Accelerating Multiparty Noise Generation Using Lookups

Fredrik Meisingseth[1], Christian Rechberger[1], and Fabian Schmid[1]

Graz University of Technology, Graz, Austria
firstname.lastname@tugraz.at

Abstract. There is growing interest in combining Differential Privacy (DP) and Secure Multiparty Computation (MPC) to protect distributed database queries from both computational parties and those observing the result. This requires implementing both query evaluation and noise generation within MPC. While secure query evaluation is well-supported by existing MPC techniques, generating noise efficiently remains a challenge due to the nonlinearity of common sampling algorithms. We propose a new approach for multiparty noise sampling using recent advances in MPC lookup table (LUT) evaluations. Our method is distribution-agnostic and maps a cheaply sampled index to a target noise distribution via oblivious LUT evaluation. We demonstrate the flexibility by approximating the discrete Laplace and Gaussian distributions to a negligible statistical distance. Our implementation, based on 3-party replicated secret sharing (RSS), achieves sub-kilobyte communication and millisecond-level computation. Per 1000 discrete Laplace or Gaussian samples, we require just 362 bytes of communication and under 1 ms per party (semi-honest setting). With recent batched multiplication checks, the amortized malicious setting adds less than 1 byte and 10 ms per sample. Our open-source implementation also extends MAESTRO-style LUT trade-offs, offering potential independent value.

Keywords: Differential Privacy, Multiparty Noise Sampling, Lookup Tables

1 Introduction

The literature on multiparty noise sampling for differential privacy (DP) was initiated in the 2006 paper by Dwork, Kenthapadi, McSherry, Mironov and Naor [11]. As [11] was published at a time when both DP and secure multiparty computation (MPC) were far away from practical implementations, it is only in recent years that more multiparty sampling protocols have been designed [8,12,20,22,28,14]. In this work we propose a new approach for multiparty noise sampling. Most of the existing works implement either a version of the discrete Laplace mechanism (i.e. geometric mechanism [15]) or the discrete Gaussian mechanism [7] (alternatively, the binomial mechanism [11]). There are three main sampling approaches in the literature:

- *‘Bit-wise’ sampling([8,12,22,28]):* This approach stems from [11], where it is shown that each bit of the noise sample can be generated independently of all other bits. This means that sampling discrete Laplace and discrete Gaussian distributions can be reduced to sampling a large number of biased coins.
- *Rejection sampling([20,14]):* The idea is to sample a cheap proposal distribution and then reject based on the outcome of a biased coin. This classic sampling method entered the DP literature in [7].

– *Distributed sampling([14]):* This approach is that each party locally samples a noise distribution such that the sum of the noises has the desired distribution.

Rejection sampling can efficiently be used to turn a sample of a discrete Laplace distribution into a sample of a discrete Gaussian [7]. Distributed sampling, though efficient, is unsuitable for malicious security and suboptimal even in the semi-honest setting, as parties can remove their added noise after seeing the output. This leads to a worse utility/privacy tradeoff in the case that both the computational parties and outsiders learn the mechanism output. We therefore compare only to the other two approaches.[1] Existing work differ on the side of MPC in terms of number of parties, security setting and techniques applied. We refer discussion of such matters to Section 5, since the main novelty of our work lies primarily in the sampling approach we propose rather than the MPC techniques used to implement it. None of the multiparty sampling protocols sample the noise distribution exactly (due to the noise distributions being impossible to sample exactly in strict finite time) but rather approximate it up to a small *approximation error* in terms of statistical distance. We denote the approximation error by δ_{appr} and upper bound it by $2^{-\lambda}$ for some $\lambda \in \mathbb{N}$.

1.1 Overview of Our Construction

Our new multiparty noise sampling protocol builds on lookup tables (LUTs), which have recently gained traction in the MPC literature. An overview of the construction is shown in Figure 1, where parameters are illustrated as arrows from above and inputs/outputs are illustrated using horizontal arrows. We assume access to the functionalities $\mathcal{F}_{\texttt{rand}}, \mathcal{F}_{\texttt{I}}$ and $\mathcal{F}_{\texttt{LUT}}$ that allow for the secure generation of random bits, sampling of an index following a given distribution and evaluation of a lookup table, respectively.

The basic idea is to build a public LUT mapping indices to noise values, then the parties jointly sample an index where the LUT is obliviously evaluated. This trivially gives a good approximation if the index and target noise distributions are close or the LUT is large enough. In the extreme cases, one should note, however, that a uniform index likely requires an impractically large LUT, while sampling directly from the target makes the LUT unnecessary. We show that the LUT can indeed be small enough to allow efficient evaluation and simultaneously give a small approximation error for common noise distributions. Our construction essentially adjusts the index distribution I by mapping multiple indices to the same noise value. Its approximation error is, however, determined by the target distribution Z and LUT size. When instantiating the LUT-based sampling approach for a given target distribution Z (e.g. discrete Laplace or Gauss), there are **three main design considerations**:

1. *How should we generate/fill the LUT?*
2. *What index distribution should we use?*
3. *How should we evaluate the LUT in MPC?*

[1] One should note that inversion-sampling is unlikely to be competitive, since it requires multiparty evaluations of logarithms.[14]

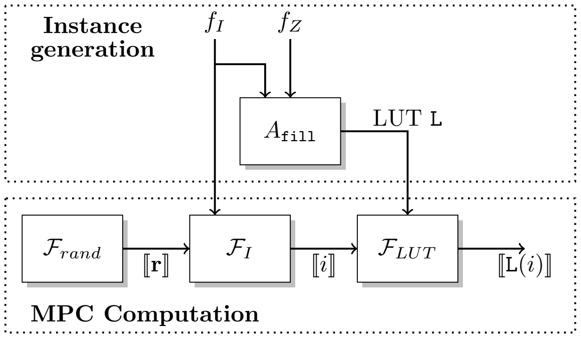

Fig. 1: Schematic overview with target distribution Z, index distribution I and respective mass functions f_Z, f_I.

These decisions are closely intertwined, as some choice may constrain another, such as the index distribution determining how large the LUT needs to be. We have **two main criteria** for the final protocol:

1. *The approximation error should be negligible.*
2. *The protocol should be efficient.*

The first criterion dictates the choice of how the LUT is filled and what index distribution I is used, but is agnostic to the choice of LUT evaluation protocol. The efficiency of the protocol, however, depends on the choice of I and LUT-evaluation method, but not the LUT-filling method, since the LUT itself is public. Regarding efficiency, we do not make a distinction between offline (preprocessing) and online costs and instead consider total costs. This is because in an operational setting, where the sampled noise is used for DP, the entirety of the noise generation is most reasonably to be seen as preprocessing, as it is independent of the query evaluation part of the DP mechanism, and can thus be done beforehand.

1.2 Contributions

– Our main contribution is a LUT-based sampling approach comprised of plain LUT generation, MPC index sampling, and LUT evaluation. We give a concrete protocol, prove its security in the UC security framework, and show it can approximate discrete Laplace and Gaussian distributions up to a negligible error.
– To demonstrate practical efficiency, we provide an open-source implementation[2] using 3-party honest-majority replicated secret sharing. We evaluate runtime and network costs under malicious and semi-honest security, for both single and batched samples across various network settings. Compared to the state of the art, our approach reduces amortized network costs and runtime by at least 1 000× and 10×, respectively.

1.3 Security Model

Our sampling approach is agnostic to the underlying MPC scheme and lookup strategy, but the following properties hold for our concrete instantiation. We use the UC framework for security [5]

[2] https://anonymous.4open.science/r/accelerating_mpc_noise_sampling-448D/README.md

and operate in the setting of static corruptions, security-with-abort, and honest-majority. We formulate ideal functionalities and security arguments using the Simple UC (SUC) framework [6], which implies security in the standard UC framework. The security of our protocol is computational rather than statistical, due to the use of pseudorandom secret sharing [9] to allow non-interactive generation of pseudorandom coins. As it is computationally secure, a DP mechanism based on our protocol gives computational differential privacy (CDP) [1,23] guarantees.

Outline Next, in Section 2, we present notation and preliminaries related to RSS and LUT evaluation protocols. In Section 3, we provide the specification and design considerations of our construction and our adapted lookup protocol. In Section 4, we outline concrete security and privacy considerations regarding the concrete instantiation and the underlying protocol. Then, in Section 5, we discuss our implementation and compare its performance to related works. Finally, in Appendices A and B, we list ideal functionalities and sub-protocols that our protocols build upon.

1.4 Recent Related Work

In an independent concurrent work, Franzese et al. [13] also propose a noise sampling strategy based on LUTs. Whereas their motivation is similar to ours, their protocol largely differs from ours in terms of pipeline, the primitives relied upon, and the envisioned use case. Additionally, it is implemented using different MPC tools and in different security settings, leading to fundamental advantages and disadvantages. First, instead of a single large table and a non-uniform index, their protocol uses a uniform index and a series of tables evaluated sequentially. Second, a different technique for LUT evaluation is used, which we discuss in Section 2.2. As a result, their protocol has a significantly higher communication cost (on the same order of magnitude as previous sampling protocols) but scales very well to larger numbers of parties, as we discuss in Section 5.2. Third, their techniques result in semi-honest security with dishonest majority, whereas ours give malicious security with honest majority.

2 Preliminaries and Notation

We denote the statistical (total-variation) distance between two distributions Z, Z' by $SD(Z, Z')$. We denote algorithms by A and protocols by π, each with some identifying subscript. For distributions, the capital letter denotes the distribution itself, and the same lowercase letter denotes a sample. If Z is a distribution (with sample z), then π_Z is a (potentially approximate) sampling protocol for Z. For a discrete probability distribution Z, we denote its probability mass function (pmf) by f_Z. We denote the Bernoulli distribution with parameter $p \in [0,1]$ as $Ber(p)$, i.e. the distribution with pmf $f_Z(z) := p^z(1-p)^{1-z}$ for $z \in \{0,1\}$. We denote the discrete Laplace distribution [15,17] with parameter $p \in (0,1)$ as $DLap(p)$ and its pmf is $f_Z(z) := \frac{1-p}{1+p}p^{|z|}$ for $z \in \mathbb{Z}$. We denote the discrete Gaussian distribution [7] by $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ and its pmf is $f_Z(z) := \frac{e^{-z^2/(2\sigma^2)}}{\sum_{y\in\mathbb{Z}} e^{-y^2/(2\sigma^2)}}$ for $z \in \mathbb{Z}$. We denote the indicator function by $\mathbb{1}$. By $\mathbb{F}_{2^\kappa} = \mathbb{Z}_2[X]/g(X)$, we denote the binary extension field where g is an irreducible polynomial with degree κ. Further, we denote by w the bit-width of the elements in the lookup table. In practice, it will be most convenient to work only with elements in the computation domain and set $w = \kappa$. For a positive integer m, $[m]$ denotes $\{0, 1, \ldots, m-1\}$.

2.1 Replicated Secret Sharing

Our protocol operates over $\mathbb{F}_{2^\kappa}$. Thus, the final results of our computation are in the extension field or, by local bit extraction, in the binary domain. However, these shares can be cheaply transformed to the arithmetic domain, as shown, e.g., in [24], resulting in bounded signed integers in $\mathbb{Z}_{2^\kappa}$. For an element $x \in \mathbb{F}_{2^\kappa}$, $[\![x]\!]$ denotes a replicated secret share [19] and $\langle\!\langle x \rangle\!\rangle$ an additive share of x. Replicated secret sharing (RSS) is a threshold variant of additive secret sharing. Each party has multiple shares, leading to the following improvements. RSS in the binary extension field has active security in an honest-majority setting when evaluating addition, bit-decomposition, and bit-recomposition. For the multiplication and dot-product evaluations, the parties need to verify the validity of the resulting share when considering active corruptions. Performance-wise, RSS is typically used in a computational setting, relying on the security of PRF evaluation. Shares of pseudorandom elements can be generated using pre-shared PRF keys, without interaction. The generation of shares of 0, multiplication of two shares and dot-products of arbitrary length all cost one round of interaction, sending one element in $\mathbb{F}_{2^\kappa}$ (i.e., κ bits of communication).

2.2 Lookup Tables in MPC

Recently, there has been a rising interest in MPC lookup table evaluation. A lot of these LUT evaluation approaches follow a similar blueprint and only differ in a preprocessing algorithm for random one-hot vector generation, which we denote as $\pi_{\texttt{RndOhv}}$. This protocol provides the parties with a secret shared tuple $([\![r]\!], [\![e^{(r)}]\!]) \leftarrow \pi_{\texttt{RndOhv}}(2^k)$, where r is uniform in $\mathbb{Z}_{2^k}$ and $e^{(r)}$ is a one-hot vector of length 2^k which is 1 at index r. Given such a preprocessing protocol, we can evaluate the LUT at position x in the online phase by computing the difference $[\![x]\!]-[\![r]\!]$, rotating the vector by the difference such that the one is at position x and computing the inner product $[\![\texttt{L}(x)]\!] \leftarrow \langle [\![e^{(x)}]\!], \texttt{L} \rangle$. There are constructions in the 2-party-plus-helper setting for Function Secret-Sharing (FSS) in the binary domain [3] and over integer rings [27]. Without a helper, there are 2-party protocols based on ABY-2 optimizing for low online communication [18], lower total communication [10], or a trade-off between online and total communication [4]. Further, Heath et al. propose a solution for Garbled Circuits (GC) [16], that requires only logarithmically many ciphertexts. Franzese et al. [13] propose a one-hot vector generation based on mixed-mode MPC, leveraging threshold homomorphic encryption to scale the number of parties. In MAESTRO [25], the authors propose a solution for RSS and additionally provide an interesting online-to-total communication trade-off. While we extend this trade-off to minimize total communication, our construction can be realized with any lookup table protocol in $\mathbb{F}_{2^\kappa}$ or $\mathbb{Z}_{2^\kappa}$.

3 Instantiation and Design Rationale

In this section, we instantiate the LUT-based sampling approach, outline the design constraints of the building blocks and detail our concrete sampling protocol π_Z. The protocol is specified in Figure 2. We set the index to consist of independent bits with some given biases and the ideal functionality for sampling such indices is $\mathcal{F}_{Ber}$ (Figure 4). That is, we set $\mathcal{F}_I = \mathcal{F}_{Ber}$. We formulate the sampling protocol π_Z with respect to the ideal functionalities $\mathcal{F}_{\texttt{Ber}}$, $\mathcal{F}_{\texttt{LUT}}$, $\mathcal{F}_{weakMult}$ and $\mathcal{F}_{verify}$, and this reduces the protocol implementation to that of subprotocols which realize them. Protocols for weak multiplication and verification of multiplications are basic building blocks that the other protocols rely upon, and for them, we use existing protocols from the MPC literature [2,25]. In the coming subsections, we give an algorithm for filling the LUT and protocols for index sampling and LUT evaluations, followed by an analysis of these. The formal statements of the subprotocols, with security arguments, and ideal functionalities that the protocols use are discussed in the appendix.

We employ two tricks to increase the efficiency of our instantiation. First, we truncate the target distribution to a bound $[-B, B]$ for some small integer B. This is done to speed up the filling of the LUT and to reduce the computational errors arising in that process. The truncation, of course, introduces a truncation error in our approximation, which needs to be accounted for. Second, in order to speed up the evaluation of the LUT, we set the LUT to approximate an (adjusted) one-sided version of the truncated discrete Laplace or Gaussian distributions where $f_{\mathtt{L}(I)}(z)$ approximates $f_Z(z) + f_Z(-z)$ for all non-zero $z \leq B$ and $f_{\mathtt{L}(I)}(0)$ approximates $f_Z(0)$. Let Z_{OS} denote this adjusted one-sided target distribution that $\mathtt{L}(I)$ approximates. Then we get an approximation of Z by assigning the sign of $\mathtt{L}(I)$ at random, leading to an output distribution of

$$Out_{\pi_Z} := \begin{cases} \mathtt{L}(I), \textit{ with probability } 1/2, \\ -\mathtt{L}(I), \textit{ otherwise.} \end{cases}$$

This trick, utilizing the symmetry of the target distribution, is commonly used in noise sampling protocols [12,20,7].

Protocol π_Z

Parameters: LUT $\mathtt{L} \in \mathbb{Z}_{2^w}^{2^k}$, bias parameter p_{Ber}, number of biased index bits $l' \leq k$.
Input: None.
Output: Shared sample $[\![z]\!]$.

```
1 :  Sample index: $[\![i]\!] \leftarrow \mathcal{F}_{\mathtt{Ber}}(p_{Ber}, k, l')$
2 :  Evaluate $\mathtt{L}$: $[\![z']\!] \leftarrow \mathcal{F}_{\mathtt{LUT}}(\mathtt{L}, [\![i]\!])$
3 :  Choose sign of the noise: $[\![z]\!] \leftarrow \mathcal{F}_{\mathtt{weakMult}}([\![z']\!], [\![b]\!])$ with $[\![b]\!] \leftarrow \mathcal{F}_{\mathtt{rand}}(\mathbb{F}_2)$.
4 :  Verify the multiplication using $\mathcal{F}_{\mathtt{verify}}$.
5 :  return $[\![z]\!]$
```

Fig. 2: The LUT-based sampling protocol.

3.1 Filling the LUT

The goal of the LUT filling is to construct a specific $\mathtt{L}$ that minimizes the approximation error $\delta_{appr} = SD(Z, Out_{\pi_Z})$, given a fixed index distribution I. The general problem of filling the LUT with minimal approximation error given arbitrary Z and I is quite similar to the bin packing problem and thus efficient error-optimal strategies intuitively appear unlikely to be found. Fortunately, the table $\mathtt{L}$ is public, so $A_{\mathtt{fill}}$ can be run by one party locally and the other parties can then upper bound the resulting approximation error numerically. We discuss in detail how to bound the approximation error in Section 4.1. In the following, we propose $A_{\mathtt{fill}}$ that results in satisfactory approximation errors for the discrete Laplace and Gaussian distributions. Our algorithm constructs $\mathtt{L}$ using a combination of a greedy initial step and an exhaustive step at the end, as outlined in Figure 3. Let $f_{Z_{OS}}$,f_I be the pmf's of Z_{OS} and I, respectively, and throughout, let $\mathtt{L}(i)$ denote the i'th element of the current version of $\mathtt{L}$. The filling algorithm $A_{\mathtt{fill}}$ proceeds as follows. First, we set $\mathtt{L}$ to be empty (all cells contain some blank symbol $\perp$) and consider each index $i \in \mathbb{F}_{2^k}$ iteratively in descending order according to mass $f_I(i)$. Similarly, for each i, we consider each $z \in [0, B]$ in descending order according to the target mass $f_{Z_{OS}}(z)$ and keep track of the total mass of z in the current version of $\mathtt{L}$, with $tot_z := \sum_i f_I(i) \cdot \mathbb{1}\{\mathtt{L}(i) = z\}$. If $f_{Z_{OS}}(z) \geq tot_z + f_I(i)$ then we set $\mathtt{L}(i) \leftarrow z$ and move to the next i, else we move to the next z. At the end of this process, $\mathtt{L}$ likely still has some empty cells, but assigning any of those cells to any value z results in $tot_z > f_{Z_{OS}}(z)$. To minimize the approximation error introduced by this final step, we iterate through the empty cells (after descending mass f_I) and assign each to the z such that $tot_z - f_{Z_{OS}}(z)$ is minimized. We do this by exhaustive search over z.

Algorithm $A_{\mathtt{fill}}$

Input: Pmf's $f_{Z_{OS}}, f_I$ of the target and index distributions, bound B and LUT size $N = 2^k$
Output: Public lookup table $\mathtt{L}$.

```
1 :  $\mathtt{L} \leftarrow [\perp]^N$ // *array of length $N$ with all entries $\perp$*
2 :  **for** $i \in \mathbb{F}_{2^k}$ sorted by descending $f_I(i)$
3 :    **for** $z \in [0, B]$ sorted by descending $f_{Z_{OS}}(z)$
4 :      $tot_z \leftarrow \sum_j f_I(j) \cdot \mathbb{1}\{\mathtt{L}(j) = z\}$
5 :      **if** $f_{Z_{OS}}(z) \geq tot_z + f_I(i)$
6 :        **then** $\mathtt{L}(i) \leftarrow z$ and skip to next $i$
7 :  **for** $i$ such that $\mathtt{L}(i) = \perp$
8 :    $z^* \leftarrow \arg\min_z \left(\sum_j f_I(j) \cdot \mathbb{1}\{\mathtt{L}(j) = z\} - f_{Z_{OS}}(z)\right)$
9 :    $\mathtt{L}(i) \leftarrow z^*$
10 : **return** $\mathtt{L}$
```

Fig. 3: The greedy LUT filling algorithm $A_{\mathtt{fill}}$.

3.2 Index Sampling

In practice, we seek an index distribution I that incurs only a very small communication cost but has a somewhat large mass difference between the least and most likely indices. We choose to simply cheaply bias some of the bits of the index and let the rest of the bits be uniform. Concretely, we let the first $l' \leq k$ bits of the index i have distribution $Ber(p_{Ber})$, with $p_{Ber} = 2^{-c}$ for some natural c, and let the remaining $k - l'$ index bits be uniform. The ideal functionality for sampling indices with this distribution is given in Figure 4. The protocol we use to realize this functionality proceeds simply by the parties generating shares of c independent fair coins per biased bit in the index and then multiplying these shares together. Then i is simply the concatenation of these biased bits and the remaining fair bits. Formally, the protocol consists solely of calls to ideal functionalities $\mathcal{F}_{\texttt{rand}}, \mathcal{F}_{\texttt{weakMult}}, \mathcal{F}_{\texttt{verify}}$ (discussed in appendix) for uniform random values, multiplication, and verification of those multiplications, respectively. We now shortly explain how

Functionality $\mathcal{F}_{\texttt{Ber}}$

Parameters: Bias parameter $p = 2^{-c}$ with $c \in \mathbb{N}$, total number of samples $k \in \mathbb{N}$ and number of biased samples $l' \leq k$.
Input: None.
Output: $[\![\mathbf{b}]\!]$ where $|\mathbf{b}| = k$ with each b_j with $j \in [l']$ is drawn independently from $Ber(p)$ and the remaining bits b_j are unbiased.

---

1 : $\mathcal{F}_{\texttt{Ber}}$ receives the corrupted parties' shares $[\![\mathbf{b}]\!]_C$ from the adversary.

2 : $\mathcal{F}_{\texttt{Ber}}$ samples $\mathbf{r}^0, \ldots, \mathbf{r}^{c-1}$ uniformly from $\mathbb{F}_2^{l'}$, sets $b_j \leftarrow \prod_{j'=0}^{c-1} r_j^{j'}$ for $j \in [l']$. $\mathcal{F}_{\texttt{Ber}}$ samples b_j uniformly from $\mathbb{F}_2$ for $j \in \{l', \ldots, k-1\}$.

3 : $\mathcal{F}_{\texttt{Ber}}$ computes the honest parties' shares $[\![\mathbf{b}]\!]_H$ of r to be consistent with $[\![\mathbf{b}]\!]_C$ and sends them to the honest parties.

Fig. 4: The ideal functionality for sampling fair or biased coins.

each of these basic functionalities can be realized with RSS. Given pre-shared PRF keys from a setup phase, the parties can generate shares of pseudorandom fair coins (thus realizing $\mathcal{F}_{\texttt{rand}}$) without interaction. Weak multiplication (i.e., multiplication where the adversary can introduce an arbitrary additive error) can also easily be done in RSS by local multiplication of shares, resulting in additive shares which are then turned back into replicated shares. Conveniently, the sampling of fair coins and multiplying them can trivially be done in parallel by simply sampling uniform random bits as an element of $\mathbb{F}_2^{l'}$, computing the multiplication in the smaller field and recomposing into $\mathbb{F}_{2^\kappa}$ locally. For the realization of $\mathcal{F}_{verify}$, we rely on the protocol of [25], which we denote as π_{verify}, that verifies batches of products at once. Since this protocol is restricted to a maximum batch size, we outline arising constraints in Section 4.2. Given these subprotocols, the index sampling protocol π_I realizes the canonical packing of bits sampled from $Ber(p_{Ber})$ into $\mathbb{F}_{2^\kappa}$. It is easy to see that these two steps of π_I can be done with $(c-1)l'$ bits of communication (caused by the re-sharing of the multiplication results).

3.3 Lookup Table Evaluation

In our construction, the wish for a large LUT (for the purpose of a low approximation error) and the wish for a small LUT for efficiency clearly compete against each other. Luckily, the choice to consider efficiency in terms of total communication throughout the protocol (not including sending the LUT to all parties in the beginning) allows us to aggressively utilize trade-offs between traditionally considered online and offline phases. In the next section, we present a new perspective on such trade-offs and use that to extend existing techniques to make larger LUTs practical for our usage.

Higher-Dimensional LUTs. Optimizing for total communication, we start by noting that the basic blueprint for LUT evaluation, as outlined in the preliminaries, has a rather extreme offline-to-online trade-off, with exponential preprocessing and linear online costs. Given that, we extend an idea from MAESTRO [25] where they reduce the preprocessing cost with small additional cost in the online phase. We see in the following that this is only feasible for secret sharing schemes that allow for dot-product computation independent of the vector size (e.g., Shamir Secret Sharing [26] or RSS). We phrase this explanation with a random index for simplicity, and adapting the protocol to use a specific index is straightforward. The first step is to sample two one-hot vectors $(\llbracket r' \rrbracket, \llbracket e^{(r')} \rrbracket)$, $(\llbracket r'' \rrbracket, \llbracket e^{(r'')} \rrbracket) \leftarrow \mathcal{F}_{\texttt{RndOHV}}(2^{k/2})$ and compute the combined index by concatenating the partial index bits $\llbracket r \rrbracket \leftarrow \llbracket r' || r'' \rrbracket$, which requires no interaction. Then, one extends the individual one-hot vectors into

$$\llbracket \mathbf{a} \rrbracket \leftarrow (\underbrace{\llbracket e_0^{(r')} \rrbracket, \cdots, \llbracket e_0^{(r')} \rrbracket}_{\times \frac{k}{2}}, \ldots, \llbracket e_{k/2-1}^{(r')} \rrbracket, \ldots, \llbracket e_{k/2-1}^{(r')} \rrbracket)$$

$$\llbracket \mathbf{b} \rrbracket \leftarrow (\llbracket e_0^{(r'')} \rrbracket, \cdots, \llbracket e_{k/2-1}^{(r'')} \rrbracket, \ldots, \llbracket e_0^{(r'')} \rrbracket, \ldots, \llbracket e_{k/2-1}^{(r'')} \rrbracket).$$

The LUT is then evaluated as $\langle \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket \circ \texttt{L} \rangle$, where $\circ$ represents the Hadamard product. This approach comes at an additional cost of 1 round and w bits (with w being the bit-length of an element in $\texttt{L}$) of online communication. It does, however, reduce the offline communication from $2^k - k - 1$ to $2(2^{k/2} - \frac{k}{2} - 1)$ bits. Since $\texttt{L}$ is public, $\llbracket \mathbf{b} \rrbracket \circ \texttt{L}$ incurs no communication, leaving only the final dot product.

At a high level, one can think of the above tensoring approach as a two-dimensional lookup, where r' and r'' represent the row and column indices. With this point of view, we extend the approach to lookup tables of higher dimensions. Starting in the three-dimensional case (illustrated in Figure 15 in Appendix B), which is also the setting that we use in the next section. We sample 3 one-hot vectors of size $k/3$ – one for the row, column, and layer of the cell index. The first product of vector and table is still free (being a local operation), but it only removes one dimension of the table, leaving us with a secret shared matrix and two remaining one-hot vectors. Collapsing the second layer incurs $2^{k/3}$ parallelizable dot-product evaluations and results in a final secret shared vector, which in turn requires one more dot product.

Generally, computing the cost for d dimensions, we get that the bit size of the one-hot vectors is divided by d and there are d vectors to produce, leaving us with $d(2^{k/d} + \frac{k}{d} - 1)$ bits of communication in the preprocessing. However, when we increase the dimensionality, it has an exponential impact on the number of dot products that we need to compute in the online phase. Given that the first dimension can be collapsed 'for free' (incurring only computational cost), we arrive at $\sum_{j=0}^{d-2} 2^{j\frac{k}{d}}$ dot-products in $d-1$ rounds. Having table elements of bit-width w, we arrive at a total communication cost of

$$d\left(2^{k/d} - \frac{k}{d} - 1\right) + w\left(\sum_{j=0}^{d-2} 2^{j\frac{k}{d}}\right). \tag{1}$$

The choice of d which minimizes this total cost, is dependent on the values of k and w. In our setting of $k = 24$ and $w = 8$, setting $d = 3$ incurs 34% of the total communication of the previous method with $d = 2$, saving 7425 bits of preprocessing but incurring 2048 bits of online communication. We note that whilst we base our implementation on MAESTRO [25] style lookup tables, the method of increasing the dimensionality of the LUT can be applied to any LUT protocol that builds on the random one-hot vector and dot-product approach. The security argument of the LUT-evaluation protocol is deferred to Appendix B. We introduce the shorthand notation for the bit length of one dimension as $l = \frac{k}{d}$.

4 Security and Privacy Considerations

In this section, we first discuss necessary precautions for concrete instantiations. On the one hand, we need to bound errors arising when computing exponentially small probabilities using finite precision arithmetic. On the other hand, the protocol π_{verify}, which realizes $\mathcal{F}_{verify}$, does so up to a failure probability proportional to the number of multiplications verified. As a consequence, we derive the maximum number of evaluations of π_Z that can be verified at once, such that the failure probability of π_{verify} remains negligible. Then, we provide the ideal functionality of our construction and state its security theorem. Finally, we conclude with a discussion of the concrete DP guarantees one gets when using our construction.

4.1 Computing the Approximation Error

The design of our algorithm $A_{\texttt{fill}}$ for constructing $\texttt{L}$ is fundamentally heuristic, and we give no general bound on the approximation error for arbitrary Z and I. Whereas such a general analytic bound would, perhaps, be preferred, we find that numerically computing the approximation error for a concrete instance $(Z, I, \texttt{L})$ can be done quite efficiently and with errors within said computation being inconsequentially small. In practice, an acceptably efficient and accurate instantiation of the sampling approach we present, with respect to a fixed Z, would be found by trial and error, with one selecting an index distribution and method for filling the LUT and then numerically bounding the approximation error. Whereas this might be time-consuming and give no optimality guarantees, it is quite unproblematic since it only needs to be done once for each target distribution Z.

For given $Z, I, \texttt{L}$, if we assume the support $z \in [-B, B]$ and the *truncation error* δ_{trunc}, given by $\delta_{trunc} = 2F_Z(-B-1)$, where F_Z is the cumulative density function of Z, then the approximation error δ_{appr} is

$$\begin{aligned}\delta_{appr} &:= SD\Big(Z, Out_{\pi_Z}\Big)\\ &= \frac{1}{2}\sum_{z\in[-B,B]}\Big|f_Z(z) - f_{Out_{\pi_Z}}(z)\Big| + \delta_{trunc}.\end{aligned}$$

Naively, one can evaluate the formula by representing the masses as floating-point numbers. However, this introduces errors due to rounding that are cumbersome to bound. The source of the rounding errors is that the masses can, in principle, be arbitrary real values whereas computers have finite precision. So we need to account for such errors arising in representing reals on a finite computer, and they may be present in both terms of the formula above. We denote our calculated value of the first term by δ^{calc}_{appr} and the upper bound on its error by δ^{err}_{appr}. Similarly, we have δ^{calc}_{trunc} and δ^{err}_{trunc}. This gives us the upper bound

$$\delta_{appr} \leq \delta^{calc}_{appr} + \delta^{err}_{appr} + \delta^{calc}_{trunc} + \delta^{err}_{trunc}.$$

To simplify the accounting for rounding errors, we use fixed-point numbers and we let n be the number of bits of precision.

Since Out_{π_Z} is a deterministic transformation of K fair coins (for some K), as long as $n \geq K$, each $f_{Out_{\pi_Z}}(z)$ can be represented without introducing any error. Due to our simple index distribution, this is always achieved in our instantiations.[3] Using such fixed-point representation for $f_Z(z)$ can, however, still introduce rounding error. We restrict the representations of all intermediate values to n bits of precision and track how error propagates through the computation of $f_Z(z)$. How errors propagate, of course, depends on Z, B and the concrete implementation of how $f_Z(z)$ is computed.

We have not yet discussed the computation of the truncation bound. We have two constraints; firstly, we need the cells of the LUT to be large enough to represent the values $[0, B]$, which restricts us to $B \leq 2^w$, and secondly we need B to be small enough that also the smallest mass $f_Z(B)$ can be computed without over- or underflow. To ensure this, we choose a bit precision n such that it is much larger than λ, which dictates the approximation error $2^{-\lambda}$ that we claim our protocol achieves and set B such that $\delta_{trunc} = 2F_Z(-B-1) \ll 2^{-\lambda}$. This necessitates an upper bound on F_Z and δ^{err}_{trunc}. To account for slack, we define $B^* := \min\{B : \delta^{calc}_{trunc} \leq 2^{-2\lambda}\}$ and set $B \leftarrow \min\{2^w - 1, B^*\}$. We defer details on how $\delta^{err}_{appr}, \delta^{err}_{trunc}$ are bounded specifically for the discrete Laplace and Gaussian distributions to Appendix C.1 and Appendix C.2, respectively. With $w = 8$, we have for the discrete Laplace

$$\delta_{appr} \leq \delta^{calc}_{appr} + \delta^{calc}_{trunc} + 2^{-n+269},$$

and for the discrete Gaussian

$$\delta_{appr} \leq \delta^{calc}_{appr} + \delta^{calc}_{trunc} + 2^{-n+15}.$$

To get a bound $\delta_{appr} \leq 2^{-\lambda}$, it is sufficient for each term on the right-hand side above to be at most $2^{-\lambda-2}$. We set our precision to $n = 512$ such that the computation error is negligible with respect to all considered values of λ in our evaluations (cf. Section 5.1).

4.2 Verification of Multiplication

We base the verification of our multiplication and dot product evaluations on Protocol 2 in MAESTRO, which securely realizes $\mathcal{F}_{\texttt{verify}}$ (Appendix A). The protocol takes m dot product relations $\{\mathbf{x}_i, \mathbf{y}_i, z_i\}_{i\in[m]}$, with each element being in a finite field $\mathbb{F}$, and verifies whether $z_i = \langle \mathbf{x}_i, \mathbf{y}_i \rangle$ for all i. However,

[3] In our implementation, z is determined uniquely by $K = c \cdot k + 1$ fair coins ($c \cdot k$ for the index and one for setting the sign) and we use $c \leq 12, k = 24, n = 512$.

with probability at most $(m+2\log N)/|\mathbb{F}|$, a cheating player is not caught, with $N = \sum_i |\mathbf{x}_i|$ being the summed up length of all vectors in the triples. First, note that the security relies on $|\mathbb{F}|$ being large and our protocols π_I and $\pi_{\texttt{RndOhv}}$ have multiplication triples in $\mathbb{F}_2$ and $\pi_{\texttt{LUT}}$ has dot-product triples in $\mathbb{F}_{2^\kappa}$. We lift our triples to the larger fields using respective isomorphisms $\psi : \mathbb{F}_{2^\kappa} \mapsto \mathbb{F}_{2^{64}}$ to retain the dot-product relations. After lifting all product and dot-product triples to $\mathbb{F}_{2^{64}}$, we can evaluate the total N and m of our construction and derive the maximum number of samples η that we can verify in one evaluation of the checking protocol, given a fixed upper bound on the failure probability.

For π_I and $\pi_{\texttt{RndOhv}}$, we record each multiplied bit individually, resulting in $(c-1)\cdot l'\eta$ and $(2^{k/d} - k/d - 1)\cdot d\eta$ recorded triples of length 1. When evaluating the LUT, we record $\eta \cdot \sum_{j=0}^{d-2} 2^{\frac{jk}{d}}$ dot-products of length $2^{k/d}$. With the upper bound $l' = k$ and the concrete parameters in Table 1, we get $m = (974 + 24c)\eta$ and $N = (66509 + 24c)\,\eta$. Thus, in our protocol, the failure probability will be dominated by m, which we bound by $2^{11}\eta$. Fixing a maximum failure probability of 2^{-40}, we arrive at an upper bound $\eta \leq 2^{13}$.

4.3 Security

We prove security using the SUC framework [6] in the setting of static corruptions, honest-majority and security-with-abort. For details about the framework, we refer to [6]. Important to note, however, is that it assumes communication is done over authenticated channels and the adversary (thus environment) controls the flow of messages, i.e. sees what messages are sent (not necessarily their contents) and can always choose whether a message should be delivered or not. This, in particular, applies to messages to and from ideal functionalities. That our protocol π_Z is secure is quite intuitive since it only consists of four consecutive calls to ideal functionalities (in particular, no intermediate shares are ever reconstructed). Therefore, the simulator in the proof only needs to emulate the message flow of the real world and handle aborts, which is straightforward. Involved ideal functionalities and subprotocols for realizing them are discussed in the appendix. The proof for the security theorem below is given in Appendix B.4.

Functionality $\mathcal{F}_{\mathsf{Z}}$

Parameters: Public LUT $\mathtt{L} \in \mathbb{Z}_{2^w}^{2^k}$, bias parameter $p_{Ber} = 2^{-c}$, number of biased coins $l' \leq k$.
Output: $[\![z]\!]$ with $z \leftarrow Out_{\pi_Z}$.

---

1 : $\mathcal{F}_{\mathsf{Z}}$ receives the corrupted parties' shares $[\![z]\!]_C$ from the adversary.

2 : $\mathcal{F}_{\mathsf{Z}}$ samples $\mathbf{r}^0, \ldots, \mathbf{r}^{c-1}$ uniformly from $\mathbb{F}_2^{l'}$, sets $i_j \leftarrow \prod_{j'=0}^{c-1} r_j^{j'}$ for $j \in [l']$, samples i_j uniformly from $\mathbb{F}_2$ for $j \in \{l', \ldots, k-1\}$ and evaluates $z' \leftarrow \mathtt{L}(i)$.

3 : $\mathcal{F}_{\mathsf{Z}}$ samples the bit b uniformly and sets $z \leftarrow (-1)^b \cdot z'$ and computes the honest parties' shares $[\![z]\!]_H$ of z to be consistent with $[\![z]\!]_C$ and sends them to the honest parties.

Fig. 5: The ideal functionality for our instantiation of the LUT-based sampling protocol.

Theorem 1. *The protocol π_Z (Figure 2) securely realizes $\mathcal{F}_{\mathtt{Z}}$ (Figure 5, with $k = 3l$) with abort in the ($\mathcal{F}_{\mathtt{Ber}}, \mathcal{F}_{\mathtt{LUT}}, \mathcal{F}_{\mathtt{weakMult}}$, $\mathcal{F}_{\mathtt{verify}}$)-hybrid model in the presence of an active adversary in the honest-majority setting.*

4.4 Privacy

Since the protocol is UC-secure, one can conveniently prove that a mechanism implementation where the query evaluation is computed UC-securely and then the noise from our protocol is added before the result is revealed is computationally DP (CDP). In particular, such a protocol would fulfill the recently proposed SIM*-CDP flavor of CDP [22], which in turn implies both SIM-CDP and IND-CDP [23] without a change in privacy parameters. When using our protocol to implement a DP mechanism, one needs to account for both the approximation error and potential failure probabilities of subprotocols in the implementation. Conveniently, both of these can be included into the δ-parameter of the DP guarantees, as is commonly done in DP mechanisms which use approximated noise distributions [28,20].

5 Implementation and Results

In general, the LUT-based sampling approach is highly parametrizable, and the following subsections aim to give an intuition on the versatility of the construction. In Section 5.1, we provide detailed insights into the building blocks, discussing different parameter regimes. In Section 5.2, we put it all together and compare our construction to various other MPC protocols for discrete Laplace and Gaussian sampling from recent literature [14,22,20]. All our benchmarks are run on a Linux machine with an AMD Ryzen 9 7900X CPU (4.7 GHz). For each party, we provide data for single-threaded evaluation.

In Table 1, we list concrete parameters choices that are fixed. Effectively, the sizes k and d influence the size of $\mathtt{L}$, while B directly influences the datatype stored in $\mathtt{L}$. We describe δ_{appr} by an upper bound $2^{-\lambda}$ and describe in Section 5.1 the λ's that we reach in different settings. As explained in Section 4.1, we set $B \leftarrow \min\{255, B^*\}$ with $B^* \leftarrow \min\{B : \delta^{calc}_{trunc} < 2^{-2\lambda}\}$, giving $w = 8$. For our implementation, we fix the computation domain and one-hot vector length such that $\kappa = l = w$. We fix $k = 24$ and given this and Equation (1) we minimize total communication with $d = 3$ such that $\mathtt{L} \in [0, B]^{2^{24}}$. For our fixed-point computation, we have a precision of $n = 512$, which we expand on in Section 4.1. Given these parameters, we fix the maximum batch size for π_{verify} to $\eta = 2^{13}$ following the discussion in Section 4.2. In case our construction is used in the malicious setting with more samples than that, π_{verify} has to be invoked multiple times.

Parameter	k	d	κ	l	w	n	η
Choice	24	3	8	8	8	512	2^{13}

Table 1: Our fixed parameter choices.

5.1 Performance of Subprotocols

Here, we discuss the concrete performance of all subprotocols as given in Table 2.

Subprotocol		Localhost		RTT = 1ms		RTT = 100ms		Total Communication		
		Single	Amort.	Single	Amort.	Single	Amort.	Rounds	Single	Amort
π_I		0.085	0.002	0.19	0.002	300	0.2	3	72	9
$\pi_{\texttt{LUT}}$	$\pi_{\texttt{RndOhv}}$	41.8	0.05	41.9	0.05	792	0.7	7	494	93
	$Eval$	42.6	1.82	45.4	1.83	243	2.18	3	260	260
π_{verify}		46.5	3.01	46.1	2.97	2050	5.82	$O(\log(N))$	448	0.664
π_Z	(SH)	84.7	1.91	86.9	1.92	1338	3.72	13	826	362
	(MAL)	133	4.92	133	4.89	3390	9.54	$O(\log(N))$	1274	363

Table 2: Cost of π_Z, with $d = 3$, $p_{Ber} = 1/16$, $k = 24$ in semi-honest (SH) and malicious (MAL) secure RSS. Communication is given in Bytes, Runtime in ms, and Amortized results with 1000 samples. Localhost runs without network restrictions. RTT 1 simulates a LAN setting with 1Gbit/s bandwidth and 0.5ms delay. RTT 100 simulates a WAN setup, giving 100Mbit/s bandwidth and 50ms delay.

Index Sampling. In our experiments testing approximation error limits, we have $k = 3l$ fixed in our index distribution and vary the two other parameters c and l'. We let c be between 2 and 12 and let l' be either $2l$ or $3l$. This can be interpreted as either all dimensions of the LUT being evaluated at indices drawn from $Ber(2^{-c})^l$ or only the first two, with the last dimension having a uniform index. We discuss the impact on network and runtime cost here and elaborate on the reason and effect in Section 5.1. The cost of the index sampling protocol π_I is dictated by the parameters c and l'. In particular, the cost analytically is $c-1$ multiplications per biased index bit (i.e., $(c-1)l'$ bits of communication) in $\lfloor \log_2 c \rfloor$ rounds. The amortized communication cost for $l' = 3l$ as given in Table 2 for $c = 4$ is given as 9 bytes (i.e., 3 multiplications per dimension in the LUT). The runtime of this subprotocol is around one percent of the semi-honest protocol for fast networks. In constrained settings, the round number incurs some delay for single-sample evaluation. Since the results quickly amortize to the exact analytic network cost above, we assume the cost of $(c-1)l'$ bits to hold also for the case of $l' = 2l$, without explicit benchmark.

Filling the LUT. Filling the LUT is a one-time cost independent of the number of samples generated. Thus, our benchmarks only report on the accuracy of the result. The concrete accuracy achievable through our filling strategy depends on the target distribution Z and the index distribution I. In Figure 6, we give accuracy results for the two settings of π_I, $l' = 2l$, and $l' = 3l$ as discussed in Section 5.1. For the target distribution, we approximate $DLap(p_{Lap})$ and $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ and report results for $p_{Lap} = \{e^{-0.5}, e^{-1}, e^{-5}\}$ and $\sigma^2 = \{\frac{1}{0.5^2}, \frac{1}{1^2}, \frac{1}{5^2}\}$. Following the discussion in Section 4.1, we have chosen the bit precision such that $\delta^{err}_{appr} + \delta^{err}_{trunc} \ll \delta^{calc}_{appr} + \delta^{calc}_{trunc}$. Hence, we claim $\delta_{appr} \leq 2^{-\lambda}$ with $\lambda := \max\{\lambda \in \mathbb{N} : 2(\delta^{calc}_{appr} + \delta^{calc}_{trunc}) \leq 2^{-\lambda-1}\}$. Looking first at the left side of Figure 6, we see a clear relation between increased bias in the index distribution and reduced approximation error, for both Laplace and Gauss. However, this relation holds until the largest probability mass in the index distribution is larger than the largest mass in the target. When that happens, assigning any sample value to this largest index leads to a significant approximation error, and we terminate the filling early. This effect can be reduced when lowering l', as seen in the right image of Figure 6, which is intuitive since this reduces the largest mass in the index distribution.

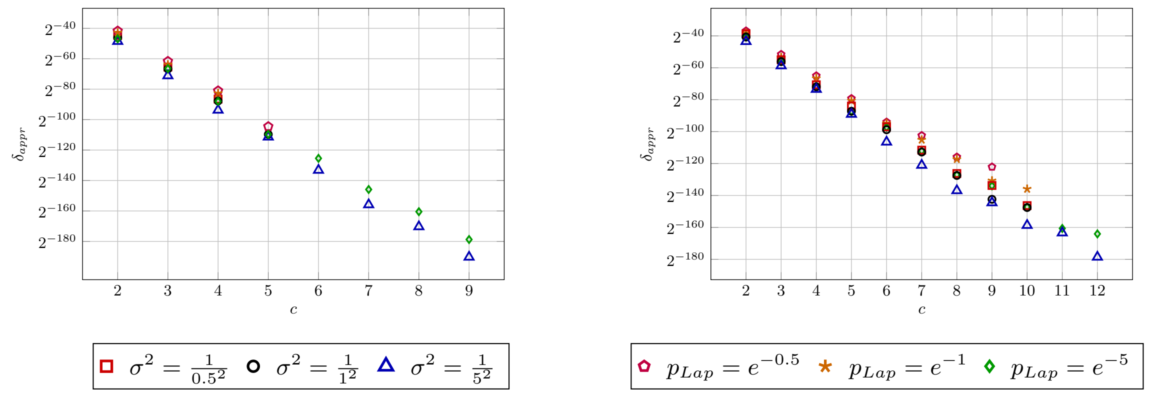

Fig. 6: The development of approximation errors under different index distribution parameters l'. The left has all index bits biased $l' = 3l$ while the right side has two dimensions of the LUT biased $l' = 2l$.

In this way, one can balance c and l' to reduce the approximation error for less concentrated targets. Finally, we stress that all of our results presented in this section are considering the LUT parameters as given in Table 1. Thus, besides the minimal performance impact of the different index sampling strategies, the efficiency of our protocol π_Z does not depend on the target distribution or approximation errors.

Lookup Table Evaluation. As we see in Table 2, the communication cost and runtime of $\pi_{\texttt{LUT}}$ dominate the overall metrics of our sampling protocol. When analytically evaluating the communication cost via Equation (1), we separate the two terms in the equation into the left one, i.e., the generation of one-hot vectors via $\pi_{\texttt{RndOhv}}$ and the right one, *Eval* (i.e., extracting a table element using these vectors). Given parameters ($k = 24, d = 3, w = 8$), we get 741 bits for $\pi_{\texttt{RndOhv}}$ and 2056 bits for *Eval*. Rounded to the next byte, we see that at 1000 samples, the implementation is fully amortized, communicating exactly 93 + 260 bytes. While 2056 bits represent 257 bytes, the remaining 24 bits are required to reconstruct the difference between the random and sampled indices. Considering the different network latencies, we see that $\pi_{\texttt{RandOhv}}$ is more heavily penalized as it has higher communication cost and round complexity. The cost of *Eval*, on the other hand, is dominated by the computational cost of collapsing the 2^{24} element cube. To further optimize the runtime of *Eval*, we provide a fully byte-sliced implementation that leverages the binary shares for a faster multiplication.

5.2 Comparison to Other Works

Before, we show that our construction can accurately sample from the Gaussian and Laplacian distributions for wide parameter regimes efficiently. Now, we provide more context on that efficiency by comparing it to related work. First in Table 3, we present benchmark numbers for recent protocols sampling discrete Laplace [20,22,14] and Gaussian [20,14] distributions. These numbers have to be taken with a grain of salt, as the security settings and number of parties differ between the

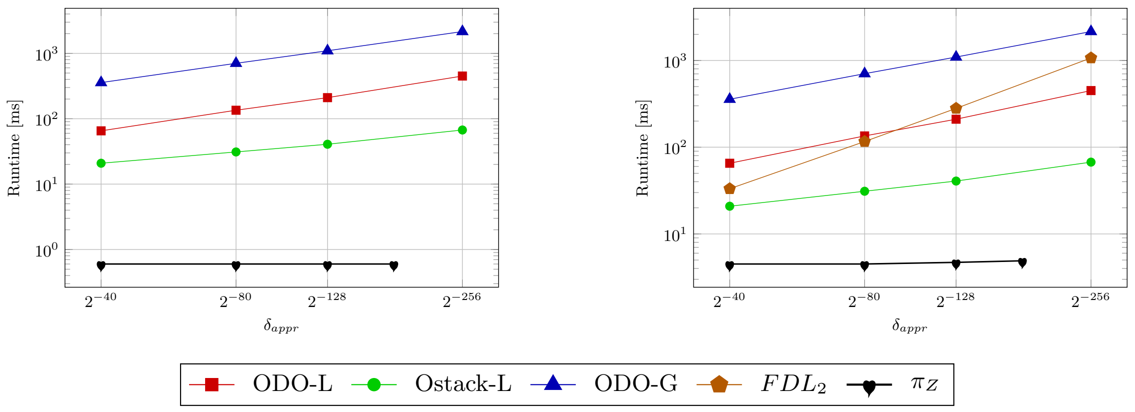

Fig. 7: Performance of different sampling strategies under different settings of λ, with $\sigma = 0.1$ for Gaussian and $p_{Lap} = e^{-3}$ for Laplacian samples. The security settings are semi-honest on the left and malicious security on the right side.

Z	Protocol	Security-Setting	Localhost Single	Localhost Amort.	RTT = 1ms Single	RTT = 1ms Amort.	RTT = 100ms Single	RTT = 100ms Amort.	Total Communication Single	Total Communication Amort
			2-Party Setting							
$DLap$	M_{DLap} [20]	◑	1643	992	-	4711	-	42 352	492 720	-
	FDL_2 [22]	●	214.1	118.5	895.7	209.7	67 009	1604	158 300	75 300
$\mathcal{N}_{\mathbb{Z}}$	M_{Gauss} [20]	◑	6254	5615	-	11 098	-	66 878	1 085 820	-
			3-Party Setting							
$DLap$	ODO-Laplace [14]	○	136.4	133.9	193.8	198.7	240 286	242 934	1062.4	1072.9
		◐	137	128.1	208.4	215	282 072	285 322	11 584	3875.2
	Ostack-Laplace [14]	○	290.8	30.4	334.5	38.8	239 909	67 785	29 873	2400.6
		◐	756.5	68.3	878.8	82.6	368 527	96 397	113 501	9106.1
$\mathcal{N}_{\mathbb{Z}}$	ODO-Gauss [14]	○	6083	704	8858	1031	10 292 540	-	16 115	5823
		◐	6200	705	8906	1031	12 081 050	-	50 869	5823
Either π_Z		○	**85**	**1.9**	**87**	**1.9**	**1338**	**3.7**	**1**	**0.4**
		◐	**133**	**4.9**	**133**	**4.9**	**3390**	**9.5**	**1.3**	**0.4**

Amortization is over 1000 samples, except for higher latencies, where M_{DLap} is amortized over 40 and 10 samples, and ODO and Ostack are amortized over 32 samples for the highest latency.

Table 3: Comparison to other MPC noise sampling protocols from the literature. Runtime is given in ms, and communication in KB. All protocols are instantiated with $\lambda = 80$. The network settings are as in Table 2. Security settings are given by ○ : Semi-Honest + Honest-Majority, ◐ : Malicious Security + Honest-Majority, ◑ : Semi-Honest + Dishonest-Majority, ● : Malicious Security + Dishonest-Majority.

benchmarked protocols (cf. Table 3). While Fu and Wang [14] and Keller et al. [20] support different numbers of parties and Meisingseth et al. [22] could be extended to more parties, we only provide data for the best performance for the sake of brevity. Additionally, Keller et al. and Fu and Wang present performance numbers for numerous strategies, however, we were not able to reproduce all of the results. For Keller et al., we had to resort to the numbers presented and select the best representatives. For Fu and Wang we were able to evaluate the best performing candidates.

With these caveats in mind, we see the advantage of our construction lies in its low networking cost. In the presented setting, π_Z outperforms related work in all network settings in both single-shot and amortized performance. While there could be some gains when implementing FDL_2 in a more relaxed MPC setting, the comparison with the discrete Gauss target is clear, with at least 46x improvement. While our improvement factor grows with increasing network latency, we see in Table 2 that even though we have a low communication cost, the malicious secure variant of our construction is more affected. This is due to the round complexity of π_{verify}, which scales logarithmically with the number of samples. Every round of communication is then penalized by the introduced network latency.

As discussed above, our comparison table fixes the security parameter to $\lambda = 80$. To give a more nuanced view, we show the performance in relation to the security parameter in Figure 7, highlighting both a strength and a weakness of our construction. Without changing the parameters k or d, we can adapt to different values of λ by changing the bias parameter c of our index distribution. The strength of this lies in the minimal performance impact of c and thus we get a relatively constant performance for different λ. Figure 7 further shows that the cost for verification is dominated by the one-hot vector, since the impact of c on malicious runtime is marginal. The fundamental downside of our construction lies in the limited approximation accuracy. For cases where $\lambda > 180$ is required, we did not find sufficiently accurate approximations with our index distribution and table size.

At the time of writing, the implementation of Franzese et al. [13] was not yet public. They present promising benchmark numbers with a runtime similar to our construction but significantly higher communication costs. While their construction is secure again dishonest majorities and scales well with an increasing number of parties, due to using homomorphic encryption, translating their protocol from semi-honest to malicious security appears challenging.

Comparative Conclusions. The results presented show the concrete limitations of this instantiation of our construction. We fix the number of players to 3 and show that a specific set of lookup table parameters has a limit on how well it can approximate specific target distributions. This restriction, however, still allows sampling discrete Laplace and Gaussian distributions with a wider range of parameters and achieve security parameters up to $\lambda = 180$. For such distributions, our protocol constitutes a substantial efficiency improvement compared to the state of the art. All previous constructions and the concurrent work have communication complexity in the megabyte range, while our construction has an amortized cost of under a kilobyte. Further, the efficiency of the protocol is independent of the target noise distribution, as long as the distribution can be approximated sufficiently well under the fixed choice of LUT size and index distribution.

Finally, our construction builds on protocols for random one-hot vectors and cheap dot-products, which are not exclusive to the 3-party setting. Thus, with some engineering work, this construction can be adapted to different MPC settings as well.

Acknowledgments. This work is partly supported by the European Union under the project Confidential6G with Grant agreement ID: 101096435. We thank Marcel Nageler for the fruitful discussions about different lookup table filling strategies. We thank anonymous reviewers for feedback which guided us to extend and clarify our contributions.

References

1. Beimel, A., Nissim, K., Omri, E.: Distributed private data analysis: Simultaneously solving how and what. In: CRYPTO 2008 (2008)
2. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear pcps. In: CRYPTO (2019). https://doi.org/10.1007/978-3-030-26954-8_3
3. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS. ACM (2016)
4. Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., Yalame, H.: FLUTE: fast and secure lookup table evaluations. In: S&P (2023)
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Paper 2000/067 (2000), https://eprint.iacr.org/2000/067
6. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: CRYPTO 2015 (2015)
7. Canonne, C., Kamath, G., Steinke, T.: The discrete gaussian for differential privacy. Journal of Privacy and Confidentiality (2022). https://doi.org/10.29012/jpc.784
8. Champion, J., shelat, a., Ullman, J.: Securely sampling biased coins with applications to differential privacy. In: CCS '19. New York, NY, USA (2019). https://doi.org/10.1145/3319535.3354256
9. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: TCC (2005)
10. Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: NDSS (2017)
11. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: Privacy via distributed noise generation. In: EUROCRYPT'06 (2006). https://doi.org/10.1007/11761679_29
12. Eriguchi, R., Ichikawa, A., Kunihiro, N., Nuida, K.: Efficient noise generation protocols for differentially private multiparty computation. IEEE Transactions on Dependable and Secure Computing (2023). https://doi.org/10.1109/TDSC.2022.3227568
13. Franzese, O., Fang, C., Garg, R., Jha, S., Papernot, N., Wang, X., Dziedzic, A.: Secure noise sampling for differentially private collaborative learning. Cryptology ePrint Archive, Paper 2025/1025 (2025), https://eprint.iacr.org/2025/1025
14. Fu, Y., Wang, T.: Benchmarking secure sampling protocols for differential privacy. In: CCS '24 (2024). https://doi.org/10.1145/3658644.3690257
15. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. SIAM Journal on Computing (2012). https://doi.org/10.1137/09076828X
16. Heath, D., Kolesnikov, V., Ng, L.K.L.: Garbled circuit lookup tables with logarithmic number of ciphertexts. In: EUROCRYPT (2024)
17. Inusah, S., Kozubowski, T.J.: A discrete analogue of the laplace distribution. Journal of Statistical Planning and Inference (2006). https://doi.org/https://doi.org/10.1016/j.jspi.2004.08.014
18. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC (2013)
19. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (1989). https://doi.org/10.1002/ecjc.4430720906
20. Keller, H., Möllering, H., Schneider, T., Tkachenko, O., Zhao, L.: Secure noise sampling for dp in mpc with finite precision. In: ARES '24 (2024). https://doi.org/10.1145/3664476.3664490
21. Keller, M., Orsini, E., Rotaru, D., Scholl, P., Soria-Vazquez, E., Vivek, S.: Faster secure multi-party computation of aes and des using lookup tables. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) Applied Cryptography and Network Security. pp. 229–249. Springer International Publishing, Cham (2017)
22. Meisingseth, F., Rechberger, C., Schmid, F.: Practical two-party computational differential privacy with active security. PoPETs (2025). https://doi.org/10.56553/POPETS-2025-0019
23. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: CRYPTO 2009 (2009)

24. Mohassel, P., Rindal, P.: Aby^3: A mixed protocol framework for machine learning. In: CCS. pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760
25. Morita, H., Pohle, E., Sadakane, K., Scholl, P., Tozawa, K., Tschudi, D.: MAESTRO: multi-party AES using lookup tables. IACR Cryptol. ePrint Arch. p. 1317 (2024), https://eprint.iacr.org/2024/1317
26. Shamir, A.: How to share a secret. Commun. ACM (1979). https://doi.org/10.1145/359168.359176
27. Wagh, S.: Pika: Secure computation using function secret sharing over rings. PoPETs **2022**(4), 351–377 (2022)
28. Wei, C., Yu, R., Fan, Y., Chen, W., Wang, T.: Securely sampling discrete gaussian noise for multi-party differential privacy. In: CCS '23 (2023). https://doi.org/10.1145/3576915.3616641

A Ideal functionalities

In this section, we discuss the ideal functionalities we use in the main body, except for the functionality for Bernoulli trials, which is treated in the next section. The functionalities are $\mathcal{F}_{\texttt{rand}}$, $\mathcal{F}_{\texttt{weakDotProduct}}$/ $\mathcal{F}_{\texttt{weakMult}}$, $\mathcal{F}_{\texttt{verify}}$, $\mathcal{F}_{\texttt{RandOHV}}$, $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ and for all of them we use their formulations as in MAESTRO [25] so we refer the reader there for more detailed discussions. Informally, the functionalities may be understood as follows:

– $\mathcal{F}_{\texttt{rand}}$ (Figure 8) samples a uniformly random value in a finite field and distributes shares of this value to all parties.
– $\mathcal{F}_{\texttt{weakDotProduct}}/\mathcal{F}_{\texttt{weakMult}}$ (Figure 9) gets shares from all parties for vectors $\mathbf{x}, \mathbf{y}$ of field elements, performs their dot product, and lets the adversary introduce an arbitrary additive error. The result is then shared between all parties. When the vectors are of length 1, we refer to the ideal functionality as $\mathcal{F}_{\texttt{weakMult}}$.
– $\mathcal{F}_{\texttt{verify}}$ (Figure 10) gets shares for a number of triples $\mathbf{x}, \mathbf{y}, z$ where $\mathbf{x}, \mathbf{y}$ are vectors of field elements and z is a single field element. The functionality reconstructs the triples and checks if $\langle \mathbf{x}, \mathbf{y} \rangle = z$ and if this is the case for all triples then it tells the parties to accept the triples. Otherwise, the functionality tells the parties to abort.
– $\mathcal{F}_{\texttt{RandOHV}}$ (Figure 11) samples a random field element r and a one-hot vector $e^{(r)}$ of length N which is 0 in all places except for position r, which instead has the value 1. The functionality shares both r and $e^{(r)}$ to the parties.
– $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ (Figure 12) holds a LUT $\texttt{L}$ and gets shares of an index i. The functionality reconstructs i, evaluates $\texttt{L}$ at position i, and shares the result with the parties.

Functionality $\mathcal{F}_{\texttt{rand}}$

Parameter: Field $\mathbb{F}$.
Input: None.
Output: $[\![r]\!]$ with r uniform in $\mathbb{F}$.

---

1 : $\mathcal{F}_{\texttt{rand}}$ receives the corrupted parties' shares $[\![r]\!]_C$ from the adversary.

2 : $\mathcal{F}_{\texttt{rand}}$ samples r uniformly from $\mathbb{F}$ and computes the honest parties shares $[\![r]\!]_H$ of r to be consistent with $[\![r]\!]_C$ and sends them to the honest parties.

Fig. 8: The ideal functionality for generating random elements.

Functionality $\mathcal{F}_{\texttt{weakDotProduct}}/\mathcal{F}_{\texttt{weakMult}}$

Input: $[\![\mathbf{x}]\!], [\![\mathbf{y}]\!]$
Output: $[\![\mathbf{x} \cdot \mathbf{y} + d]\!]$, with d being an error decided by the adversary.
We refer to this functionality as $\mathcal{F}_{\texttt{weakMult}}$ when $|\mathbf{x}| = |\mathbf{y}| = 1$.

---

1 : $\mathcal{F}_{\texttt{weakDotProduct}}$ receives $[\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j$ from each honest party P_j and reconstructs the secrets $\mathbf{x}, \mathbf{y}$.

2 : $\mathcal{F}_{\texttt{weakDotProduct}}$ computes the shares $[\![\mathbf{x}]\!]_j, [\![\mathbf{y}]\!]_j$ for each corrupted party P_j and sends them all to the adversary.

3 : Upon receiving an error d and a set of shares $[\![z]\!]_C$ from the adversary, $\mathcal{F}_{\texttt{weakDotProduct}}$ computes $z \leftarrow x \cdot y + d$, samples $[\![z]\!]_j$ for each honest party P_j to be consistent with $[\![z]\!]_C$ and sends them to the respective honest parties.

Fig. 9: The ideal functionality for weak dot products/multiplications.

B Sub-protocols and their Security Arguments

For realizing $\mathcal{F}_{\texttt{rand}}, \mathcal{F}_{\texttt{weakDotProduct}}/\mathcal{F}_{\texttt{weakMult}}, \mathcal{F}_{\texttt{verify}}, \mathcal{F}_{\texttt{RandOHV}}$ and $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$, we use the same subprotocols (some with small tweaks inconsequential to security) as in MAESTRO [25] so we refer the reader there for details. The security of the protocols below rely solely on two elementary facts; Firstly, the adversary never gets hold of sufficiently many shares of intermediate values to reconstruct, and thus the shares it does hold are uniformly random regardless of the underlying value and the adversary cannot check consistency with the corresponding shares held by the honest parties. Secondly, basic arithmetic operations on shares, i.e. local addition and bit decomposition and multiplication via weak multiplication followed by verification, always are correct and introduce no opportunity for cheating except for aborting, of course.

Functionality $\mathcal{F}_{\texttt{verify}}$

Input: $\{[\![\mathbf{x}]\!]^i, [\![\mathbf{y}]\!]^i, [\![\mathbf{z}]\!]^i\}_{i\in[m]}$.
Output: $b \in \{\texttt{accept}, \texttt{abort}\}$

---

1 : $\mathcal{F}_{\texttt{verify}}$ receives the honest parties' shares of $\{[\![\mathbf{x}]\!]^i, [\![\mathbf{y}]\!]^i, [\![\mathbf{z}]\!]^i\}_{i\in[m]}$ and reconstructs the secrets $\{\mathbf{x}^i, \mathbf{y}^i, \mathbf{z}^i\}_{i\in[m]}$.

2 : $\mathcal{F}_{\texttt{verify}}$ computes the corrupted parties' shares of $\{\mathbf{x}^i, \mathbf{y}^i, \mathbf{z}^i\}_{i\in[m]}$ and sends these to the adversary.

3 : $\mathcal{F}_{\texttt{verify}}$ sets $b \leftarrow \texttt{abort}$ if there exists an $i \in [m]$ such that $z^i \neq \mathbf{x}^i \cdot \mathbf{y}^i$, else it sets $b \leftarrow \texttt{accept}$. $\mathcal{F}_{\texttt{verify}}$ sends b to the adversary.

4 : If the adversary replies `continue`, $\mathcal{F}_{\texttt{verify}}$ sends b to the honest parties. If the adversary instead replies `abort`, $\mathcal{F}_{\texttt{verify}}$ sends `abort` to the honest parties.

Fig. 10: The ideal functionality for verifying dot product triples.

Functionality $\mathcal{F}_{\texttt{RandOHV}}$

Parameter: $N = 2^k$.
Input: None.
Output: Shares of k random bits $\{[\![r_i]\!]\}_{i\in[k]}$ and one shared N-length one-hot vector $[\![e^{(r)}]\!]$ which is 1 at element $r \leftarrow \sum_{i=0}^{k-1} 2^i r_i$.

---

1 : $\mathcal{F}_{\texttt{RandOHV}}$ receives the corrupted parties' shares of $\{r_i\}_{i\in[k]}$ and $e^{(r)}$ from the adversary.

2 : $\mathcal{F}_{\texttt{RandOHV}}$ samples $\{r_i\}_{i\in[k]}$ uniformly and computes $r \leftarrow \sum_{i=0}^{k-1} 2^i r_i$ and $e^{(r)}$.

3 : $\mathcal{F}_{\texttt{RandOHV}}$ computes the honest parties' shares of $\{r_i\}_{i\in[k]}$ and $e^{(r)}$ to be consistent with the corrupted parties' shares. $\mathcal{F}_{\texttt{RandOHV}}$ sends the honest parties' shares to the honest parties.

Fig. 11: The ideal functionality for generating random one-hot vectors.

B.1 Bernoulli Index Sampling

The protocol for Bernoulli index sampling is found in Figure 13. That the protocol is correct (outputs a shared index i in $\mathbb{Z}_{2^k}$ where each bit of i has $Ber(2^{-c})$ or $Ber(1/2)$ distribution, respectively) is immediate since the multiplication of c uniform bits has $Ber(2^{-c})$ distribution

Functionality $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$

Parameter: Public LUT $\texttt{L}$.
Input: Index share $[\![i]\!]$.
Output: $[\![\texttt{L}(i)]\!]$, the share of $\texttt{L}$ evaluated at index i.

```
1 : $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ receives the honest parties' shares of $i$ and reconstructs $i$. $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ computes the corrupted parties' shares of $i$ and sends them to the adversary.
2 : $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ evaluates $\texttt{L}(i)$. $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ Receives the corrupted parties' shares of $\texttt{L}(i)$ from the adversary and samples the honest parties' shares to be consistent with those for the corrupted parties.
3 : $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ sends the honest parties' shares to them.
```

Fig. 12: The ideal functionality for obliviously evaluating a lookup table.

Protocol π_I, with $i \sim Ber(2^{-c})^{l'} \times Ber(1/2)^{k-l'}$

Parameters: Bias parameter $c \in \mathbb{N}$, total number of samples $k \in \mathbb{N}$, number of biased samples $l' \leq k$.
Input: None.
Output: Shared index $[\![i]\!]$, with $i \in \mathbb{Z}_{2^k}$.
Each party acts as below. Multiplication of multiple elements is done via repeated calls to $\mathcal{F}_{\texttt{weakMult}}$.

```
1 : Sample randomness: $([\![r^{j'}_{l'-1}]\!], \dots, [\![r^{j'}_0]\!]) \leftarrow \mathcal{F}_{\texttt{rand}}$ for $j' = [c]$.
2 : Set the index bits: $[\![i_j]\!] \leftarrow \prod_{j'=0}^{c-1} [\![r^{j'}_j]\!]$ for $j \in [l']$ and $i_j \leftarrow \mathcal{F}_{\texttt{rand}}$ for $j \in \{l', \dots k-1\}$.
3 : Verify all the multiplication triples via $\mathcal{F}_{\texttt{verify}}$.
4 : return $[\![i]\!]$
```

Fig. 13: The index sampling protocol π_I, where the bits of the index are either i.i.d. according to $Ber(2^{-c})$ or $Ber(1/2)$.

and $\mathcal{F}_{\texttt{rand}}$ returns independent fair coins. The security of the protocol (also with respect to active corruptions) follows directly since it only consists of operations on secret shares (that are never opened), and in particular, all operations we do are weak multiplications that are then verified.

Lemma 1. *Protocol π_I (Figure 13) securely realizes functionality $\mathcal{F}_{\texttt{Ber}}$ (Figure 4) in the ($\mathcal{F}_{\texttt{weakDotProduct}}$, $\mathcal{F}_{\texttt{verify}}, \mathcal{F}_{\texttt{RandOHV}}$)-hybrid model in the presence of an active adversary in the honest-majority setting.*

B.2 One-hot Vector Generation

Protocol$\pi_{\texttt{RndOhv}}$

Parameter: $N = 2^k$.
Input: None.
Output: Index $[\![r]\!]$, OHV $[\![e^{(r)}]\!]$
Each party acts as below. All multiplications between shares are done via $\mathcal{F}_{\texttt{weakDotProduct}}$.

```
1 :  $([\![r_{k-1}]\!], \ldots, [\![r_0]\!]) \leftarrow \mathcal{F}_{\texttt{rand}}$
2 :  $[\![\mathbf{v}]\!] \leftarrow (1 - [\![r_0]\!], [\![r_0]\!])$
3 :  for $j = 1, \ldots, k-1$ :
4 :    $M \leftarrow 2^j$
5 :    for $j' = 0, \ldots, M/2 - 2$ :
6 :      $[\![e_{j'}]\!] \leftarrow \mathcal{F}_{\texttt{weakMult}}([\![r_j]\!], [\![v_{j'}]\!])$
7 :    endfor
8 :    $[\![e_{M/2-1}]\!] \leftarrow \left(\bigoplus_{j'=0}^{M/2-2} [\![e_{j'}]\!]\right) \oplus [\![r_{j'}]\!]$
9 :    $[\![\mathbf{v}]\!] \leftarrow ([\![v_0]\!] \oplus [\![e_0]\!], \ldots, [\![v_{\frac{M}{2}-1}]\!] \oplus [\![e_{\frac{M}{2}-1}]\!], [\![e_0]\!], \ldots, [\![e_{\frac{M}{2}-1}]\!])$
10 : endfor
11 : $[\![r]\!] \leftarrow \sum_{j=0}^{k-1} 2^j [\![r_j]\!]$
12 : Run $\mathcal{F}_{\texttt{verify}}$ on all multiplication triples used above.
13 : return $([\![r]\!], [\![\mathbf{v}]\!])$
```

Fig. 14: The protocol $\pi_{\texttt{RndOhv}}$ (from [25]) for sampling random one-hot vectors.

The protocol for sampling random one-hot vectors is given in Figure 14. It is precisely the one from MAESTRO [25] with the purely cosmetic change that we phrase it iteratively rather than recursively. The protocol specification in MAESTRO is itself essentially a reformulation of the corresponding protocol in [21], where the correctness of the protocol and UC-security in the *Arithmetic Black-Box (ABB)* model is proven. The protocol formulation we use is instead in the $(\mathcal{F}_{\texttt{weakDotProduct}}, \mathcal{F}_{\texttt{verify}}, \mathcal{F}_{\texttt{rand}})$-hybrid model. Therefore, the UC-security of the protocol formulation above follows via the proof in the ABB model, since $\mathcal{F}_{\texttt{weakDotProduct}}$ and $\mathcal{F}_{\texttt{verify}}$ together guarantee faithful multiplications and linear operations trivially are secure in RSS, in the honest majority setting.

Lemma 2 **([25,21]).** *Protocol $\pi_{\texttt{RndOhv}}$ (Figure 14) securely realizes functionality $\mathcal{F}_{\texttt{RandOHV}}$ in the ($\mathcal{F}_{\texttt{weakDotProduct}}, \mathcal{F}_{\texttt{verify}}, \mathcal{F}_{\texttt{rand}}$)-hybrid model in the presence of an active adversary in the honest majority setting.*

B.3 Lookup Table Evaluation

Protocol $\pi_{\texttt{LUT}}$ with $d = 3$

Parameters: LUT size $N = 2^k = 2^{3l}$, LUT $\mathtt{L} \in \mathbb{Z}_{2^w}^N$
Input: Shared index $[\![i]\!]$ with $i \in \mathbb{Z}_N$.
Output: $[\![s := \mathtt{L}(i)]\!]$.

```
1 :  Sample 3 random one-hot vectors:
         $(\{[\![\mathbf{r}]\!]\}, e^{(r)}), ([\![\mathbf{s}]\!]\}, e^{(s)}), ([\![]\!], \mathtt{e}^{(\mathtt{t})}),$
         each from $\mathcal{F}_{\texttt{RandOHV}}(2^l)$.
2 :  $[\![\hat{\mathbf{r}}]\!] \leftarrow ([\![r_0]\!], \ldots, [\![r_{2^l-1}]\!], [\![s_0]\!], \ldots, [\![s_{2^l-1}]\!],$
     $[\![t_0]\!], \ldots, [\![t_{2^l-1}]\!])$
3 :  $c \leftarrow \mathit{Reconstruct}([\![i]\!] + [\![\hat{\mathbf{r}}]\!])$.
4 :  $[\![\mathbf{v}]\!] \leftarrow (\underbrace{[\![e_0^{(r)}]\!], \ldots, [\![e_0^{(r)}]\!]}_{\times 2^{2l}}, \ldots, [\![e_{2^l-1}^{(r)}]\!], \ldots, [\![e_{2^l-1}^{(r)}]\!])$
         with $\mathbf{v}$ being of length $N$.
5 :  $[\![\mathbf{v}']\!] \leftarrow (\underbrace{[\![e_0^{(s)}]\!], \ldots, [\![e_0^{(s)}]\!]}_{\times 2^{l}}, \ldots, [\![e_{2^l-1}^{(s)}]\!], \ldots, [\![e_{2^l-1}^{(s)}]\!])$
6 :  $[\![\mathbf{v}'']\!] \leftarrow ([\![e_0^{(t)}]\!], \ldots, [\![e_{2^l-1}^{(t)}]\!], \ldots, [\![e_0^{(t)}]\!], \ldots, [\![e_{2^l-1}^{(t)}]\!])$
7 :  $[\![\mathbf{w}]\!] \leftarrow (\mathtt{L}_c \cdot [\![v_0]\!], \ldots, \mathtt{L}_{c \oplus N-1} \cdot [\![v_{N-1}]\!])$
8 :  $[\![w'_j]\!] \leftarrow \mathcal{F}_{\texttt{weakMult}}([\![w_j]\!], [\![v'_j]\!])$ for $j \in [N]$.
9 :  $[\![\mathbf{w}']\!] \leftarrow ([\![w'_0]\!], \ldots, [\![w'_{N-1}]\!])$.
10 : $[\![w]\!] \leftarrow \mathcal{F}_{\texttt{weakDotProduct}}([\![\mathbf{w}']\!], [\![\mathbf{v}'']\!])$
11 : Run $\mathcal{F}_{\texttt{verify}}$ on all dot products and multiplications above and **return** $[\![w]\!]$.
```

Fig. 15: MPC lookup protocol with $d = 3$. The protocol is an extension of Protocol 10 in MAESTRO[25], the only difference is that we have $d = 3$ and they have $d = 2$.

The protocol for oblivious lookup table evaluations is given in Figure 15. It is the same as the protocol in MAESTRO [25] for two-dimensional LUT evaluations (their Protocol 10), except that we introduce one extra dimension. The security argument, however, as is easy to see, remains unchanged since the only extra operations we incur is a set of weak multiplications that are later verified. In particular, the security of our protocol follows immediately from that we simply do a number of basic arithmetic operations on RSS shares and public values.

Lemma 3. *Protocol $\pi_{\texttt{LUT}}$ (Figure 15) securely realizes the functionality $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ in the ($\mathcal{F}_{\texttt{weakDotProduct}}$, $\mathcal{F}_{\texttt{verify}}, \mathcal{F}_{\texttt{RandOHV}}$)-hybrid model in the presence of an active adversary in the honest-majority setting.*

B.4 Security of Full Sampling Protocol

We now give the security proof of our sampling protocol.

Proof (Theorem 1). The theorem follows essentially directly from that the protocol consists of nothing except for four calls to ideal functionalities, so the simulator only needs to emulate the flow of messages and handle aborts. In more detail, the view of the environment in the real world consists of the corrupted parties' shares of i, z', z, b and intermediate random values, the adversary's internal randomness, and the final outputs of the honest parties. In the ideal world, the simulator $\mathcal{S}$ can be constructed as below. If at any point the simulator $\mathcal{S}$ receives an `abort` message from the environment then $\mathcal{S}$ tells all honest parties to abort. Similarly, if the environment instructs $\mathcal{S}$ to not allow the delivery of some message, then $\mathcal{S}$ obeys.

1. $\mathcal{S}$ samples the corrupted parties' shares, $[\![z]\!]_C$, of z.
2. $\mathcal{S}$ emulates $\mathcal{F}_{\texttt{Ber}}(p_{Ber}, 3l, l')$ and $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ by sampling uniform fake shares of the index i for the corrupted parties. $\mathcal{S}$ then samples the corrupted parties' shares of z', $[\![z']\!]_C$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\texttt{rand}}$ by sampling the corrupted parties' shares of b.
4. $\mathcal{S}$ emulates $\mathcal{F}_{\texttt{weakMult}}$ by sampling uniform fake shares of the multiplied values and then computes the error d to be added to the multiplication result.
5. $\mathcal{S}$ emulates $\mathcal{F}_{\texttt{verify}}$ by sampling uniform fake shares of the multiplication triple and computing the bit indicating if the multiplication triples in question were correct (the bit is set to `accept` if $d = 0$ and to `abort` otherwise).
6. If $\mathcal{S}$ is instructed to answer `continue` then S calls $\mathcal{F}_{\texttt{Z}}(\texttt{L}, p_{Ber}, l')$ by sending it $[\![z]\!]_C$.

Note that throughout, the environment may request to get the (fake) shares of i, z', z, b and instruct the simulator to change the messages it sends (in particular, the error d and the accept-bit in the verification), in which case the simulator (just as the real-world adversary) obeys. The views of the environment in the ideal and real worlds are identically distributed. In particular, in both worlds, during the execution the environment gets at most:

- The corrupted parties' shares of the index.
- The corrupted parties' shares of z, z' and b.
- The message saying if the multiplication was correct or not.
- The outputs of the honest parties.

In the real world, all the shares are indeed shares of the supposed values and in the ideal world, each of them are uniformly random. Since the environment in each case never gets hold of the honest parties' shares (as corruptions are static), every share is uniformly random in both worlds. The verification bit, similarly, is in both worlds set to `accept` if and only if $d = 0$, i.e. the adversary is not instructed to introduce an error in the multiplication. Further, the outputs of the honest parties in both worlds are precisely consistent shares of $b \cdot z'$ with b being fair and z' drawn from $\mathtt{L}(I)$, unless the environment at some point aborts. Therefore, unless the environment request an abort, the views of it in the two worlds are identically distributed. Finally, in both worlds, the adversary can cause an abort after each of the messages it receives, causing the honest parties to immediately abort. Therefore, the views of the environment in the two worlds are identically distributed also in the case of aborts.

□

C Details on Upper Bounding δ_{appr}

C.1 Bounding $\delta^{err}_{appr}, \delta^{err}_{trunc}$ for Laplace

For the discrete Laplace distribution with parameter $p \in (0,1)$ and truncation to $[-B, B]$, the error propagation of our computation of the statistical distance works as follows. The formula for the mass function is $f_Z(z) := \frac{1-p}{1+p} p^{|z|}$, and we start by scaling p up and round it to the closest n-bit integer. Letting $\Delta := 2^n$, we have $\tilde{p} := \lceil \Delta \cdot p \rfloor$ and $|p - \Delta^{-1}\tilde{p}| \leq 2^{-n}$. Assume that $\Delta \cdot p \geq \tilde{p}$ and thus $\Delta \cdot p = \tilde{p} + e_p$, with e_p denoting the rounding error, which is no larger than 1. This simplifies notation and is WLOG because the other case analogously gives the same bound on the absolute error. We have $p^{|z|} = (\Delta^{-1}(\tilde{p} + e_p))^{|z|} = \Delta^{-|z|}(\tilde{p} + e_p)^{|z|}$ and we approximate $p^{|z|}$ by $(\Delta^{-1} \cdot \tilde{p})^{|z|}$, which we compute by repeated multiplication and rounding. This gives the error

$$
\begin{aligned}
&|p^{|z|} - (\Delta^{-1}\tilde{p})^{|z|}| = \Delta^{-|z|} |(\tilde{p} + e)^{|z|} - \tilde{p}^{|z|}| \\
&= \Delta^{-|z|} \sum_{i=0}^{|z|-1} \binom{|z|}{i} \tilde{p}^i e^{|z|-i} \\
&\leq \Delta^{-|z|} \sum_{i=0}^{|z|-1} \binom{|z|}{\lceil |z|/2 \rfloor} \tilde{p}^i e^{|z|-i} \\
&\leq \Delta^{-|z|} \binom{|z|}{\lceil |z|/2 \rfloor} \sum_{i=0}^{|z|-1} \tilde{p}^{|z|-1} \\
&= \Delta^{-|z|} \binom{|z|}{\lceil |z|/2 \rfloor} (|z|-1) \tilde{p}^{|z|-1} \\
&\leq \Delta^{-|z|} \binom{|z|}{\lceil |z|/2 \rfloor} (|z|-1) \Delta^{(|z|-1)} \\
&= \binom{B}{\lceil B/2 \rfloor} (B-1) 2^{-n}.
\end{aligned}
$$

It is now clear that we can set $n \propto \lambda$ and B constant which makes the error exponentially small. Since the fraction in the formula for $f_Z(z)$ is smaller than 1, the error for f_Z is upper bound by the error caused by the exponentiation. Therefore we arrive at $\delta^{err}_{appr} \leq (2B+1)\binom{B}{\lceil B/2 \rfloor}(B-1)2^{-n}$. In our instantiation, we have $B \leq 255$ which gives $\binom{B}{\lceil B/2 \rfloor} \leq \binom{255}{127} < 2^{251}$, $(2B+1) \leq 511 < 2^9$, $B-1 \leq 254 < 2^8$, thus resulting in $\delta^{err}_{appr} \leq 2^9 \cdot 2^{251} \cdot 2^8 \cdot 2^{-n} = 2^{-n+268}$.

For δ^{calc}_{trunc}, we use the bound [17] $F_{DLap(p)}(-B-1) = \frac{p^{-B-1}}{1+p} < p^{-B-1}$. Conveniently, the term p^{-z} arises during the computation of the pmf's of the respective discrete distributions, which means that we can reuse the bounds we have for its accuracy. Concretely, we have $\delta^{calc}_{trunc} \leftarrow (\Delta^{-1}\tilde{p})^{|B+1|}$ and have $\delta^{err}_{trunc} \leq 2\binom{B+1}{\lceil (B+1)/2 \rfloor} B 2^{-n} < 2^{-n+256+8+1} = 2^{-n+265}$

For the discrete Laplace, we get in the end

$$\begin{aligned} \delta_{appr} &\leq \delta^{calc}_{appr} + \delta^{err}_{appr} + \delta^{calc}_{trunc} + \delta^{err}_{trunc} \\ &\leq \delta^{calc}_{appr} + \delta^{calc}_{trunc} + 2^{-n+269}. \end{aligned}$$

C.2 Bounding δ^{err}_{appr}, δ^{err}_{trunc} for the Gaussian

The pmf of the discrete Gaussian distribution is given by $f_{\mathcal{N}_\mathbb{Z}}(z) = \frac{e^{-z^2/(2\sigma^2)}}{\sum_{y \in \mathbb{Z}} e^{-y^2/(2\sigma^2)}}$, however we again truncate it to a bound $[-B, B]$ as the extreme concentration of the distribution quickly makes it infeasible to represent the masses in its tails. Given a truncation bound, we get $f_{trunc\mathcal{N}_\mathbb{Z}}(z) = \frac{e^{-z^2/(2\sigma^2)}}{\sum_{y=-B}^{B} e^{-y^2/(2\sigma^2)}}$. We compute the elements of the truncated pmf by computing each numerator individually and then at the end we divide each by their sum. If we denote the numerator for z by d_z then we get $d_z \leftarrow e^{-z^2/(2\sigma^2)}$ and denote their sum by d_{tot}. We compute d_z by first computing $x \leftarrow z^2/2\sigma^2$ and then e^x. Since our implementation (based on the *fastnum* library[4]) can perform both multiplications and division with full bit precision, the error in computing x is at most 2^{-n}. Similarly, the exponential function is computed with precision accuracy unless there are under- or overflow errors, which are, in that case, caught and raise an error message. Since the d_z's are never larger than 1, the total absolute error in our approximation of $e^{-z^2/(2\sigma^2)}$ is at most $2^{-n} \cdot e^{-z^2/(2\sigma^2)+2^{-n}} < 2^{-n+1}$. This in turn implies that the computation of d_{tot} is at most $(2B+1)2^{-n+1}$ wrong. Finally, the error in the computation of d_z/d_{tot} is at most $2^{-n+1}/(d_{tot} + (2B+1)2^{-n+1}) + 2^{-n} \leq 2^{-n+1}/(1 + (2B+1)2^{-n}) + 2^{-n+1} \leq 2^{-n+1} + 2^{-n+1} \leq 2^{-n+2}$. Thus we get $\delta^{err}_{appr} \leq (2B+1) \cdot 2^{-n+2}$.

For δ^{calc}_{trunc}, we use Proposition 25 in [7] which proves that the discrete Gaussian is strictly more concentrated than the corresponding continuous Gaussian and then we apply a standard Chernoff bound to bound the cumulative density function of the continuous Gaussian. Letting the continuous Gaussian be denoted $\mathcal{N}(0, \sigma^2)$, this gives us $F_{\mathcal{N}_\mathbb{Z}(0,\sigma^2)}(-B-1) \leq F_{\mathcal{N}(0,\sigma^2)}(-B) \leq e^{-\frac{B^2}{2\sigma^2}}$. Again, the error we incur when computing a term $e^{-\frac{B^2}{2\sigma^2}}$ is already computed above. So we get $\delta^{calc}_{trunc} \leftarrow 2 \cdot e^{-\frac{B^2}{2\sigma^2}}$ and $\delta^{err}_{trunc} \leq 2 \cdot 2^{-n+1} = 2^{-n+2}$. Finally, we get

$$\begin{aligned} \delta_{appr} &\leq \delta^{calc}_{appr} + (2B+1) \cdot 2^{-n+2} + \delta^{calc}_{trunc} + 2^{-n+2} \\ &\leq \delta^{calc}_{appr} + \delta^{calc}_{trunc} + 2^{-n+15}. \end{aligned}$$

[4] https://crates.io/crates/fastnum