 can be omitted if arithmetic of boolean shares is available.

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

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

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

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

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

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

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

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

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

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

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

Protocol A.8: MPC Protocol for binary to unary conversion.

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

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

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

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

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

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

Protocol A.9: MPC Protocol for binary to unary conversion.

A.3 Proofs

A.3.1 Function $\mathrm{Split}(L, R, \lambda)$

Function $\mathrm{Split}(L, R, \lambda) = L\text{—}\mathrm{int}\left(\frac{\ln(0.5)+\ln\left(1+e^{-\lambda(R-L)}\right)}{\lambda}\right)$ calculates the middle point M of interval $(L \ldots R]$ such that for $Geo\left(p = 1-e^{-\lambda}\right)$'s PMF

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

First, we calculate $\Pr(L < x \leq M)$ and $\Pr(L < x \leq R)$ as follows:

$$\begin{aligned}
\Pr(L < x \leq M) &= \Pr(x \leq M) - \Pr(x \leq L) \\
&= \left(1-(1-p)^M\right) - \left(1-(1-p)^L\right) \\
&= \left(1-e^{-\lambda M}\right) - \left(1-e^{-\lambda L}\right) \\
&= e^{-\lambda L} - e^{-\lambda M} \\
&= \frac{1}{e^{\lambda L}} - \frac{1}{e^{\lambda M}} \\
&= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda L+\lambda M}},
\end{aligned} \tag{A.4}$$

$$\begin{aligned}
\Pr(L < x \leq R) &= \Pr(x \leq R) - \Pr(x \leq L) \\
&= \left(1-(1-p)^R\right) - \left(1-(1-p)^L\right) \\
&= \left(1-e^{-\lambda R}\right) - \left(1-e^{-\lambda L}\right) \\
&= e^{-\lambda L} - e^{-\lambda R} \\
&= \frac{1}{e^{\lambda L}} - \frac{1}{e^{\lambda R}} \\
&= \frac{e^{\lambda R} - e^{\lambda L}}{e^{\lambda L+\lambda R}}.
\end{aligned} \tag{A.5}$$

Then, we calculate $\Pr(L < x \leq M \mid L < x \leq R)$ as follows:

$$\begin{aligned}
\Pr(L < x \leq M \mid L < x \leq R) &= \frac{\Pr(L < x \leq M)}{\Pr(L < x \leq R)} \\
&= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda L+\lambda M}} \cdot \frac{e^{\lambda L+\lambda R}}{e^{\lambda R} - e^{\lambda L}} \\
&= \frac{e^{\lambda M} - e^{\lambda L}}{e^{\lambda R} - e^{\lambda L}} \cdot e^{\lambda(R-M)} \\
&= \frac{e^{\lambda R} - e^{(L+R-M)}}{e^{\lambda R} - e^{\lambda L}}.
\end{aligned} \tag{A.6}$$

Finally, we calculate the value middle point M such that $\Pr(L < x \leq M \mid L < x \leq R) \approx \frac{1}{2}$ as follows:

$$
\begin{aligned}
\Pr(L < x \leq M \mid L < x \leq R) &\approx \frac{1}{2} \\
\frac{e^{\lambda R} - e^{\lambda(L+R-M)}}{e^{\lambda R} - e^{\lambda L}} &\approx \frac{1}{2} \\
e^{\lambda R} - e^{\lambda(L+R-M)} &\approx \frac{1}{2}\left(e^{\lambda R} - e^{\lambda L}\right) \\
\frac{1}{2}\left(e^{\lambda R} + e^{\lambda L}\right) &\approx e^{\lambda(L+R-M)} \\
\ln\left(\frac{1}{2}\left(e^{\lambda R} + e^{\lambda L}\right)\right) &\approx \ln\left(e^{\lambda(L+R-M)}\right) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) &\approx \lambda(L+R-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) &\approx \ln\left(e^{\lambda R}\right) + \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(e^{\lambda R} + e^{\lambda L}\right) - \ln\left(e^{\lambda R}\right) &\approx \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(\frac{e^{\lambda R} + e^{\lambda L}}{e^{\lambda R}}\right) &\approx \lambda(L-M) \\
\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right) &\approx \lambda(L-M) \\
\frac{\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda} &\approx L - M \\
L - \frac{\ln\left(\frac{1}{2}\right) + \ln\left(1 + e^{-\lambda(R-L)}\right)}{\lambda} &\approx M
\end{aligned}
\tag{A.7}
$$