Secure Multiparty Computation of the Laplace Mechanism

Amir Zarei[a] and Staal A. Vinterbo[b]
Norwegian University of Science and Technology, Norway

Keywords: Secure Multiparty Computation, Differential Privacy, Laplace Mechanism.

Abstract: Differential Privacy (DP) employs perturbation mechanisms to protect individual data, formulated as $Y = q(D) + i$, where $q(D)$ is a query result from dataset D and i is random noise. Adjusting the variance of i ensures that an adversary cannot discern if Y originates from D or its neighboring D'. Complications arise when computing Y using floating-point (FL) or fixed-point (FP) arithmetic. Such approximations mean not every potential output for Y is feasible, leading to a mismatch between the output of $q(D) + i$ and $q(D') + i$ that allows the adversary to breach DP. One solution is to approximate Y as $\tilde{Y} = q_r(D) + ir$, where $q_r(D)$ is $q(D)$ rounded to a discretization factor $r = 2^{-k}$. However, integrating this solution into secure multiparty computation (MPC) is still unexplored. Our work addresses this challenge by proposing an MPC protocol that rounds an FL number to the nearest multiple of r. We show how this protocol enables secure MPC of $\tilde{Y}$ by introducing two Laplace mechanisms. These are then specifically adapted for linear queries. Importantly, our protocols support real-valued query functions on FL inputs and are not limited to integer-valued ones. The first protocol uses FL arithmetic to generate the noise for DP, while the second utilizes integer arithmetic. Both offer information-theoretical security against passive adversaries and can be extended for protection against malicious adversaries. We also analyze the complexity of our protocols to evaluate their performance. Our protocols represent the first provably secure MPC for the Laplace mechanism managing real-valued queries.

1 INTRODUCTION

The surge in data collection, driven by artificial intelligence advancements like OpenAI's GPT (Radford et al., 2019) and Google's BERT (Devlin et al., 2019), underscores the need for data security and privacy. Secure multiparty computation (MPC) ensures sensitive information remains protected from unauthorized access during collaborative computations. Meanwhile, differential privacy (DP) (Dwork and Roth, 2014) provides a framework to release statistics without jeopardizing individual privacy. The fusion of MPC and DP ensures data security during computations as well as individual data privacy when sharing the computed results (Dwork et al., 2006).

To illustrate the combination of MPC and DP, consider n parties with private and random inputs $(d_1, s_1), \ldots, (d_n, s_n)$. The secure MPC for a differentially private Laplace mechanism seeks to compute $f(d,s) = q(d) + h(s)$, where q is a query function mapping a set of databases to a real number, h is a Laplace noise sampler, $d = (d_1, \ldots, d_n)$ and $s = (s_1, \ldots, s_n)$. Protocols based on secret sharing over $\mathbb{Z}_{q'}$ were provided (Dwork et al., 2006) to realize f, and later refined (Eriguchi et al., 2021). These protocols treat q as an integer-valued query function and approximate the Laplace distribution using a Truncated Discrete Laplace (TDL) distribution to generate noise compatible with integer arithmetic. However, these protocols only accept integer numbers, limiting their use in scenarios needing precise non-integer calculations. To address this, protocols were proposed that leverage secret sharing over real numbers (Eigner et al., 2014; Wu et al., 2016). These protocols employ floating-point (FL) and fixed-point (FP) representations to handle real numbers. However, a crucial drawback of these protocols lies in the lack of analysis concerning the impact of finite-precision implementations of the Laplace mechanism, which allows an adversary to undermine DP guarantees (Mironov, 2012; Gazeau et al., 2016).

In 2020, improving on the snapping mechanism (Mironov, 2012), the DP team at Google proposed algorithms to approximate the Gaussian and Laplace mechanisms (Google'sDP, 2020). We refer to the latter as GLM in our context. GLM ensures DP

[a] https://orcid.org/0000-0002-0287-513X
[b] https://orcid.org/0000-0003-2633-2305

Zarei, A. and Vinterbo, S.
Secure Multiparty Computation of the Laplace Mechanism.
DOI: 10.5220/0012453700003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 582-593
ISBN: 978-989-758-683-5; ISSN: 2184-4356

under double FL arithmetic, resulting in a noisy output $q_r(D) + ir$. Here, q represents a real-valued query on a database D, rounded to a resolution parameter r, and i indicates a randomly sampled integer from a Discrete Laplace (DL) distribution. This approach resolves the vulnerability inherent in naive FL implementations of the Laplace mechanism and supports real-valued queries. Despite its advantages, GLM has not yet been implemented in MPC settings, which is our research motivation.

2 OTHER RELATED WORKS

Two primary models implement DP in distributed settings: central and local. In the central model, a trusted party aggregates actual inputs from all parties, introduces noise according to a specific distribution and publishes the perturbed result. Our MPC protocols function as alternatives to implementing the central model without relying on a trusted party. On the other hand, the local model (Kasiviswanathan et al., 2011) involves each party independently randomizing its input and sending it to a designated party that aggregates noisy inputs and publishes the result. Known mechanisms in the local model are limited to specific functions, such as aggregate-sum queries (Shi et al., 2011; Wang et al., 2017), and histograms (Bassily and Smith, 2015), making them inadequate for scenarios where more complex functions need to be computed. Furthermore, in this model, the accuracy of the noisy output is limited to a certain extent, affecting the precision of the final results (Beimel et al., 2008; Chan et al., 2012). Although a recent intermediate model called the shuffled model (Cheu et al., 2019) has been introduced, known mechanisms within this model remain applicable only for simple functions like summation (Cheu et al., 2019; Balle et al., 2019), and histograms (Balcer and Cheu, 2019).

In addition to GLM, alternative methods are designed to thwart attacks related to the FL implementation of differentially private mechanisms, focusing on eliminating the need for rounding. One proposal suggested sampling from Laplace and Gaussian noise in a manner that intensifies the computational difficulty for attackers to backtrack the output (Holohan and Braghin, 2021). Another method (OpenDP, 2021) was introduced to produce a Laplace or Gaussian distribution that leverages an arbitrary precision arithmetic library (MPFR'slibrary, 2021). However, these methods remain vulnerable to a novel type of FL threat termed precision-based attacks (Haney et al., 2022) due to avoiding the need for rounding when adding noise in double-precision arithmetic.

2.1 Our Results

This paper aims to deploy GLM in an MPC setting, leveraging secret sharing and analyzing its complexity. We first introduce an MPC framework for the Laplace mechanism handling FL inputs/output(s). In this framework, users provide input shares to computation parties, which then run an MPC protocol to compute GLM and send results to a data analyzer.

As a requirement for the MPC of GLM, we describe an MPC method to round an FL number to the nearest multiple of a power of 2, extending the traditional rounding protocol that rounds to the nearest integer (Aliasgari et al., 2013). We also showcase two techniques for DL distribution sampling: one with FL arithmetic and another using integer arithmetic.

Combining these elements, we formulate two MPC protocols for GLM and tailor them specifically for linear queries. Our protocols are primarily secure against passive adversaries but can be adapted to thwart malicious ones. We analyze our protocols based on the number of rounds and interactive operations, revealing that the first protocol excels in interactive operations while the second requires fewer rounds. In our framework, the users employ FL numbers to represent their input for precise numerical computations. We demonstrate that an alternative approach using integers leads to inherent rounding errors and imprecision.

Our protocols are the first to achieve MPC of GLM, upholding DP properties while adeptly handling real-valued query functions, thereby enhancing their applicability in real-world scenarios. Additionally, our framework and rounding protocol can support the MPC of the Gaussian mechanism, requiring only a modification in the sampling method to use a binomial distribution instead of a DL distribution.

3 PRELIMINARIES

This section first describes the secure MPC setting in use and discusses the primitives underlying our protocols. It then delves into the details of GLM. Finally, it covers number representation for ensuring accuracy in our computations.

3.1 The Secure MPC Setting

We use a linear secret sharing scheme in which $n > 2$ parties $P_1, \ldots, P_n$ jointly compute the function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ using protocol Π. Each party P_i contributes input x_i and receives output y_i. This scheme integrates a secure multiplica-

tion protocol for secret values and ensures inter-party confidentiality via perfectly secure channels. The scheme can be implemented through Shamir's secret sharing (Shamir, 1979) and the protocols introduced in (Ben-Or et al., 1988). In the (t,n) sharing scheme, a value is shared among n parties and can be reconstructed by $t+1$ or more parties but remains concealed from t or fewer, offering information-theoretic security. We define a finite field $\mathbb{F}_{q'}$, where q' is a large prime. The secret sharing of a value $x \in \mathbb{F}_{q'}$ among the parties is denoted by $[x]$. Due to the scheme's inherent linearity, parties can locally compute linear combinations like $[x]+[y]$, $[x]+c$, and $c[x]$ using their respective shares of x and y. However, the multiplication, $[x][y]$, needs interaction between parties. Using the reconstruction (Rec) protocol, $t+1$ or more parties can reveal a shared value in FL or integer representation.

We consider two primary metrics to measure the complexity of a protocol Π. The first, Int(Π), includes operations like multiplication, share distribution, and secret reconstruction. The second evaluates round complexity, capturing the protocol's sequential interactive tasks. Concurrent interactive actions are accounted for as a single round. Our evaluation overlooks overflow, underflow, invalid operation, and division by zero.

We define the security of our protocols using the standard definition of the passive adversary model in MPC. To support security against stronger adversary models, we will employ various techniques discussed in Section 5.

Definition 3.1 (t-secure) *Given the n parties and protocol Π, we define the view of party P_i during the protocol execution as* $\text{VIEW}_\Pi(P_i) = (x_i, r_i, M)$. *This includes its input x_i, random coin tosses r_i, and exchanged messages M. For a coalition T of parties with size $t < \frac{n}{2}$ and combined view* $\text{VIEW}_\Pi(T)$, *protocol Π is t-secure against passive adversaries if there exists a probabilistic polynomial-time simulator S_T. This simulator makes the distribution of $\{\text{VIEW}_\Pi(T), y_T\}$ computationally indistinguishable from $\{S_T(x_T, f(x_1,x_2,...,x_n))\}$, where $x_T = \cup_{P_i \in T}\{x_i\}$ and $y_T = \cup_{P_i \in T}\{y_i\}$.*

3.2 Known Sub-Protocols

The following sub-protocols underpin the protocols in this paper:

- $[r] \leftarrow \text{RandInt}(l)$ allows the parties to create shares of a uniformly random l-bit value $r \in \mathbb{Z} \cap [0, 2^l)$ (Catrina and De Hoogh, 2010b).
- $[r] \leftarrow \text{RandBit}()$ allows the parties to produce shares of a random bit $r \in \{0,1\}$ (Catrina and De Hoogh, 2010b).
- $[r] \leftarrow \text{RandBiasedBit}(s,p)$ takes inputs p and random unbiased bits $s = (s_{i1}, ..., s_{id})$, where $i \in \{1,...,n\}$ and allows the parties to generate shares of a random bit r with bias p (Eriguchi et al., 2021).
- $[c] \leftarrow \text{XOR}([a],[b])$ computes the exclusive Or of shared bits $[a]$ and $[b]$ as $[a]+[b]-2[a][b]$.
- $[c] \leftarrow \text{OR}([a],[b])$ computes the Or of shared bits $[a]$ and $[b]$ as $[a]+[b]-[a][b]$.
- $[c_1],...,[c_l] \leftarrow \text{PRE}_\vee([b_1],...,[b_l])$ takes on l shared bits $[b_i] \in \{0,1\}$, and computes shares of the prefixes as $[c_i] = [\vee_{j=1}^{i} b_j]$ for $i \in \{1,\ldots,l\}$ (Damgård et al., 2006).
- $[c_1],...,[c_l] \leftarrow \text{MULT}^*([b_1],...,[b_l])$, takes l shared values $[b_i] \in \mathbb{F}_q \backslash \{0\}$, and computes shares of the prefix products as $[c_i] = [\prod_{j=1}^{i} b_j]$ for $i \in \{1,\ldots,l\}$ (Damgård et al., 2006).
- $[b] \leftarrow \text{EQ}([x],[y],l)$ takes on two shared l-bit values $[x]$ and $[y]$, and outputs shared bit $[b]=1$ if $[x]=[y]$. We use an implementation of this operation that uses the protocol $[b] \leftarrow \text{EQZ}([x'],l)$, which outputs $[b]=1$ if $[x']=0$ (Catrina and De Hoogh, 2010a).
- $[b] \leftarrow \text{LT}([x],[y],l)$ takes on two shared l-bit values $[x]$ and $[y]$, and outputs shared bit $[b]=1$ if $[x]<[y]$. We use a realization of this function that utilizes the protocol $[b] \leftarrow \text{LTZ}([x'],l)$, which outputs $[b]=1$ if $[x']<0$ (Catrina and De Hoogh, 2010a).
- FLAdd, FLMul, and FLDiv enable the parties to add, multiply, and divide two shared FL numbers.
- FLLog2 takes on a shared FL number $[r]$ and returns the shared FL number $[\log r]$,[1] or an error when $r \leq 0$.
- FLRound takes on a shared FL number $[r]$ and a mode parameter and computes $[\lceil r \rceil]$ if mode = 1, and $[\lfloor r \rfloor]$ if mode = 0.
- FP2FL, Int2FL, and FL2Int enable the parties to convert a number represented as FP, integer, or FL to its corresponding FL or integer representation.

All the protocols that handle computation on FL numbers are detailed in (Aliasgari et al., 2013). Table 1 collects the exact number of rounds and interactive operations required for the aforementioned protocols. Notably, these complexities can be further reduced through pre-computation and when an operand,

[1] For any positive real number a, we denote $\log(a)$ as the logarithm to base 2 of a, and $\ln(a)$ as the natural logarithm of a.

or its components, given to a protocol that handles FL input(s) is not a secret.

3.3 The Laplace Mechanism (GLM)

Here is how GLM works. Let r be a resolution parameter selected according to the scale of the Laplace distribution, i.e., $\frac{\Delta}{\varepsilon}$. Let $q(D)$ be a function mapping a database D to a real number, and let $q_r(D)$ be $q(D)$ rounded to the nearest multiple of r. We define Δ as $\max_{D,D'} \parallel q(D) - q(D') \parallel$, representing the sensitivity of q, and Δ_r as $\max_{D,D'} \parallel q_r(D) - q_r(D') \parallel$, representing the r-sensitivity of q. We assume there is a random noise sampler that takes a 4-tuple $(\varepsilon, \delta, r, \Delta_r)$ and returns i according to a DL distribution. Generally, if $r = 2^k$, where k is an integer in $\{-1022, \ldots, 1023\}$, and $i \sim \text{Sampler}(\varepsilon, \delta, r, \Delta_r)$, then $q_r(D) + ir$ is (ε, δ)-DP. More precisely, given a constant integer k, the algorithm steps for computing GLM are as follows.

1. Set r as the smallest power of 2 greater than $\frac{\varepsilon}{\Delta}$
2. Sample an integer i with a probability proportional to $\exp(-|i| r \frac{\varepsilon}{\Delta_r})$
3. Compute $q_r(D) + ir$

If $q(D) \leq r2^{52}$, $i \leq 2^{52}$, and r is a power of two between 2^{-1022} and 2^{1023}, then GLM is ε-DP under double FL arithmetic (see (Google'sDP, 2020) for DP proofs). The event $q(D) > r2^{52}$ is controlled by adjusting r or constraining $q(D)$. The event $i > 2^{52}$ has a probability under $\exp(-1000)$, which is ignored or seen as the δ mechanism's guarantee. Following (Google'sDP, 2020), we presume the first event would not occur and disregard the second. The value of k determines discretization accuracy, with recommended values between 10 and 45 (Google'sDP, 2020). Parameters r, ε, Δ, Δ_r, and k are public inputs to our protocols.

3.4 Number Representation

A binary FL number u is represented as a quadruple (v, p, z, s), where $v \in [2^{l-1}, 2^l)$ is an l-bit, unsigned, normalized significand, $p \in \mathbb{Z}_{\langle k \rangle} = \{x \in \mathbb{Z} \mid x \in (-2^{k-1}, 2^{k+1})\}$ is a k-bit, signed exponent, z is a bit set to 1 when $u = 0$, and s is a sign bit. The value of the number $u = (1-2s)(1-z)v \cdot 2^p$. Values in $\mathbb{Z}_{\langle k \rangle}$ are encoded in $\mathbb{F}_{q'}$ by the function $\text{fld} : \mathbb{Z}_{\langle k \rangle} \to \mathbb{F}_{q'}, \text{fld}(a) = a \mod q'$.

For a binary FP number w, γ denotes the bit-length of w, and λ denotes the bit-length of the fractional part of w. The value of the number $w = \hat{w} \cdot 2^{-\lambda}$, where $\hat{w} \in \mathbb{Z}_{\langle \gamma \rangle} = \{x \in \mathbb{Z} \mid x \in (-2^{\gamma-1}, 2^{\gamma+1})\}$. Values in $\mathbb{Z}_{\langle \gamma \rangle}$ are encoded in $\mathbb{F}_{q'}$ by the fld function. We also use γ to indicate the bit-length of integer values. We assume that $\gamma = 2\lambda$ for FP numbers and $\gamma = 2l$ for integers and FP numbers as applications usually need (Eigner et al., 2014; Aliasgari et al., 2013). It is also needed that $k > \max(\lceil \log(l+\lambda) \rceil, \lceil \log(\gamma) \rceil)$, and $q' > \max(2^{2l}, 2^{\gamma}, 2^k)$ (Eigner et al., 2014). The parameters l, k, λ, and γ are public inputs (not secrets) to the protocols working with FL and FP numbers. As GLM works on standard double FL arithmetic, we consider $l = 53$ and $k = 11$ while measuring the complexity of our proposed protocols.

4 SECURE MULTIPARTY LAPLACE MECHANISM

This section first discusses a framework for computing the Laplace mechanism over distributed data. We then detail a rounding technique that enables the calculation of the nearest multiple of r and subroutines for generating integer DL distribution samples. Combining these concepts, we introduce two MPC protocols for the Laplace mechanism. In particular, we specialize these protocols for linear queries and comprehensively analyze their complexity.

4.1 The Protocol Framework

We adopt a robust combination of MPC and DP in a scenario aiming to derive insights for a data analyzer from users' information while preserving data security and privacy. To realize this approach, we use n computation parties $C_1, \ldots, C_n$ to collaboratively compute the information in two main phases: aggregation and perturbation.

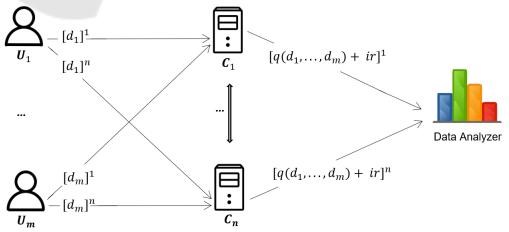

Figure 1: Protocol Flow.

The protocol flow is depicted in Fig. 1. Consider m users; each user, denoted as U_i, has a database D_i, all within the shared domain $D \subseteq \mathbb{R}$. A real-valued query function $q : \mathbb{R}^m \to \mathbb{R}$ maps the users' input to a numerical value(s) from $\mathbb{R}$. The inputs and output(s) are represented using FL numbers to handle real data accurately. The specific nature of q determines the aggregation operations, which may involve summation, averaging, counting, or other statistical methods.

Table 1: Complexity of the employed MPC protocols in this paper.

Protocol	Rounds	Interactive operations
Rec	1	1
RandInt	0	0
RandBit	1	1
RandbiasedBit	11	$(5n+19)d$
XOR	1	1
OR	1	1
$\text{PRE}_{\vee}$	9	$17l$
MULT^*	3	$5l$
LT	4	$4l-2$
EQ	4	$l+4\log l$
FLMul	11	$8l+10$
FLAdd	$\log l+\log\log l+27$	$14l+9k+(\log l)\log\log l+(l+9)\log l+4\log k+37$
FLDiv	$2\log l+7$	$2\log l(l+2)+3l+8$
FP2FL	$\log l+13$	$\log l(2l-1)-10$
FL2Int	$3\log\log(l)+53$	$27l+3\log\log(l)+18\log l+20k+19$
Int2FL	$\log l+13$	$\log l(2l-1)-11$
FLRound	$\log\log l+30$	$15l+(\log l)\log\log l+15\log l+8k+10$
FLLog2	$13.5l+0.5l\log l+3\log l+$ $0.5l\log\log l+3\log\log l+146$	$15l^2+90.5l+0.5l(\log l)\log\log l+0.5(\log l)\log\log l$ $+0.5l^2\log l+11.5l\log l+4.5lk+28k+2l\log k+16\log k+128$

Protocol 1: Π_{GLM}.

Inputs: The FL representation of the private values $\{d_1,...,d_m\}$ and public value r.
Output: The FL representation of $q_r(D)+ir$.

1. The parties run Π_q to compute $q(D)$.
2. The parties run Π_{rnd} to round $q(D)$ to a multiple of r.
3. The parties run Π_i to generate the integer noise i.
4. The parties compute $q_r(D)+ir$ and output the result.

Before computing q, each user securely sends shares of its local computation result, computed as $d_i = q(D_i)$, to the computation parties. In the first phase, aggregation, the computation parties aggregate shared values from users and compute q. In the second phase, perturbation, the computation parties add noise per GLM, i.e., the outcome $q_r(d_1,\ldots,d_m)+ir$ is calculated, where r represents a resolution parameter, and i is an integer random noise from a DL distribution related to $\exp\left(-r\frac{\varepsilon}{\Delta_r}\right)$.

Assuming protocols Π_q, Π_{rnd}, and Π_i respectively compute $q(D)$, round $q(D)$ to a multiple of r, and generate the noise i per the specified DL distribution, then Π_{GLM}, as shown in Protocol 1, represents a general implementation of secure MPC for GLM.

In the literature exploring the fusion of MPC and DP, it is well-established that a protocol which t-securely computes an (ε,δ)-DP mechanism can offer both the t-secrecy property and DP guarantees (refer to (Eriguchi et al., 2021; Wu et al., 2016; Eigner et al., 2014) for formal proofs). To abstract such a protocol, let the parties settle on a uniformly random element s, which can be a value in the range $(0,1]$ (Eigner et al., 2014) or represented as s for a specific set S, e.g., a set of all possible shares of 0 and 1 (Eriguchi et al., 2021; Wu et al., 2016). Subsequently, the parties securely compute a deterministic function (known as a noise sampler), which takes s as input and generates the necessary noise to satisfy DP. Assuming that the distribution of this function provides (ε,δ)-DP even in the presence of passive adversaries for a coalition T of the parties of size $t<\frac{n}{2}$, we can generally construct an (ε,δ)-DP protocol against the coalition T from protocols securely sampling s and computing the noise. Building on this foundational approach, our protocol, Π_{GLM}, adheres to this security framework, ensuring its alignment with the established security guarantees of the fusion between MPC and DP. This approach simplifies distributed noise generation by focusing on the secure computation of deterministic functions and generating a uniformly random element(s). For conciseness, we omit the detailed proof and instead focus on presenting our main results: protocols for implementing Π_{rnd}, Π_i, Π_q, and Π_{GLM}.

4.2 The Rounding Technique

We formulate Π_{rnd} to round the FL value of $q(D)$ to its closest multiple of r, a negative power of two. Recall that r is the smallest power of 2 greater than $\frac{\varepsilon}{\Delta}\times 2^{-k}$, where k ranges from 10 to 45. The study of (Aliasgari et al., 2013) proposed the FLRound protocol (detailed in Appendix A) to round to the nearest integer. Building upon this, we introduce EFLRound, an extension to round to any power of 2. EFLRound is detailed in Protocol 2, where the secret shared value

Protocol 2: EFLRound.

Inputs: The FL representation of the private value $q(D)$ as $\langle [v_q],[p_q],[z_q],[s_q]\rangle$, the public value r as $\langle v_r,p_r,0,0\rangle$, and mode.

Output: The FL representation of the private value $q_r(D)$ as $\langle [v],[p],[z],[s]\rangle$.

1. $[a] \leftarrow \text{LT}([p_q],p_r+l-1,k)$;
2. $[b] \leftarrow \text{LT}([p_q],p_r,k)$;
3. $\langle [v_1],[2^{-(p_q-(p_r+l-1))}]\rangle \leftarrow \text{Mod2m}([v_q],l,-[a](1-[b])([p_q]-(p_r+l-1)))$;
4. $[c] \leftarrow \text{EQZ}([v_1],l)$;
5. $[v] \leftarrow [v_q]-[v_1]+(1-[c])[2^{-(p_g-(p_r+l-1))}](\text{XOR}(\text{mode},[s_q]))$;
6. $[d] \leftarrow \text{EQ}([v],2^l,l+1)$;
7. $[v] \leftarrow [d]2^{l-1}+(1-[d])[v]$;
8. $[v] \leftarrow [a]((1-[b])[v]+[b](\text{mode}-[s_q]))+(1-[a])[v_q]$;
9. $[s] \leftarrow (1-[b]\text{mode})[s_q]$;
10. $[z] \leftarrow \text{OR}(\text{EQZ}([v],l),[z_q])$;
11. $[v] \leftarrow [v](1-[z])$;
12. $[p] \leftarrow (([p_q]+[d][a])(1-[b])+p_r[b])(1-[z])$;
13. return $\langle [v],[p],[z],[s]\rangle$;

$q(D)$ and the public r are inputted as $\langle [v_q],[p_q],[z_q],[s_q]\rangle$ and $\langle v_r,p_r,0,0\rangle$. EFLRound operates in mode 0 to round $q(D)$ down and mode 1 to round it up.

Rounding may be necessary for $q(D)$ only when $[p_q] < p_r+l-1$. In this case $[v_q]$ is truncated by removing the fraction $[2^{-(p_q-(p_r+l-1))}]$. Therefore, lines 1 and 2 of Protocol 2 compare the inputs' exponent $[p_q]$ and p_r. If $[p_q] < p_r$, the value of $q(D)$ lies in $(-r,r)$, and thus the output should be either 0 or $\pm r$ depending on the mode. Moreover, if $[p_q] \geq p_r+l-1$, the output of the rounding function is equal to the input $q(D)$. Significant computation is necessary when $p_r \leq [p_q] < p_r+l-1$, as shown in lines 3-7. In this case, the $-([p_q]-(p_r+l-1))$ least significant bits of the significand $[v_q]$ are set to 0. This is done by first using Mod2m to calculate $v_1 = [v_q \bmod [2^{-(p_q-(p_r+l-1))}]$ in line 3, then computing $[v_q]-[v_1]$ in line 5. Additionally, if $[v_1]$ is not 0 in line 4, the protocol adds $[2^{-(p_q-(p_r+l-1))}]$ to $[v_q-v_1]$ based on the input's sign and mode. Now if $2^{l-1} \leq v_q < 2^l$ and the protocol is adding $[2^{-(p_q-(p_r+l-1))}]$, an overflow occurs, resulting in $v = 2^l$ in line 6. In this case, v is set to 2^{l-1} in line 7, and the exponent is incremented by 1 in line 12. Notice that the input's sign changes only if the input lies in $(-r,0)$ and the mode is set to 0. In this case, line 9 sets the sign s to 0. Finally, lines 10 and 11 compute the zero bit of the result, and line 12 adjusts the exponent.

If $p_r = -l+1$, EFLRound operates equivalently to the FLRound protocol in (Aliasgari et al., 2013). As for complexity, similar to the FLRound protocol, EFLRound requires $15l+(\log\log l)\log l+15\log l+8k+10$ interactive operations and $\log\log l+30$ rounds. To achieve rounding to the nearest multiple of r, we add the FL representation of $\frac{r}{2}$ as $\langle v_r,p_r-1,0,0\rangle$ to $\langle [v_q],[p_q],[z_q],[s_q]\rangle$, and then execute the EFLRound protocol with mode 0. Another idea to compute $q_r(D)$ is to implement $q_r(D) = r\lfloor \frac{q(D)}{r}+\frac{1}{2}\rfloor$. This requires one of each FLDiv, FLAdd, FLRound, and FLMul, which is more expensive.

4.3 Noise Generation Techniques

We now implement Π_i, a protocol to sample i from a DL distribution with probability proportional to $\exp(-r\frac{\varepsilon}{\Delta_r})$. To achieve this, we introduce two techniques that will subsequently provide two protocols for computing GLM.

The FL Implementation of Π_i. To generate a DL random sample, the first technique reduces the problem to generating geometric samples. This is further simplified by generating uniform random values. Given U uniformly distributed over the interval $(0,1]$, we can express two independent geometric random variables, Y_1 and Y_2, both with success probability p, as $Y = \lceil \frac{\ln(U)}{\ln(1-p)} \rceil$. The difference $X = Y_1 - Y_2$ then follows a DL distribution (Devroye, 2006). Recall that per GLM, $p = \exp(-r\frac{\varepsilon}{\Delta_r})$. We can realize a secure MPC of this technique, termed as DL(p), by Protocol 3.

Lines 1-3 of Protocol 3 generate two uniform random numbers in the interval $(0,1]$. This is based on a method by (Eigner et al., 2014), which involves generating two shared $(\gamma+1)$-bit integers $[a_1]$ and $[a_2]$ using RandInt. These integers are then treated as the fractional part of two FP numbers, where the integer parts are set to 0 (represented by $\lambda = \gamma$). To convert these FP numbers into their FL representation, we use FP2FL. Lines 3-9 calculate two geometric samples, and line 10 generates a DL random variable. Recall that FLLog2 computes the logarithm in base 2, while we require the logarithm in terms of the Euler's number (e). To address this, we multiply the results of FLLog2 by the FL representation of $\alpha = \frac{1}{\ln(1-p)\log e}$ in lines 6 and 7. The complexity of Protocol 3 involves 128870 interactive operations in 1203 rounds. This has considered reductions in FLMul's complexity, as it now needs $8l+7$ interactive operations and 10 rounds due to one of its operands provided in the clear, compared to its complexity in Table 1. FLMul

Protocol 3: Discrete Laplace Sampler DL(p).

Inputs: The FL representation of the public input $\alpha = \frac{1}{\ln(1-p)\log e}$ as $\langle v_\alpha, p_\alpha, z_\alpha, s_\alpha \rangle$.
Output: The FL representation of the private value $i \sim$ DL distribution as $\langle [v_i], [p_i], [z_i], [s_i] \rangle$.

1. $[a_1] \leftarrow$ RandInt $(\gamma+1)$; $[a_2] \leftarrow$ RandInt $(\gamma+1)$;
2. $\langle [v_{a1}], [p_{a1}], 0, 0 \rangle \leftarrow$ FP2FL$([a_1], \gamma, \lambda = \gamma, l, k)$;
3. $\langle [v_{a2}], [p_{a2}], 0, 0 \rangle \leftarrow$ FP2FL$([a_2], \gamma, \lambda = \gamma, l, k)$;
4. $\langle [v_1], [p_1], [z_1], [s_1] \rangle \leftarrow$ FLLog2$(\langle [v_{a1}], [p_{a1}], 0, 0 \rangle)$;
5. $\langle [v_2], [p_2], [z_2], [s_2] \rangle \leftarrow$ FLLog2$(\langle [v_{a2}], [p_{a2}], 0, 0 \rangle)$;
6. $\langle [v_{\alpha ln1}], [p_{\alpha ln1}], [z_{\alpha ln1}], [s_{\alpha ln1}] \rangle \leftarrow$ FLMul$(\langle v_\alpha, p_\alpha, z_\alpha, s_\alpha \rangle, \langle [v_1], [p_1], [z_1], [s_1] \rangle)$;
7. $\langle [v_{\alpha ln2}], [p_{\alpha ln2}], [z_{\alpha ln2}], [s_{\alpha ln2}] \rangle \leftarrow$ FLMul$(\langle v_\alpha, p_\alpha, z_\alpha, s_\alpha \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$;
8. $\langle [v_{i1}], [p_{i1}], [z_{i1}], [s_{i1}] \rangle \leftarrow$ FLRound$(\langle [v_{\alpha ln1}], [p_{\alpha ln1}], [z_{\alpha ln1}], [s_{\alpha ln1}] \rangle, 1)$;
9. $\langle [v_{i2}], [p_{i2}], [z_{i2}], [s_{i2}] \rangle \leftarrow$ FLRound$(\langle [v_{\alpha ln2}], [p_{\alpha ln2}], [z_{\alpha ln2}], [s_{\alpha ln2}] \rangle, 1)$;
10. $\langle [v_i], [p_i], [z_i], [s_i] \rangle \leftarrow$ FLSub $(\langle [v_{i1}], [p_{i1}], [z_{i1}], [s_{i1}] \rangle, \langle [v_{i2}], [p_{i2}], [z_{i2}], [s_{i2}] \rangle)$
11. return Rec$(\langle [v_i], [p_i], [z_i], [s_i] \rangle)$;

Protocol 4: Discrete Laplace Sampler TDL(p,N).

Inputs: $s_j = (s_{i1}, ..., s_{id})$ for $j \in \{0, 1, \ldots, N-1\}$ and $i \in \{1, \ldots, n\}$, N, $\alpha_0 = \frac{1-p}{1+p}$ and $\alpha_j = 1-p$.
Output: The integer representation of the private value $i \sim$ DL distribution.

1. $[b_j] \leftarrow$ RandbiasedBit(s_j, α_j) for $j \in \{0, 1, \ldots, N-1\}$;
2. $[c_j] \leftarrow \text{PRE}_\vee[b_j]$ for $j \in \{0, 1, \ldots, N-1\}$;
3. $[l] = N - \sum_{j=0}^{N-1}[c_j]$;
4. Each P_i shares $[\sigma_i] \in \{0, 1\}$ for $i \in \{1, \ldots, n\}$;
5. $[\sigma_j] \leftarrow \text{MULT}^*[\sigma_i]$ for $i \in \{1, \ldots, n\}$, $[\sigma] = \prod_{j=1}^{n}[\sigma_j]$;
6. $[i] = [\sigma][l]$;
7. return Rec$([i])$;

is detailed in Appendix A.

The Integer Implementation of Π_i. A different approach to sample an integer variate from a DL distribution measures the probability of taking N coins to obtain success. To realize this approach, the work of (Eriguchi et al., 2021) introduce TDL(p,N) as,

$$\Pr[L = i] = \begin{cases} p^{|i|}\frac{(1-p)}{(1+p)} & \text{if} |i| < N, \\ \frac{p^N}{1+p} & \text{if} |i| = N, \\ 0 & \text{otherwise}, \end{cases}$$

and show a mechanism that adds noise from TDL(p,N) to an integer-valued query is (ε, δ)-DP for $p = \exp(-\frac{\varepsilon}{\Delta})$ and $\delta \geq p^N \frac{1+p^{-\Delta}}{1+p}$. Recall that per GLM, $q_r(D) + ir$ is ε-DP, where i follows a DL distribution with $p = \exp(-r\frac{\varepsilon}{\Delta_r})$.

Protocol 5: The Laplace mechanism using DL(p).

Inputs: The FL representation of the public input r as $\langle v_r, p_r, 0, 0 \rangle$.
Output: The FL representation of the private value of $q_r(D) + ir$ as $\langle [v], [p], [z], [s] \rangle$.

1. $\langle [v_q], [p_q], [z_q], [s_q] \rangle \leftarrow \Pi_q$;
2. $\langle [v_1], [p_1], [z_1], [s_1] \rangle \leftarrow$ FLAdd $(\langle [v_q], [p_q], [z_q], [s_q] \rangle, \langle v_r, p_r - 1, 0, 0 \rangle)$;
3. $\langle [v_{gr}], [p_{gr}], [z_{gr}], [s_{gr}] \rangle \leftarrow$ EFLRound$\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle v_r, p_r, 0, 0 \rangle, 0)$;
4. $\langle [v_i], [p_i], [z_i], [s_i] \rangle \leftarrow$ DL(p);
5. $\langle [v_{ir}], [p_{ir}], [z_{ir}], [s_{ir}] \rangle \leftarrow$ FLMul$(\langle v_r, p_r, 0, 0 \rangle, \langle [v_i], [p_i], [z_i], [s_i] \rangle)$;
6. $\langle [v], [p], [z], [s] \rangle \leftarrow$ FLAdd$(\langle [v_{gr}], [p_{gr}], [z_{gr}], [s_{gr}] \rangle, \langle [v_{ir}], [p_{ir}], [z_{ir}], [s_{ir}] \rangle)$;
7. return Rec$(\langle [v], [p], [z], [s] \rangle)$;

The study of (Eriguchi et al., 2021) also presents a secure MPC protocol for sampling from TDL(p,N) using the idea of (Dwork et al., 2006) in which one shared biased coin is generated from d shared unbiased coins. Given p and N, the MPC of TDL(p,N) is implemented by Protocol 4, which operates on integers and requires $(5n+19)dN + 17N + 5n + 4$ interactive operations and 15 rounds to generate one DL sample. An extra interactive operation and round have been considered because of the Rec protocol, compared to the complexity reported by (Eriguchi et al., 2021).

4.4 The Laplace Mechanism Implementation

Protocol 5 is an MPC protocol for computing GLM using DL(p).

In Protocol 5, we assume the sub-protocol Π_q computes $q(D)$. Lines 2 and 3 round $q(D)$ to the nearest multiple of r. We note that to obtain $\frac{r}{2}$, we only need to subtract 1 from p_r since r is a power of two. Line 4 computes i, and line 5 scales the noise as ir. Line 6 calculates $q_r(D) + ir$. Finally, in line 7, the parties reconstruct the FL representation of $q_r(D) + ir$. As for complexity, Protocol 5 can be implemented using 132800+ Int(Π_q) interactive operations. For FLAdd in line 2, one operand provided in the clear reduces the number of interactive operations by 14 compared to the value in Table 1, while the number of rounds remains the same. FLAdd is detailed in Appendix A. Similarly, FLMul in line 5

Protocol 6: The Laplace mechanism using TDL(p,N).

Inputs: The FL representation of the public input r as $\langle v_r, p_r, 0, 0\rangle$, and the integer representation of N and p.
Output: The FL representation of the private value $q_r(D)+ir$ as $\langle[v],[p],[z],[s]\rangle$.

1. $\langle[v_q],[p_q],[z_q],[s_q]\rangle \leftarrow \Pi_q$;
2. $\langle[v_1],[p_1],[z_1],[s_1]\rangle \leftarrow$ FLAdd $(\langle[v_q],[p_q],[z_q],[s_q]\rangle, \langle v_r, p_r-1,0,0\rangle)$;
3. $\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle \leftarrow$ EFLRound$(\langle[v_1],[p_1],[z_1],[s_1]\rangle, \langle v_r, p_r, 0, 0\rangle, 0)$;
4. $[i] \leftarrow$ TDL(p,N);
5. $\langle[v_i],[p_i],[z_i],[s_i]\rangle \leftarrow$ Int2FL$([i],\gamma,l)$;
6. $\langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle \leftarrow$ FLMul$(\langle v_r, p_r, 0, 0\rangle, \langle[v_i],[p_i],[z_i],[s_i]\rangle)$;
7. $\langle[v],[p],[z],[s]\rangle \leftarrow$ FLAdd$(\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle, \langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle)$;
8. return Rec$(\langle[v],[p],[z],[s]\rangle)$;

also benefits from one operand provided in the clear, as previously discussed, reducing its complexity.

Protocol 6 is a variant of Protocol 5, which replaces TDL(p,N) with DL(p). In line 5 of Protocol 6, Int2FL converts the integer sample i to its FL representation. Protocol 6 can be realized with $(5n+19)dN+17N+5n+3940+\text{Int}(\Pi_q)$ interactive operations.

4.5 A Specialization of Π_q for Linear Queries

To simplify the presentation, we consider a class of queries defined by $q(D_1,\ldots,D_m) = q(D_1)+\cdots+q(D_m)$. For this, we outline our algorithm for computing the Laplace mechanism. The algorithm takes m real inputs $d_1,\ldots,d_m$, each representing the local execution of q on U_i's database D_i as $d_i = q(D_i)$. It also takes the resolution parameter r as an input. The algorithm computes and returns the real output $w = q_r(D)+ir$ by the following steps:

1. Compute $q(D)$ as $q(D) = \sum_{i=1}^{m} d_i$.
2. Round $q(D)$ to the nearest multiple of r.
3. Generate a random sample i distributed per the DL distribution with $p = \exp(-r\frac{\varepsilon}{\Delta_r})$.
4. Multiply i by r.
5. Add $q_r(D)$ to ir to obtain the result w.

We adopt the following approach to implement the algorithm in an MPC setting. We assume the users are connected to the computation parties by perfectly secure channels. Before computation begins, users initiate the $(\lfloor\frac{n-1}{2}\rfloor, n)$ sharing scheme to distribute shares

Protocol 7: A specialization of Laplace mechanism using DL(p).

Inputs: The FL representation of the private values d_i as $\langle[v_{d_i}],[p_{d_i}],[z_{d_i}],[s_{d_i}]\rangle$ for $i\in\{1,\ldots,m\}$, the public inputs r as $\langle v_r, p_r, 0, 0\rangle$ and the integer value p.
Output: The FL representation of the private value $q_r(D)+ir$ as $\langle[v],[p],[z],[s]\rangle$.

1. $\langle[v_q],[p_q],[z_q],[s_q]\rangle \leftarrow \langle[v_{d_1}],[p_{d_1}],[z_{d_1}],[s_{d_1}]\rangle$;
2. for $i=2$ to m do
   $\langle[v_q],[p_q],[z_q],[s_q]\rangle \leftarrow$ FLAdd$($
   $\langle[v_q],[p_q],[z_q],[s_q]\rangle, \langle[v_{d_i}],[p_{d_i}],[z_{d_i}],[s_{d_i}]\rangle)$;
3. $\langle[v_1],[p_1],[z_1],[s_1]\rangle \leftarrow$ FLAdd $(\langle[v_q],[p_q],[z_q],[s_q]\rangle, \langle v_r, p_r-1,0,0\rangle)$;
4. $\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle \leftarrow$ EFLRound$\langle[v_1],[p_1],[z_1],[s_1]\rangle, \langle v_r, p_r, 0, 0\rangle, 0)$;
5. $\langle[v_i],[p_i],[z_i],[s_i]\rangle \leftarrow$ DL(p);
6. $\langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle \leftarrow$ FLMul$(\langle v_r, p_r, 0, 0\rangle, \langle[v_i],[p_i],[z_i],[s_i]\rangle)$;
7. $\langle[v],[p],[z],[s]\rangle \leftarrow$ FLAdd$(\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle, \langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle)$;
8. return Rec$(\langle[v],[p],[z],[s]\rangle)$;

of their inputs among the computation parties, adhering to the honest majority assumption. The (n,n) sharing scheme may lead to inefficiencies, as FLAdd requires the multiplication protocol at certain stages. Once the shares are distributed among the computation parties, each party P_k has a share of all input values d_i for $k\in\{1,\ldots,n\}$ and $i\in\{1,\ldots,m\}$. The input values are represented as shared FL numbers, denoted as $\langle[v_{d_i}],[p_{d_i}],[z_{d_i}],[s_{d_i}]\rangle$ to indicate the FL representation of d_i. Additionally, to generate the noise i, we still rely on the honest majority assumption. Once the parties collectively compute the shared result $q_r(D)+ir$, they can reconstruct the final result by invoking the Rec protocol.

Protocol 7 and 8 are realizations of the Laplace mechanism for computing $q(D)$ using respectively DL(p) and TDL(p,N).

To assess the complexity of Protocol 7, we divide it into two main parts that can be executed in parallel to reduce the number of rounds. Part (1) computes $q_r(D)$ in lines 1-4, and part (2) computes ir in lines 5 and 6. Part (1) can be realized using $1262m+980$ interactive operations in $36\log m+68$ rounds when the additions in the for loop are performed in parallel. On the other hand, part (2) can be implemented using 129300 interactive operations in 1178 rounds. Under the assumption that $m<2^{30}$, the number of rounds in part (2) is the dominant factor in computing the total number (i.e., we disregard the number of rounds in part (1)). Considering FLAdd and Rec in lines 7 and 8,

Protocol 8: A specialization of Laplace mechanism using TDL(p,N).

Inputs: The FL representation of the private values d_i as $\langle[v_{d_i}],[p_{d_i}],[z_{d_i}],[s_{d_i}]\rangle$ for $i \in \{1,\ldots,m\}$, the public inputs r as $\langle v_r, p_r, 0, 0\rangle$ and the integer values N and p.
Output: The FL representation of the private value $q_r(D)+ir$ as $\langle[v],[p],[z],[s]\rangle$.

1. $\langle[v_q],[p_q],[z_q],[s_q]\rangle \leftarrow \langle[v_{r_1}],[p_{r_1}],[z_{r_1}],[s_{r_1}]\rangle$;
2. for $i = 2$ to m do
   $\langle[v_q],[p_q],[z_q],[s_q]\rangle \leftarrow$ FLAdd($\langle[v_q],[p_q],[z_q],[s_q]\rangle, \langle[v_{d_i}],[p_{d_i}],[z_{d_i}],[s_{d_i}]\rangle$);
3. $\langle[v_1],[p_1],[z_1],[s_1]\rangle \leftarrow$ FLAdd $(\langle[v_q],[p_q],[z_q],[s_q]\rangle, \langle v_r, p_r - 1, 0, 0\rangle)$;
4. $\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle \leftarrow$ EFLRound$\langle[v_1],[p_1],[z_1],[s_1]\rangle, \langle v_r, p_r, 0, 0\rangle, 0)$;
5. $[i] \leftarrow$ TDL(p,N);
6. $\langle[v_i],[p_i],[z_i],[s_i]\rangle \leftarrow$ Int2FL$([i],\gamma,l)$;
7. $\langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle \leftarrow$ FLMul$(\langle v_r, p_r, 0, 0\rangle, \langle[v_i],[p_i],[z_i],[s_i]\rangle)$;
8. $\langle[v],[p],[z],[s]\rangle \leftarrow$ FLAdd$(\langle[v_{gr}],[p_{gr}],[z_{gr}],[s_{gr}]\rangle, \langle[v_{ir}],[p_{ir}],[z_{ir}],[s_{ir}]\rangle)$;
9. return Rec$(\langle[v],[p],[z],[s]\rangle)$;

Protocol 7 requires $1262m+131540$ interactive operations and 1250 rounds.

To analyze the complexity of Protocol 8, we set $d = 50$ as recommended by (Eriguchi et al., 2021), $r = 2^{-k} = 2^{-10}$, $\varepsilon = 1$, and $\Delta_r = 1$ to execute TDL$(p = \exp(-r\frac{\varepsilon}{\Delta_r}), N)$. The number of interactive operations of Protocol 8 depends on the variable N. We find a suitable value for N by satisfying $\delta \geq p^N \frac{1+p^{-\Delta}}{1+p}$. After numerical analysis with $\varepsilon = 1$, $\Delta_r = 1$, and $r = 2^{-k} = 2^{-10}$, aiming for $\delta \leq 2^{-20}$, we found that $N \geq 14196$. Hence, we use $N = 14196$ for the complexity analysis of Protocol 8. Recall that $10 \leq k \leq 45$ is recommended by (Google'sDP, 2020). Note that larger values of N will be required for any $k > 10$ and $\varepsilon < 1$ to achieve $\delta = 2^{-20}$. Protocol 8 can be divided into two main parts that can be executed concurrently to save the number of rounds. Part (1) computes $q_r(D)$ in lines 1-4, and part (2) computes ir in lines 5-7. Part (1), like part (1) of Protocol 7, can be implemented using $1262m+980$ interactive operations and $36\log m+68$ rounds. On the other hand, part (2) can be implemented using $3549005m+13728000$ interactive operations and 44 rounds. Since the number of rounds of part (1) is the dominant factor, we disregard the number of rounds of part (2) while computing the total number. Considering FLAdd and Rec in lines 8 and 9, Protocol 8 requires overall $3550267m+13730000$ interactive operations and $36\log m+105$ rounds.

Both Protocols 7 and 8 have interactive operations of $O(m)$. Specifically, Protocol 7 requires $1262m+131540$ interactive operations and a constant of 1250 rounds. In comparison, Protocol 8 demands $3550267m+13730000$ interactive operations and $36\log m+105$ rounds. While Protocol 7 has a fixed round count, Protocol 8 realistically requires fewer rounds, given $m < 2^{30}$. The constants under big-O for Protocol 8 are substantially larger than Protocol 7, such that for $m = 100,000$ users, Protocol 8 needs about 2810 times more interactive operations than Protocol 7. Despite its fewer rounds, Protocol 8 has much higher communication complexity.

5 SECURITY ANALYSIS

Our protocols are secure against passive adversaries in the information-theoretic model. For malicious adversaries, the security of our protocols depends on the chosen verifiable secret sharing scheme. These claims are detailed in the following.

In our MPC setting, we established a linear secret sharing scheme that satisfies the t-secure property against coalitions of size up to t passive adversaries. Additionally, our setting relies on a secure multiplication protocol to ensure no information leakage occurs; the only data transmitted among the parties is secret shares. Moreover, we detailed the sub-protocols used in section 3.2, which have previously been demonstrated to uphold the t-secrecy property. Invoking Canetti's composition theorem (Canetti, 2000), one can infer that the composition of secure sub-protocols ensures the security of the overall solution. As is common in MPC literature, and without furnishing formal proof, we claim the existence of a simulator for our protocols. Such a simulator, when invoking simulators for the corresponding sub-protocols, would theoretically create an environment indistinguishable from the real protocol execution by the parties.

To ensure security against malicious adversaries, all parties must demonstrate that every computation step has been executed correctly. This is particularly important when any linear combination of shares is computed locally; each party must validate that it has accurately carried out each multiplication on its shares. If any party acts dishonestly or abandons the protocol, the remaining parties should be able to reconstruct their shares and proceed with the computation. A verifiable secret sharing (VSS) scheme can effectively address these concerns (Cramer et al., 2000). We can employ VSS schemes such as (Cramer et al., 2000) that offer security in the information-theoretic

model or VSS schemes such as (Pedersen, 1992) that ensure security only in the computational model. Additionally, when parties provide input values of a particular form, they must verify that their inputs adhere to these requirements. Such verification is crucial when implementing specific building blocks, such as RandInt. This verification can be achieved by existing protocols like those in (Peng and Bao, 2010).

6 AN ALTERNATIVE SOLUTION

One might suggest that each user locally rounds its input to the nearest integer and then shares an integer value with the computation parties. While such a method sidesteps the complexities of FL arithmetic in evaluating the function q, it may significantly undermine its accuracy. For instance, given m independent variables d_i where each d_i is uniformly distributed over $[0,1]$, the expected absolute error between the sum of the original variables and the sum of their rounded values is approximately $\sqrt{\frac{m}{12}}\sqrt{\frac{2}{\pi}}$. A detailed derivation of this result is as follows.

Given a variable d_i, the error introduced by rounding is defined as $e_i = d_i - \text{round}(d_i)$. This error lies in the range $[-0.5, 0.5]$, specifically for $d_i \in [0, 0.5)$, $e_i = d_i$ and for $d_i \in [0.5, 1]$, $e_i = d_i - 1$. The expected value of the square of the error is

$$E[e_i^2] = \int_0^{0.5} x^2 dx + \int_{0.5}^{1} (x-1)^2 dx = \frac{1}{12}.$$

The variance of the error is

$$Var(e_i) = E[e_i^2] - (E[e_i])^2,$$

since $E[e_i] = 0$, then $Var(e_i) = \frac{1}{12}$.

The standard deviation of the sum of independent errors e_i is

$$\sigma_{\text{sum}} = \sqrt{m \times Var(e_i)} = \sqrt{\frac{m}{12}}.$$

By Central Limit Theorem: For large m, the sum of the errors tends toward a normal distribution with mean 0 and standard deviation σ_{sum}. The expected absolute deviation for a standard normal distribution is $\sqrt{\frac{2}{\pi}}\sigma$. Combining these results, we arrive at

$$E[|e_1 + e_2 + \cdots + e_m|] \approx \sqrt{\frac{m}{12}}\sqrt{\frac{2}{\pi}}.$$

For $m = 100,000$, the error equals 72.836. Therefore, in contexts where precision is essential, these rounding inaccuracies can amplify, especially in subsequent computations (e.g., when factoring in the random noise introduced by the DP mechanisms), the combined error in q may escalate. This culmination of rounding error and noise-induced variability suggests that such rounding might not be a favorable approach if the accuracy of q is a priority.

7 CONCLUSION

This paper has bridged an important gap in the field by addressing the need for secure MPC of differentially private Laplace mechanisms designed explicitly for real-valued queries. Although prior works have provided MPC of DP for integer-valued queries, including real-valued queries is pivotal in many applications. Despite a few attempts to extend the MPC of DP to accommodate real-valued queries, such approaches are susceptible to side-channel attacks, thereby undermining privacy guarantees. In the following, we will discuss some avenues for future work.

While this paper does not address the optimization of Protocol 8, it does invite the research community to explore methods to enhance the protocol's efficiency. One potential direction is to make TDL(p,N) $(\varepsilon, \delta = 0)$-DP as $\delta > 0$ significantly increases the protocol's communication complexity. Moreover, improving the complexity of the FLLog2 primitive used in Protocol 7 represents a significant area for future work as the current implementation requires over 1000 rounds of computation (Eigner et al., 2014). While our study has thoroughly analyzed the complexity of our protocols, it is essential to emphasize the need for practical implementation to assess their real-world efficiency and applicability.

REFERENCES

Aliasgari, M., Blanton, M., Zhang, Y., and Steele, A. (2013). Secure computation on floating point numbers. *Network and Distributed System Security (NDSS) Symposium*.

Balcer, V. and Cheu, A. (2019). Separating local & shuffled differential privacy via histograms. *1st Conference on Information-Theoretic Cryptography (ITC 2020)*.

Balle, B., Bell, J., Gascón, A., and Nissim, K. (2019). The privacy blanket of the shuffle model. In *Proceedings of 39th Annual International Cryptology Conference (CRYPTO 2019)*, pages 638–667. Springer.

Bassily, R. and Smith, A. (2015). Local, private, efficient protocols for succinct histograms. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 127–135.

Beimel, A., Nissim, K., and Omri, E. (2008). Distributed private data analysis: Simultaneously solving how and what. In *Proceedings of 28th Annual International*

Cryptology Conference (CRYPTO), pages 451–468. Springer.

Ben-Or, M., Goldwasser, S., and Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 351–371.

Canetti, R. (2000). Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202.

Catrina, O. and De Hoogh, S. (2010a). Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer.

Catrina, O. and De Hoogh, S. (2010b). Secure multiparty linear programming using fixed-point arithmetic. In *Proceedings of 15th European Symposium on Research in Computer Security (ESORICS 2010)*, pages 134–150. Springer.

Chan, T. H., Shi, E., and Song, D. (2012). Optimal lower bound for differentially private multi-party aggregation. In *European Symposium on Algorithms*, pages 277–288. Springer.

Cheu, A., Smith, A., Ullman, J., Zeber, D., and Zhilyaev, M. (2019). Distributed differential privacy via shuffling. In *Proceedings of 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2019)*, pages 375–403. Springer.

Cramer, R., Damgård, I., and Maurer, U. (2000). General secure multi-party computation from any linear secret-sharing scheme. In *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT 2000)*, pages 316–334. Springer.

Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B., and Toft, T. (2006). Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. pages 4171–4186. Association for Computational Linguistics.

Devroye, L. (2006). Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. (2006). Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer.

Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407.

Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. (2014). Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 316–325.

Eriguchi, R., Ichikawa, A., Kunihiro, N., and Nuida, K. (2021). Efficient noise generation to achieve differential privacy with applications to secure multiparty computation. In *International Conference on Financial Cryptography and Data Security*, pages 271–290. Springer.

Gazeau, I., Miller, D., and Palamidessi, C. (2016). Preserving differential privacy under finite-precision semantics. *Theoretical Computer Science*, 655:92–108.

Google'sDP (2020). Secure Noise Generation.

Haney, S., Desfontaines, D., Hartman, L., Shrestha, R., and Hay, M. (2022). Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers. *arXiv preprint arXiv:2207.13793*.

Holohan, N. and Braghin, S. (2021). Secure random sampling in differential privacy. In *Computer Security – ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II*, page 523–542, Berlin, Heidelberg. Springer-Verlag.

Kasiviswanathan, S. P., Lee, H. K., Nissim, K., Raskhodnikova, S., and Smith, A. (2011). What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826.

Mironov, I. (2012). On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 650–661.

MPFR'slibrary (2021). The GNU MPFR library.

OpenDP (2021). samplers.rs – opendp.

Pedersen, T. P. (1992). Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum, J., editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg. Springer Berlin Heidelberg.

Peng, K. and Bao, F. (2010). An efficient range proof scheme. In *2010 IEEE second international conference on social computing*, pages 826–833. IEEE.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Shi, E., Chan, H., Rieffel, E., Chow, R., and Song, D. (2011). Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society.

Wang, T., Blocki, J., Li, N., and Jha, S. (2017). Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 729–745.

Wu, G., He, Y., Wu, J., and Xia, X. (2016). Inherit differential privacy in distributed setting: Multiparty randomized function computation. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 921–928. IEEE.

APPENDIX

A FL PROTOCOLS INTRODUCED BY (Aliasgari et al., 2013)

Protocol 9 rounds an FL number to the nearest integer depending on its mode parameter. Protocols 10 and 11 compute respectively the product and addition of two FL numbers as inputs.

Protocol 9: FLRound.

Inputs: The FL number $\langle [v_1],[p_1],[z_1],[s_1]\rangle$ and mode.
Output: The FL number $\langle [v],[p],[z],[s]\rangle$ which is the result of rounding the input to an integer.

1. $[a] \leftarrow \text{LTZ}([p_1],k)$;
2. $[b] \leftarrow \text{LT}([p_1],-l+1,k)$;
3. $\langle [v_2],[2^{-p_1}]\rangle \leftarrow \text{Mod2m}([v_1],l,-[a](1-[b])[p_1])$;
4. $[c] \leftarrow \text{EQZ}([v_2],l)$;
5. $[v] \leftarrow [v_1]-[v_2]+(1-[c])[2^{-p_1}](\text{XOR}(\text{mode},[s_1]))$;
6. $[d] \leftarrow \text{EQ}([v],2^l,l+1)$;
7. $[v] \leftarrow [d]2^{l-1}+(1-[d])[v]$;
8. $[v] \leftarrow [a]((1-[b])[v]+[b](\text{mode}-[s_1]))+(1-[a])[v_1]$;
9. $[s] \leftarrow (1-[b]\text{mode})[s_1]$;
10. $[z] \leftarrow \text{OR}(\text{EQZ}([v],l),[z_1])$;
11. $[v] \leftarrow [v](1-[z])$;
12. $[p] \leftarrow ([p_1]+[d][a](1-[b]))(1-[z])$;
13. return $\langle [v],[p],[z],[s]\rangle$;

Protocol 10: FLMul.

Inputs: The FL numbers $\langle [v_1],[p_1],[z_1],[s_1]\rangle$ and $\langle [v_2],[p_2],[z_2],[s_2]\rangle$.
Output: The FL number $\langle [v],[p],[z],[s]\rangle$ as the result of the product of the inputs.

1. $[v] \leftarrow [v_1][v_2]$;
2. $[v] \leftarrow \text{Trunc}([v],2l,l-1)$;
3. $[b] \leftarrow \text{LT}([v],2^l,l+1)$;
4. $[v] \leftarrow \text{Trunc}(2[b][v]+(1-[b])[v],l+1,1)$;
5. $[c] \leftarrow \text{OR}([z_1],[z_2])$;
6. $[v] \leftarrow \text{XOR}([s_1],[s_2])$;
7. $[p] \leftarrow ([p_1]+[p_2]+l-[b])(1-[z])$;
8. return $\langle [v],[p],[z],[s]\rangle$;

Protocol 11: FLAdd.

Inputs: The FL numbers $\langle [v_1],[p_1],[z_1],[s_1]\rangle$ and $\langle [v_2],[p_2],[z_2],[s_2]\rangle$.
Output: The FL number $\langle [v],[p],[z],[s]\rangle$ as the result of the addition of the inputs.

1. $[a] \leftarrow \text{LT}([p_1],[p_2],k)$;
2. $[b] \leftarrow \text{EQ}([p_1],[p_2],k)$;
3. $[c] \leftarrow \text{LT}([v_1],[v_2],l)$;
4. $[p_{max}] \leftarrow [a][p_2]+(1-[a])[p_1]$;
5. $[p_{min}] \leftarrow (1-[a])[p_2]+[a][p_1]$;
6. $[v_{max}] \leftarrow (1-[b])([a][v_2]+(1-[a])[v_1])+[b]([c][v_2]+(1-[c])[v_1])$;
7. $[v_{min}] \leftarrow (1-[b])([a][v_1]+(1-[a])[v_2])+[b]([c][v_1]+(1-[c])[v_2])$;
8. $[s_3] \leftarrow \text{XOR}([s_1]+[s_2])$;
9. $[d] \leftarrow \text{LT}(l,[p_{max}]-[p_{min}],k)$;
10. $[2^{\Delta}] \leftarrow \text{Pow2}((1-[d])([p_{max}]-[p_{min}]),l+1)$;
11. $[v_3] \leftarrow 2([v_{max}]-[s_3])+1$;
12. $[v_4] \leftarrow [v_{max}][2^{\Delta}]+(1-2[s_3])[v_{min}]$;
13. $[v] \leftarrow ([d][v_3]+(1-[d])[v_4])2^l\text{Inv}([2^{\Delta}])$;
14. $[v] \leftarrow \text{Trunc}([v],2l+1,l-1)$;
15. $[u_{l+1}],\ldots,[u_0] \leftarrow \text{BitDec}([v],l+2,l+2)$;
16. $[h_0],\ldots,[h_{l+1}] \leftarrow \text{PreOr}([u_{l+1}],\ldots,[u_0])$;
17. $[p_0] \leftarrow l+2-\sum_{i=0}^{l+1}[h_i]$;
18. $[2^{p_0}] \leftarrow 1+\sum_{i=0}^{l+1}2^i(1-[h_i])$;
19. $[v] \leftarrow \text{Trunc}([2^{p_0}][v],l+2,2)$;
20. $[p] \leftarrow [p_{max}]-[p_0]+1-[d]$;
21. $[v] \leftarrow (1-[z_1])(1-[z_2])[v]+[z_1][v_2]+[z_2][v_1]$;
22. $[z] \leftarrow \text{EQZ}([v],l)$;
23. $[p] \leftarrow ((1-[z_1])(1-[z_2])[p]+[z_1][p_2]+[z_2][p_1])(1-[z])$;
24. $[s] \leftarrow (1-[b])([a][s_2]+(1-[a])[s_1])+[b]([c][s_2]+(1-[c])[s_1])$;
25. $[s] \leftarrow (1-[z_1])(1-[z_2])[s]+(1-[z_1])[z_2][s_1]+[z_1](1-[z_2])[s_2]$;
26. return $\langle [v],[p],[z],[s]\rangle$;