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. We propose a novel method for using lookup tables (LUTs) in multiparty noise sampling which allows using smaller and/or fewer LUTs compared to previous work (Franzese et al., CCS'25, and Kii et al., PETS'25), leading to efficiency improvements of several orders of magnitude. This is primarily achieved by not evaluating the LUTs at uniformly random indices but rather using a non-uniform index distribution that can be sampled efficiently. Our method is largely distribution-agnostic, and we demonstrate its flexibility by approximating the discrete Laplace and Gaussian distributions (for a wide range of parameters) to a negligible statistical distance. Our concrete implementation, based on 3-party replicated secret sharing, achieves sub-kilobyte communication and millisecond-level computation. Amortized over 1000 discrete Laplace or Gaussian ($\sigma \leq 1\,000$) samples, we require just 619 bytes of communication (constituting over two orders of magnitude less communication than all previous work) and under 2 ms per party in the semi-honest setting, in all considered network configurations. With recently proposed batched multiplication checks, the malicious setting adds less than 1 byte and 5 ms per sample (amortized). Offering potential independent value, our open-source implementation also extends LUT communication trade-offs (Morita et al., USENIX'25), thus enabling more efficient evaluation of larger LUTs. Further, the merits of our sampling method are not contingent on the above MPC setting and we give analytical performance estimations for use with other MPC machinery, which indicate that it widely is a competitive alternative, especially for somewhat concentrated distributions.

Keywords: Differential Privacy, Multiparty Noise Sampling, Lookup Tables

1 Introduction

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. The literature on multiparty noise sampling for DP was initiated in 2006 by Dwork, Kenthapadi, McSherry, Mironov and Naor [18]. At that time, both DP and MPC were far away from practical implementations so it is only in recent years that more multiparty sampling protocols have been designed [12,19,33,41] [21,30,20,32]. Most of the existing works implement either a version of the discrete Laplace mechanism (i.e. geometric mechanism [24]) or the discrete Gaussian mechanism [11] (alternatively, the binomial mechanism [18]). None of the multiparty sampling protocols sample the noise distribution exactly 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 LUT-based Noise Sampling

The potential of using LUTs to make multiparty noise sampling more efficient was recently demonstrated in two other works [20,32]. Both make the observation that any discrete noise distribution with finite support can be sampled arbitrarily well by evaluating some sufficiently large LUT at a uniform index. The problem, however, is that to get approximation errors small enough for usage in DP (essentially negligibly small) one would need LUTs of impractical size. This problem is overcome in two different ways:

1. Franzese et al. [20] create a tree of LUTs (each evaluated at a uniform index), where each element in a LUT is either a value in the support of the target distribution Z or the result of evaluating another LUT. By calibrating the size of the tables to the support of Z, this approach allows arbitrarily low approximation error for any discrete distribution with finite support by simply increasing the number of LUTs evaluated (the depth of the tree).
2. Kii et al. [32] instead note that the discrete Laplace distribution can be decomposed into one-sided geometric distributions. They use this observation by generating each of the geometric samples through uniform evaluation of a small LUT, and these samples are then combined into an approximate discrete Laplace distribution by computing a shallow circuit in MPC.[1]

So both of these related works circumvent the problem of having to evaluate one very large LUT by instead evaluating a few smaller LUTs. Clearly, the efficiency is optimized by have smaller and/or fewer LUTs. Our road towards that goal is based on the observations that the *typical noise distributions in DP are highly concentrated* (i.e. have most of their mass on a small part of their support) and that *the LUTs do not have to be evaluated at a uniformly chosen index.* This means that if we introduce an index distribution which increasingly often results in evaluating the LUT on a small number of cells, we might get away with using a much smaller LUT than would be necessary if the index was uniform. To see this, consider the trivial edge-case where the target distribution Z has some small support $S \subset \mathbb{N}$ and the index distribution I has at most $2^{-\lambda}$ statistical distance to Z. Then the LUT can simply be the identity function of its index ($\mathtt{L}(i) = i$) and thus the LUT need only have $|S|$ cells, as compared to the roughly $2^{\lambda} \gg |S|$ cells needed in the worst case for a LUT with uniform index. Of course, if the index distribution is already a good approximation of the target, the LUT is not needed at all, but the hope is that by using a non-uniform index which is still far from the target, we can get a good approximation whilst using a single somewhat small LUT.

1.2 Overview of Our Construction

We consider a *target distribution* Z, truncated to have finite support $S \subset \mathbb{Z}$, and want an MPC protocol which approximately samples from Z with a bounded approximation error. For $k \in \mathbb{N}$, we use a lookup table $\mathtt{L} \in S^{2^k}$ and denote the i'th element of $\mathtt{L}$ by $\mathtt{L}(i)$. This index i is drawn from some *index distribution* I on $[0, \ldots, 2^k - 1]$. The computation pipeline of our construction is that the parties first together generate some uniform random seed $\mathbf{r}$, which is then transformed into an index i drawn from I and finally the output is given by $\mathtt{L}(i)$. Thus, the output has distribution

[1] Kii et al. does not claim an approximation of the discrete Laplace distribution in terms of statistical distance and instead prove directly that their composed Laplace-like distribution itself gives DP guarantees.

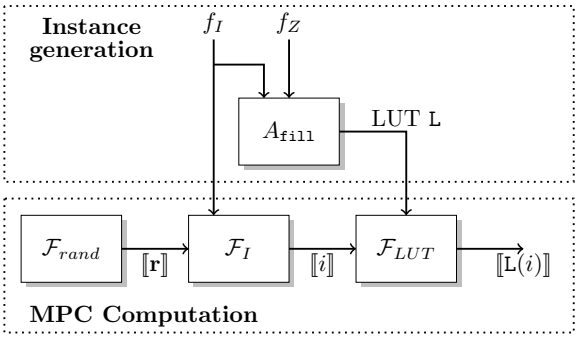

Fig. 1: Schematic overview with target distribution Z, index distribution I and respective mass functions f_Z, f_I.

$\mathtt{L}(I)$. 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 ideal functionalities $\mathcal{F}_{\texttt{rand}}, \mathcal{F}_{\texttt{I}}$ and $\mathcal{F}_{\texttt{LUT}}$ that allow for the secure generation of random bits, translating those bits into an index and evaluation of a lookup table, respectively.

This pipeline requires a fixed index distribution I and LUT $\mathtt{L}$. The LUT needs to be constructed (or *filled*) with respect to I and the target Z, and this procedure is represented by the algorithm $A_{\texttt{fill}}$ in Figure 1. We refer to the construction of $\mathtt{L}$ (and choice of I) as instance generation, since it can be done offline and need not be repeated for future executions of the protocol. When realizing our blueprint for a given target distribution Z (e.g. discrete Laplace or Gaussian), there are **three main design considerations**:

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

These decisions are closely intertwined, as some choice may constrain another, like the index distribution determining the size of the LUT. 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[2] and does not need to be re-generated for future executions of the protocol. 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.

[2] Letting the LUT be public is unproblematic as it does not reveal any information about the final sample, apart from that it is an approximate sample from Z.

In Section 3, we present the detailed design of our protocol and in Section 4, we give the security and accuracy arguments. We show that for a wide range of parameters for the discrete Laplace and Gaussian distributions, our procedure is sufficient to get negligible approximation errors and practically small LUTs.

1.3 Contributions

- We propose a new approach for LUT-based sampling which avoids the use of multiple LUTs by using a nonuniform index. We give a concrete protocol for our construction, prove its security in the UC security framework, and show it can approximate discrete Laplace and Gaussian distributions (with common parameters) up to a negligible statistical distance.
- To demonstrate practical efficiency, we provide an open-source implementation using 3-party honest-majority replicated secret sharing.[3] 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.

1.4 Further Related Works

Generally, there are four main multiparty noise sampling approaches:

- *Distributed sampling* [21]: Each party locally samples a noise distribution such that the sum of the noises has the desired distribution.
- *'Bit-wise' sampling*[12,19,33,41]: This approach stems from [18], 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 intricately biased coins.
- *Rejection sampling*[30,21]: The idea is to sample a cheap proposal distribution and then reject based on the outcome of a biased coin.
- *LUT-based sampling*[20,32]: The sample is generated by (primarily) evaluating some number of specifically constructed LUTs at a random index.

Rejection sampling generally has the benefit of requiring a very small amount of computation on average, making it suitable for sampling in the plain [11]. In MPC, however, it necessitates fixing the number of repetitions a priori (as to avoid branching on secret data) and therefore introduces a failure probability to account for the possibility that no valid sample was generated within those repetitions. Bit-wise sampling has the benefits of not having such a failure probability and that it is simple since the main procedure consists only of sampling biased coins. LUT-based sampling (Section 1.1) also avoids the failure probability of rejection sampling but introduces new challenges, as treated in the rest of this paper, such as how the LUT should be efficiently generated and evaluated.[4] 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

[3] https://github.com/Fable95/lut_sampler

[4] We note that the approaches are not mutually exclusive and can often be combined as to improve performance, for instance by using bit-wise sampling to generate a discrete Laplace sample and then turning that into a discrete Gaussian sample via rejection sampling [21,11].

output. This leads to a worse utility/privacy tradeoff when outsiders learn the mechanism output. We therefore compare only to the other three approaches. Existing work differ on the side of MPC in terms of number of parties, security setting and techniques applied, as can be seen in Table 1. We defer practical comparisons between and discussion of existing implementations to Section 5.

Table 1: Summary of recent related works, grouped together when they use versions of the same protocol. '$\mathcal{N}_{\mathbb{Z}}$' denotes (versions of) the discrete Gaussian and 'DLap' similarly for the discrete Laplace distribution. 'SH' denotes semi-honest and 'Mal' malicious security. 'Rej' denotes rejection sampling and 'BW' bitwise sampling.

Paper(s)	**Distr.**	**Parties**	**Security**	**Method**
[30]	$\mathcal{N}_{\mathbb{Z}}$/DLap	2,3	SH	Rej
[12,21,41]	$\mathcal{N}_{\mathbb{Z}}$/DLap	Any	Both	BW/Rej
[19,33]	DLap	2	Mal	BW
[32]	DLap	2	SH	LUT
[20]	$\mathcal{N}_{\mathbb{Z}}$	Any	SH	LUT
This	$\mathcal{N}_{\mathbb{Z}}$/DLap	3	Both	LUT

In multiparty noise sampling, the security properties of concrete protocols are uniquely determined by the MPC schemes used. This means, in principle, that each protocol can be instantiated with both static and adaptive corruptions, both active and passive adversaries, any threshold on the fraction of corrupted parties, and with either security-with-abort or some version identifiable abort etc., given that the sub-procedures of the protocols are available with such security properties. How practical each approach is in various security settings is, however, so far not well-understood. Whereas the related works have been implemented with differing numbers of parties, fraction of corrupted parties and differ in considering active or passive corruptions, all existing implementations consider static corruptions and security-with-abort.

1.5 Setting and Adversarial Model

Our sampling method, as such, 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 [9] and operate in the setting of static corruptions, security-with-abort, and honest-majority[5]. We formulate ideal functionalities and security arguments using the Simple UC (SUC) framework [10], 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 [15] 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) [2,34] guarantees.[6]

[5] For an overview of how our protocols may be instantiated in other security settings, and consequences for performance, see Section 5.2.

[6] Note that, due to reliance on computationally secure MPC (for efficiency), essentially all works on multiparty noise sampling result in CDP [33,19,41,30].

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 $p^z(1-p)^{1-z}$ for $z \in \{0, 1\}$. We denote the discrete Laplace distribution [24,26] with parameter $p \in (0, 1)$ as $DLap(p)$ and its pmf is $\frac{1-p}{1+p}p^{|z|}$ for $z \in \mathbb{Z}$. We denote the discrete Gaussian distribution [11] by $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ and its pmf is $\frac{e^{-z^2/(2\sigma^2)}}{\sum_{y\in\mathbb{Z}} e^{-y^2/(2\sigma^2)}}$ for $z \in \mathbb{Z}$. We denote the indicator function by $\mathbb{1}$. By w, we denote the bit-width of the elements in the lookup table, and 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. For a positive integer m, $[m]$ denotes $\{0, 1, \ldots, m-1\}$.

2.1 Replicated Secret Sharing

We use replicated secret sharing (RSS) [28] with bit-wise sharings (see e.g. [35,36]). That is, a value in $\mathbb{Z}_{2^w}$ is interpreted as a w-length vector of binary values (i.e. a vector in $\mathbb{F}_2^w$), and $[\![x]\!]$ denotes the bit-wise RSS share of this vector. Thus, the output of our protocols will be in $\mathbb{F}_2^w$, but can be cheaply transformed into the arithmetic domain (see [35], for instance), resulting in bounded signed integers in $\mathbb{Z}_{2^w}$.

Performance-wise, RSS is typically used in a computational setting, relying on the security of PRF evaluations. Shares of pseudorandom elements can be generated using pre-shared PRF keys, without interaction [15,16]. The generation of shares of 0, multiplication of two shares and dot products of arbitrary length all require one round of interaction, sending one element in $\mathbb{F}_2^w$ (i.e., w bits of communication). For multiplication and dot product evaluations, the parties need to verify the validity of the resulting share when considering active (malicious) corruptions. We refer to RSS multiplications that need verification as *weak multiplications*. For more background on RSS and its security properties, we refer to [16,35,22].

2.2 Lookup Tables in MPC

Recently, there has been a rising interest in MPC lookup table evaluation. Many 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 $[\![\mathtt{L}(x)]\!] \leftarrow \langle [\![e^{(x)}]\!], \mathtt{L}\rangle$. There are constructions in the 2-party-plus-helper setting for Function Secret-Sharing (FSS) in the binary domain [6] and over integer rings [40]. Without a helper, there are 2-party protocols based on ABY-2 optimizing for low online communication [27], lower total communication [17], or a trade-off between online and total communication [7]. Further, Heath et al. propose a solution for Garbled Circuits (GC) [25], that requires only logarithmically many ciphertexts. Franzese et al. [20] propose

a one-hot vector generation protocol enhancing the lookup capabilities of mixed-mode MPC [23], leveraging threshold homomorphic encryption to scale the number of parties. In MAESTRO [36], 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 our method for improved LUT-based sampling, outline the design constraints of the building blocks and detail our concrete sampling protocol 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 3). 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 fundamental building blocks that other protocols rely upon, and for these, we utilize existing protocols from the MPC literature [3,36]. In the following subsections, we present an algorithm for filling the LUT, protocols for index sampling and LUT evaluation, and provide an analysis of these. The formal statements of the subprotocols, with security arguments, and other ideal functionalities that the protocols use are discussed in Appendix A.

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 $f_{\texttt{L}(I)}(z)$ approximates $f_Z(z) + f_Z(-z)$ for all non-zero $z \leq B$ and $f_{\texttt{L}(I)}(0)$ approximates $f_Z(0)$.[7] Let Z_{OS} denote this adjusted one-sided target distribution that $\texttt{L}(I)$ approximates. Then we get an approximation of Z by assigning the sign of $\texttt{L}(I)$ at random, leading to an output distribution of

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

3.1 On Optimal Instance Generation

Given a target Z, the instance generation (choosing $k, I, \texttt{L}$) presents an optimization problem with many variables and two competing objectives – low approximation error and concrete efficiency. Firstly, there are, in principle, infinitely many possible index distributions, unless we restrict the number of fair coins used to generate the index. Fortunately, we are only interested in index distributions which can be efficiently sampled, so it seems reasonable to only consider some family of distributions for which we know this is the case. In our instantiation, we fix one such family

[7] This trick of utilizing the symmetry of the target distribution is commonly used in noise sampling protocols [19,30,11].

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: LUT-based sampling

(Section 3.2). Secondly, even after fixing k and I, there are B^{2^k} possible LUTs. In Section 3.3, we argue that filling the LUT optimally appears highly challenging, so we turn to a heuristic approach.

Whilst our instantiation does give highly satisfying results, and we do believe that optimally solving the instance-generation problem in generality is infeasible, we do not rule out that there are better ways of generating instances. It seems plausible that better performance can be reached by, for instance, considering a different family of index distributions or by finding a better LUT-filling algorithm, perhaps by deepening the connection between LUT-filling and well-studied optimization problems. We note that the LUT is public and the instance does only need to be generated once for each target distribution, so it is in practice likely not an issue if the instance generation is somewhat expensive.[8] Therefore, a reasonable approach is to exhaustively try each index distribution (of some given type), fill a LUT for each and then pick the instance which results in the best performance.

3.2 Index Sampling

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 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 $c \in \mathbb{N}$, and let the remaining $k - l$ index bits be uniform. The ideal functionality for sampling indices with this distribution is given in Figure 3. 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 the concatenation of these biased bits and the remaining fair bits. Formally, the protocol consists solely of calls to ideal functionalities $\mathcal{F}_{\mathtt{rand}}, \mathcal{F}_{\mathtt{weakMult}}, \mathcal{F}_{\mathtt{verify}}$ (Appendix A) for uniform random values, multiplication, and verification of those multiplications, respectively.

[8] One party can generate the instance beforehand and then send it to the other parties, who can efficiently compute the resulting approximation error before agreeing to take part in the protocol.

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. 3: The ideal functionality for sampling fair or biased coins.

We now shortly explain how 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 $\mathbb{F}_2^w$ locally. For realizing $\mathcal{F}_{verify}$, we use a protocol from [36], 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 the arising constraints in Section 4.1. Given these subprotocols, the index sampling protocol π_I realizes the canonical packing of bits sampled from $Ber(p_{Ber})$ into $\mathbb{F}_2^w$. 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). In Section 3.5, we further show that the different LUT evaluation methods we use have no impact on the probability distribution in our table cells, which is not true for arbitrary index distributions.

3.3 Filling the LUT

The goal of the LUT filling is to construct a specific $\texttt{L}$ 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 bears resemblance to the *bin packing problem*[9] [29,1] and the *multiple knapsack problem* [13,8] (which are NP-complete) so general and

[9] In the bin packing problem, a list of items, each with some weight, are to be placed in a minimal number of *bins*, each with some capacity, without the total weights in a bin exceeding its capacity. We may think of each element in the range of the target distribution as a bin, with its capacity being its target mass. Then each cell in the LUT, with its weight being its probability, must be assigned to some sample value (bin) without, essentially, exceeding the capacity by 'too much'. There are, however, important differences between the problems, for instance in the objective functions.

efficient error-optimal strategies appear unlikely to be found. We therefore turn to a heuristic filling algorithm, which is inspired by the *best fit decreasing* algorithm (see [29]) for approximately solving the bin packing problem. We propose an algorithm, $A_{\texttt{fill}}$, that results in satisfactory approximation errors for the discrete Laplace and Gaussian distributions. We leave strengthening the ties between LUT-filling and classical combinatorial optimization problems, exploring alternative filling algorithms, and studying their potential approximate optimality for future work.

$A_{\texttt{fill}}$ constructs $\texttt{L}$ using a combination of a 'greedy' initial step and an exhaustive step at the end, as outlined in Figure 4. Let $f_{Z_{OS}}$,f_I be the pmf's of Z_{OS} and I, respectively, and throughout, let $\texttt{L}(i)$ denote the i'th element of the current version of $\texttt{L}$. $A_{\texttt{fill}}$ proceeds as follows. First, we set $\texttt{L}$ to be empty (all cells contain some blank symbol $\perp$) and consider each index $i \in \mathbb{Z}_{2^k}$ 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 $\texttt{L}$, with $tot_z := \sum_i f_I(i) \cdot \mathbb{1}\{\texttt{L}(i) = z\}$. If $f_{Z_{OS}}(z) \geq tot_z + f_I(i)$ then we set $\texttt{L}(i) \leftarrow z$ and move to the next i, else we move to the next z. At the end of this process, $\texttt{L}$ 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 $z \in [0, B]$.

Algorithm $A_{\texttt{fill}}$

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 $\texttt{L}$.

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

Fig. 4: The LUT filling algorithm $A_{\texttt{fill}}$.

3.4 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[10] computing the approximation error for a concrete instance $(Z, I, \texttt{L})$ can be done efficiently and with errors within said computation being inconsequentially small.[11] For given $Z, I, \texttt{L}$, if we truncate the support to $[-B, B]$ and thus get a *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

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

Computing the statistical distance can, in principle, be done by hand but since we do not have a closed form for $f_{Out_{\pi_Z}}$, this becomes tedious to the point of practical infeasibility. So we turn to numerically upper bounding the distance. This could, naively, be done 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 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

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

To simplify the accounting for rounding errors, we use fixed-point numbers and we let n be the number of bits of precision. Using such fixed-point representation for $f_Z(z)$ introduces rounding errors since the pmfs of the discrete Laplace and Gaussian distribution generally are not perfectly representable using n bits. We restrict the representations of all intermediate values to n bits of precision and track how errors propagate 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.

Since Out_{π_Z} is a deterministic transformation of K fair coins (for some K), as long as $n \ge K$, each $f_{Out_{\pi_Z}}(z)$ can be represented without introducing any error. Due to our simple index distributions, this is always achieved in our instantiations.[12]

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 $B \le 2^w$, and secondly we need B to be small enough that also the smallest mass $f_Z(B)$ can

[10] We use the *fastnum* library (https://crates.io/crates/fastnum) for arbitrary (but fixed) precision in fixed-point arithmetics, giving fine-grained control over rounding errors.

[11] We note that the approximation errors we compute, and thus the resulting DP guarantees, are *not* heuristic or experimental, as they are computed with high accuracy and stringent bounds on numerical errors. The heuristic uncertainty lies rather in that we do not know how close the results (approximation error) are from optimal or how well the filling works for other distributions.

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

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 $\delta_{trunc} = F_Z(-B-1) \ll 2^{-\lambda}$. 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 $B \leftarrow \min\{2^w - 1, B^*\}$.

As the detailed derivations of the specific upper bounds on computational error are long and without conceptual novelty, we defer them to Appendix D. With $n \in \{512, 8192\}$, we get

- *Discrete Laplace* ($B \leq 2^{12}$): $\delta^{err}_{appr} + \delta^{err}_{trunc} \leq 2^{-233}$,
- *Discrete Gaussian* ($B \leq 2^{16}$): $\delta^{err}_{appr} + \delta^{err}_{trunc} \leq 2^{-472}$.

In Section 5.1, we report on the values of δ^{calc}_{appr} and δ^{calc}_{trunc} for a wide range of distribution parameters. To get a bound $\delta_{appr} \leq 2^{-\lambda}$, it is sufficient for δ^{calc}_{appr} and δ^{calc}_{trunc} to be at most $2^{-\lambda-2}$. So clearly, the bounds above make the computation error inconsequential with respect to all considered values of λ in our evaluations (cf. Section 5.1).

3.5 Lookup Table Evaluation

In our construction, the wish for a large lookup table (for the purpose of a low approximation error) and the wish for a small LUT for efficiency clearly compete against each other. We begin by noting that the basic blueprint for LUT evaluation, as outlined in the preliminaries, exhibits a rather extreme offline-to-online trade-off, characterized by exponential preprocessing and linear online costs. In our work, we do not shift communication to preprocessing but instead aim to minimize the total communication cost. This choice reflects our view of noise generation as a data-independent protocol, essentially removing the traditional argument in favor of a preprocessing phase. We now extend existing techniques to make larger LUTs practical for our usage.

Higher-Dimensional LUTs Consider the idea from MAESTRO [36] where the authors reduce the preprocessing cost with a small additional cost in the online phase. We reiterate that this trade-off can be applied in all secret sharing schemes that allow for dot product computation, with communication independent of the vector size (e.g., Shamir Secret Sharing [39], RSS [28,35], or the 2-party ABY2 [38]). We phrase this following explanation with a uniform index for simplicity, and adapting the protocol to use our biased index is straightforward.

The first step is to sample two one-hot vectors $(\llbracket s \rrbracket, \llbracket e^{(s)} \rrbracket)$, $(\llbracket t \rrbracket, \llbracket e^{(t)} \rrbracket) \leftarrow \mathcal{F}_{\texttt{RndOHV}}(2^{k/2})$ and compute the combined index by concatenating the partial index bits $\llbracket r \rrbracket \leftarrow \llbracket s || t \rrbracket$, which requires no interaction. Then, one extends the individual one-hot vectors into

$$\llbracket \mathbf{a} \rrbracket \leftarrow (\underbrace{\llbracket e^{(s)}_0 \rrbracket, \cdots, \llbracket e^{(s)}_0 \rrbracket}_{\times \frac{k}{2}}, \ldots, \llbracket e^{(s)}_{k/2-1} \rrbracket, \ldots, \llbracket e^{(s)}_{k/2-1} \rrbracket)$$

$$\llbracket \mathbf{b} \rrbracket \leftarrow (\llbracket e^{(t)}_0 \rrbracket, \cdots, \llbracket e^{(t)}_{k/2-1} \rrbracket, \ldots, \llbracket e^{(t)}_0 \rrbracket, \ldots, \llbracket e^{(t)}_{k/2-1} \rrbracket).$$

The LUT is then evaluated as $\langle \llbracket \mathbf{a} \rrbracket, \llbracket \mathbf{b} \rrbracket \circ \mathtt{L} \rangle$, 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 $\mathtt{L}$) 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 $\mathtt{L}$ is public, computing $\llbracket \mathbf{b} \rrbracket \circ \mathtt{L}$ incurs no communication, leaving only the final dot product.

At a high level, one can think of the above approach as a two-dimensional lookup, where r' and r'' represent the row and column indices. From this perspective, we extend the approach to lookup tables of higher dimensions. Starting in the three-dimensional case (full protocol in Figure 11 in Appendix B.2), we sample 3 one-hot vectors of bit-lengths k_1, k_2, k_3 – one for the row, column, and layer of the cell index such that $k_1 + k_2 + k_3 = k$. 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 of length 2^{k_2} and results in a final secret shared vector, which in turn requires one more dot product. Generalizing the communication cost for d dimensions, we first compute the cost of one-hot vector generation as

$$\sum_{i=1}^{d}(2^{k_i} - k_i - 1).$$

Second, we need to account for the exponential impact of additional dimensions when evaluating the lookup table. Concretely, we need to iteratively remove each dimension of our table. With a public table, we can remove the first dimension without communication. Afterwards, removing dimension $i > 1$ requires $2^{k_{i+1}} 2^{k_{i+2}} \cdots 2^{d}$ dot products of length 2^{k_i}. The final dimension (i.e., the LUT is a vector) incurs one dot product of length 2^{k_d}. Compactly, this evaluation method incurs

$$w \cdot \left(\sum_{i=3}^{d} 2^{\sum_{j=i}^{d} k_j} + 1 \right),$$

bits of communication, within d rounds.

Finding d and $k_1, \ldots, k_d$ such that the total communication is minimized is an optimization problem parametrized by w and constrained by $\sum_{i=1}^{d} k_i = k$. Given that k_1 and k_2 only appear in the cost of OHV generation, which is a convex function, their impact is minimized by setting $k_1 \approx k_2$. Noting further that practical sizes for k are limited by memory capacity and thus well below $k = 64$, finding the most efficient $d, k_1, \ldots, k_d$ can be done in seconds by exhaustive search. In Appendix C in Table 6, we provide optimal settings for selected choices of w and k.

For our parameter regime, the most efficient d lies between 2 and 4. For our largest table with $k = 24$ and table entry size $w = 16$, the setting of $k_1 = 10, k_2 = 9, k_3 = 5$ incurs 29% of the total communication compared to the previous method with $d = 2$, saving 6113 bits of total communication, cutting 7425 bits of preprocessing, but adding 1312 bits of online communication. We note that whilst we base our implementation on MAESTRO [36] 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.

4 Security and Privacy Considerations

In this section, we first discuss protocol π_{verify}, which realizes $\mathcal{F}_{verify}$, and how we instantiate it in our implementation. 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 Verification of Multiplication

We base the verification of our multiplication and dot product evaluations on Protocol 2 in MAESTRO, which securely realizes $\mathcal{F}_{\texttt{verify}}$ [36]. The protocol takes m dot product triples $(\mathbf{x}_i, \mathbf{y}_i, z_i) \in \mathbb{F}^l \times \mathbb{F}^l \times \mathbb{F}$, for some field $\mathbb{F}$, and verifies whether $z_i = \langle \mathbf{x}_i, \mathbf{y}_i \rangle$ for all i. Let $N = \sum_i |\mathbf{x}_i|$ be the total length of all vectors to verify. The proposed protocol that realizes $\mathcal{F}_{\texttt{verify}}$ requires $\log_2 N$ rounds, where it uses the input triples to generate $N/2$ randomized degree-1 polynomials a_i, b_i, and a degree-2 polynomial c which together satisfy a dot product relation. In the final step, the remaining multiplication triple is checked without revealing anything beyond the verification result. In total, with probability at most $(m + 2\log N)/|\mathbb{F}|$, the final evaluation check fails to detect incorrect triples. Thus, the security of this protocol relies on $|\mathbb{F}|$ being large and m being significantly smaller. We will first discuss how to embed our small domain triples into this larger field, and further discuss the resulting implications for our construction.

Our protocols π_I and $\pi_{\texttt{RndOhv}}$ require verification of triples in $\mathbb{F}_2$, which can be canonically embedded into $\mathbb{F}_{2^{64}}$ since $\mathbb{F}_2$ is the prime subfield of any $\mathbb{F}_{2^k}$ [37]. $\pi_{\texttt{LUT}}$ has dot product triples $(\mathbf{x}, \mathbf{y}, c)$, where $x_i \in \mathbb{F}_2, y_i \in \mathbb{F}_2^w, z \in \mathbb{F}_2^w$, and $|\mathbf{x}| = |\mathbf{y}| = 2^{k_i}$. We distinguish two cases for their embedding. First, the reduction of the final dimension requires verification of a single dot product. Due to the domain $x_i \in \mathbb{F}_2$, note that z_i is either y_i or 0, and z is the bit-wise sum of all z_i. It follows that even under the canonical embedding of our values into $\mathbb{F}_{2^{64}}$, the dot product relation for these triples holds. Second, the reduction of dimension $i < d$, we verify $2^{\sum_{j=i+1}^{d} k_j}$ dot products of length 2^{k_i}. However, these triples $\{\mathbf{x}, \mathbf{y_i}, z_i\}$ all share a common one-hot vector $\mathbf{x}$. We set the bit-width w to divide 64 and $\tau = 64/w$ to divide the size of our last dimension 2^{k_d}. For our input to $\mathcal{F}_{\texttt{verify}}$, we concatenate τ many values in $\mathbb{F}_2^w$, thus mapping them into $\mathbb{F}_2^{64}$, which we then canonically embed into $\mathbb{F}_{2^{64}}$ effectively reducing the number of required triples by τ.

After lifting all product and dot product triples to $\mathbb{F}_{2^{64}}$, we can derive the total N and m of our construction and evaluate the impact on the failure probability. For π_I and $\pi_{\texttt{RndOhv}}$, we record each multiplied bit individually, resulting in $(c-1) \cdot l$ and $\sum_{i=1}^{d}(2^{k_i} - k_i - 1)$ recorded triples of length 1. When evaluating the LUT, we record $\frac{1}{\tau}(\sum_{i=3}^{d} 2^{\sum_{j=i}^{d} k_j} + \tau)$ dot products of a total length $\sum_{i=1}^{d} 2^{k_i}$. Given concrete parameters l, k, d and the choices $\{k_i\}$ for $i \in [d]$, it is easy to compute the maximum number of iterations permissible before the failure probability is unacceptable. Fixing a maximum failure probability of 2^{-40}, we arrive at an upper bound $\eta \leq 2^{13}$ for all parameter regimes considered in this paper.

The straightforward option for lowering the failure probability to below 2^{-40} is increasing the field size. This will increase the security margin and decrease the number of triples slightly, with no significant impact on the protocol beyond performance.

4.2 Security

We prove security using the SUC framework [10] in the setting of static corruptions, honest-majority and security-with-abort. For details about the framework, we refer to [10]. 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 Appendices A and B, respectively. The proof for the theorem below is given in Appendix B.3.

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

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

2 : $\mathcal{F}_{\mathtt{Z}}$ 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 : $\mathcal{F}_{\mathtt{Z}}$ 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 version of LUT-based sampling.

Theorem 1. *The protocol π_Z (Figure 2) securely realizes $\mathcal{F}_{\mathtt{Z}}$ (Figure 5) 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.*

4.3 Privacy

Since the protocol is UC-secure, one can easily prove that a mechanism implementation where the query evaluation is computed UC-securely and then the noise from our protocols is added is computationally DP (CDP).[13] 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. Both of these can straight-forwardly be included into the δ-parameter of the DP guarantees, as is commonly done [41,30].

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 Sections 5.1, 5.1, and 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 [21,33,32,20]. All our benchmarks are run on a Linux machine with an AMD Ryzen 9 7900X CPU (4.7 GHz). We give runtime data for single-threaded

[13] In particular, such a protocol would fulfill the recently proposed SIM*-CDP flavor of CDP [33], which in turn implies both SIM-CDP and IND-CDP [34].

evaluation and communication cost per party. In Tables 3 and 4, we denote single sample evaluation by Latency and round-trip time by RTT.

In Table 2, we list concrete parameter regimes that we considered. Effectively, the sizes k and d influence the size of $\mathtt{L}$, while w specifies 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 3.4, we set $B \leftarrow \min\{65536, B^*\}$ with $B^* \leftarrow \min\{B : \delta^{calc}_{trunc} < 2^{-2\lambda}\}$, and choose w to be the smallest multiple of 8 that fulfills $2^w \geq B$. Further, we consider LUT sizes in the range of $k \in [12, 24]$ and choose $d \in \{2, 3\}$ depending on the cost computation as described in Section 3.5 and concretely calculated in Appendix C Table 5. For our fixed-point computation, we have a precision of $n \in \{512, 8192\}$ bits, which we expand on in Section 3.4. Given these parameters, we fix the maximum batch size for π_{verify} to $\eta = 2^{13}$ following the discussion in Section 4.1.

Table 2: Our considered parameter regimes.

k	d	c	l	w	η
$[12, 24]$	$\{2, 3\}$	$\{2, 12\}$	$[0, k]$	$\{8, 16\}$	2^{13}

5.1 Performance of Subprotocols

We now discuss the concrete performance of all subprotocols and provide a detailed breakdown of the performance impact in Table 3. As a reference point, we select the worst-performing parameters used in our approximation experiments and covered in Table 2. Concretely, we have $k = 24, d = 3, \{k_i\} = \{10, 9, 5\}, c = 12, l = 24, w = 16$.

Table 3: Cost of π_Z, with $d = 3$, $c = 12$, $k = 24$, and $w = 16$ in semi-honest (SH) and malicious (MAL) secure RSS. Communication is given in Bytes, Latency in ms, and throughput in 1/s amortized over 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.

Subprotocol		Localhost		RTT = 1ms		RTT = 100ms		Total Communication		
		Latency	TP	Latency	TP	Latency	TP	Rounds	Single	Amort.
π_I		0.32	150 375	11.2	65 274	800	1 240	12	36	36
$\pi_{\mathtt{LUT}}$	$\pi_{\mathtt{RndOhv}}$	63	12 970	61.1	14 164	2 170	595	25	3210	194
	$Eval$	42.9	666	45.1	662	142	628	2	66	66
π_{verify}		42.1	2 293	57.9	2 217	1 750	322	31	352	0.59
π_Z	(SH)	108	680	107	689	3 510	251	39	3 315	299
	(MAL)	148	495	176	487	4 360	139	70	3 667	300

Index Sampling In our experiments testing approximation error limits for different LUT sizes and target distributions, we vary the two other parameters c and l of our index distribution. We

test c between 2 and 12 and all l lower than k. Recall that these parameters dictate the bias of the bits and the number of biased bits in the index, respectively. The LUT with k index bits is evaluated at indices with the first l bits drawn from $Ber(2^{-c})^l$ and the remaining bits being uniformly distributed.

We discuss the impact on network and runtime cost here and elaborate on the reason and effect in Section 5.1. In particular, the cost of π_I analytically is $c-1$ multiplications per biased index bit (i.e., $(c-1)l$ bits of communication) in $\lceil \log_2 c \rceil$ rounds. For our selected parameters, the communication cost is given as 33 bytes (i.e., 11 multiplications of 24 bits). The 36 bytes as given in Table 3 include the cost of opening the difference to the one-hot vector indices. The runtime of this subprotocol is below one percent of the semi-honest protocol for fast networks. In constrained settings, the round number incurs some delay for single-sample evaluation. Amortized over 1000 samples, it remains the sub-protocol with the highest throughput for all network settings.

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.

For our experiments, we run a grid search over parameters (k, l, c) to evaluate the maximum accuracy achieved through $A_{\texttt{fill}}$ for different target distributions. In Figure 6, we evaluate $A_{\texttt{fill}}$ for tables with $k \in [12, 24]$. Note that due to the multiplicative nature of our index distribution, the LUT dimension d does not impact the available probability masses in the table. Hence, we can evaluate each table with the most efficient dimension setup without changing the distribution Out_{π_Z}. Our grid search further considers, for each target distribution, all $l \in [0, k]$ and $c \in [2, 12]$, and reports the best result below 2^{-40}. As our grid search includes the case of $l = 0$, we always include uniformly distributed indices in our experiments. However, as expected, they never yield satisfying approximations. Following the discussion in Section 3.4, 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 $\lambda := \max\{\lambda \in \mathbb{N} : 2(\delta^{calc}_{appr} + \delta^{calc}_{trunc}) \leq 2^{-\lambda-1}\}$. For $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ and $DLap(p_{DLap})$, we report results for $\sigma = \{2^0, 2^2, 2^2, 2^6, 2^8, 2^{10}\}$ in red and $p_{DLap} = \{e^{-2^0}, e^{-2^{-2}}, e^{-2^{-4}}, e^{-2^{-6}}\}$ in blue[14]. We formulate the distribution parameters as $p_{Dlap} = e^{-\varepsilon}$ and $\sigma^2 = \frac{1}{\varepsilon^2}$ for some positive value ε to reflect the rough relation between the noise distribution parameters and the privacy parameters in (ε, δ)-DP in the case of one-dimensional sensitivity 1 queries [11]. This also allows us to compare fairly to the work of Kii et al. [32] later, since they use a custom target distribution but report on the privacy parameters one gets from using their distribution with queries of sensitivity 1.

From the results in Figure 6, we see a clear relation between table size and accuracy. In general, more concentrated distributions (with larger ε) react better to the biased index distribution and can be approximated with much smaller tables. On the downside, we see that our approach is less accurate for very small values of ε. $A_{\texttt{fill}}$ finds an approximation for $\sigma = 1024$ satisfying $\lambda = 40$ only with $l = 10, k = 22, c = 5$. For higher values of λ, either the index distribution or filling method needs to be improved, or a larger table must be used.

Lookup Table Evaluation We refer to Table 3 with the LUT parameters introduced in Section 5.1. The communication cost and runtime of $\pi_{\texttt{LUT}}$ dominate the overall metrics of our sampling protocol.

[14] Decreasing ε increases our truncation Bound B and our required precision n. Our implementation supports a maximum of $n = 8192$ and is thus limited to $\varepsilon \geq 2^{-6}$.

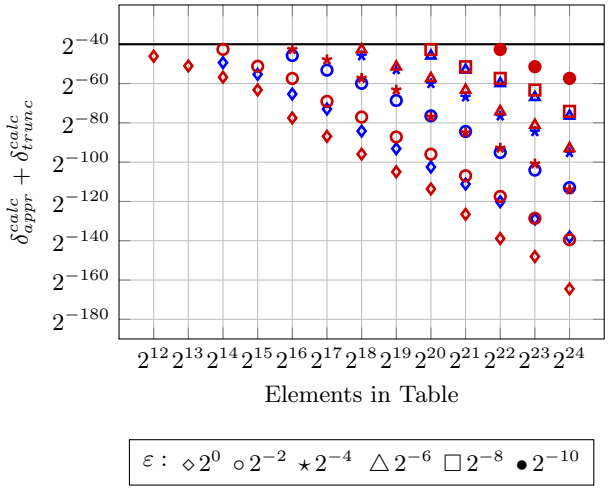

Fig. 6: Lowest approximation error achieved for different table sizes and target distributions. We depict $\mathcal{N}_{\mathbb{Z}}$ in red with $\sigma^2 = \frac{1}{\varepsilon^2}$ and $DLap$ in blue with $p_{DLap} = e^{-\varepsilon}$

When analytically evaluating the communication as outlined in Section 3.5, we account for the cost of generating the one-hot vectors via $\pi_{\texttt{RndOhv}}$, and the cost of reducing all dimensions via $Eval$ (i.e., extracting a table element using multiple vectors) separately. For our cost breakdown, analytically we get 1544 bits for $\pi_{\texttt{RndOhv}}$ and 528 bits for $Eval$, rounding to 196 and 66 bytes, respectively. Averaged over 1000 samples, the implementation is almost fully amortized, communicating exactly $194 + 66$ bytes. Considering the different network latencies, we see that $\pi_{\texttt{RandOhv}}$ is more heavily penalized as it has a higher communication cost and round complexity. However, when accounting for throughput, high network cost is much better amortized than the runtime cost of collapsing the 2^{24} element table in $Eval$, equaling out performance in constrained networks. To further optimize the runtime of $Eval$, we provide a fully byte-sliced implementation that leverages the binary shares for faster multiplication. As a result, the single-shot network cost is slightly inflated as the implementation operates on larger machine words.

5.2 Comparison to Other Works

In the preceding sections, we demonstrate that our construction can accurately sample from the Gaussian and Laplace distributions for wide parameter regimes efficiently. Now, we provide more context on that efficiency by comparing it to related work. First in Table 4, we present benchmark numbers for recent Laplace [30,33,21,32] and Gaussian [30,21,20] sampling strategies. Since all of these protocols operate within different MPC schemes, with varying security settings and party numbers (cf. Table 4), we evaluate them in different contexts considering various metrics. While Franzese et al. [20], Fu and Wang [21], and Keller et al. [30] support different numbers of parties, and Meisingseth et al. [33] could be extended to more parties, we only provide data for the respective best-performing setting in our benchmark environment, for the sake of brevity. Additionally, for Keller et al. and Kii et al., we were unable to reproduce the results and report the numbers presented in the respective papers, selecting the most favorable datapoints instead. For Fu and Wang, we

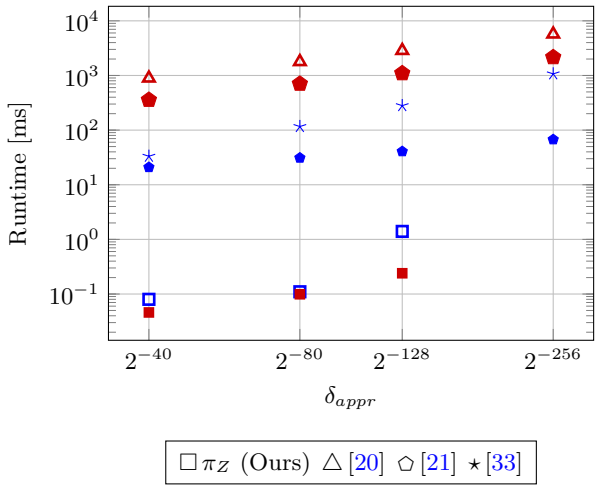

Fig. 7: Amortized runtime of different sampling strategies under different settings of λ, with $\sigma = 1$ and $p_{DLap} = e^{-1}$. *DLap* is depicted in blue and $\mathcal{N}_{\mathbb{Z}}$ in red.

Table 4: Comparison to related works. Latency is given in ms, and communication in KB. All protocols are instantiated with $\lambda = 40$ and $\varepsilon = 1$ s.t. $\sigma = 1, p_{DLap} = e^{-1}$. Grayed out values are taken from the original paper. The network settings are as in Table 3. Security settings are given by ○ : Semi-Honest + Honest-Majority, ◐ : Malicious Security + Honest-Majority, ◑ : Semi-Honest + Dishonest-Majority, ● : Malicious Security + Dishonest-Majority.

Distr.	Protocol	Security-	Localhost		RTT = 1ms		RTT = 100ms		Total Comm.	
		Setting	Latency	TP	Latency	TP	Latency	TP	Single	Amort.
2-Party Setting										
DLap	M_{DLap} [30]	◑	1643	1.01	-	0.21	-	0.03	492720	-
	FDL_2 [33]	●	**116.6**	28.9	431.2	14.3	31742	1.7	65300	23800
$\mathcal{N}_{\mathbb{Z}}$	M_{Gauss} [30]	◑	6254	0.17	-	0.09	-	0.02	1085820	-
	DE-Sample [20]	◑	-	1.12	-	1.12	-	1.06	-	393.1
DDLap	π_{DDLap} [32]	◑	**11.2**	-	-	-	-	-	200	-
3-Party Setting										
DLap	Ostack [21]	○	75.7	59.88	913	4.15	73010	0.04	6994	1202
		◐	353.5	19.84	2824	2.23	174098	0.02	48964	6738
$\mathcal{N}_{\mathbb{Z}}$	ODO [21]	○	1271	4.01	27944	0.18	2607425	0.001	10752	2057
		◐	1755	2.82	35125	0.14	37935700	-	58618	11404
Either	π_Z	○	42.2	**8 000**	**59.2**	**7 575**	**1 580**	**613**	**0.4**	**0.05**
		◐	126	**5 154**	**160**	**5 128**	**6 660**	**250**	**0.7**	**0.05**

evaluate the best performing candidates for the discrete Laplace and Gaussian distributions. As for

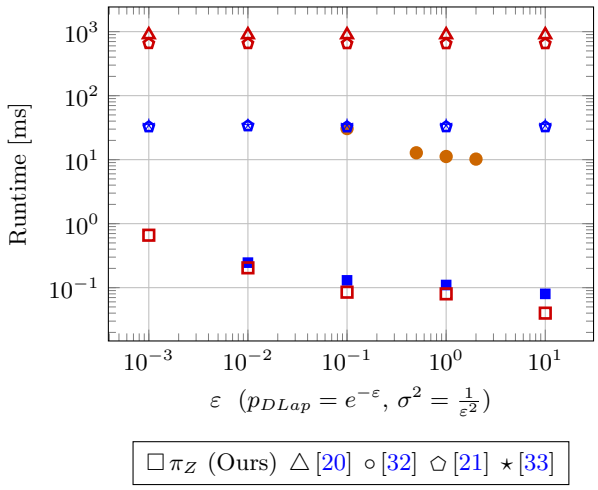

Fig. 8: Amortized runtime to sample for different concentration parameters with $\lambda = 40$. *DLap* is depicted in blue and $\mathcal{N}_{\mathbb{Z}}$ in red, and DDLap [32] in orange.

Kii et al. [32], they propose a distribution for which they give no bounds on approximation error with respect to the discrete Laplace distribution (more on this in the next subsection). Therefore, we use the performance numbers reported in their work, corresponding to distributions that yield similar privacy parameters as those in other works. In the following, we refer to the distribution that they do sample as DDLap.

Evaluation within varying network settings With these caveats in mind, we first compare the protocols with privacy parameter $\varepsilon = 1$ and approximation parameter $\lambda = 40$. We do not distinguish between *DLap* and $\mathcal{N}_{\mathbb{Z}}$ for our construction, and instead use the first table size with sufficiently small statistical distances, as can be seen in Figure 6, a table with $k = 14$ suffices for both distributions. In the presented setting, π_Z outperforms related work in all network settings in amortized performance. Our protocols $\pi_I, \pi_{\texttt{RndOhv}}$ always generate a small minimum number of biased indices and one-hot vectors, respectively. This effect is due to the byte-sliced implementation, as discussed in Section 5.1, which explains the significant amortization in runtime and network cost of our construction. However, for single-sample evaluation, Kii et al. have a 4x faster runtime in the LAN setting.

For $\varepsilon = 1$ and the small table associated with it, our construction incurs an amortized communication cost of below 52 bytes for the malicious and semi-honest settings, while single-shot evaluation incurs 650 and 394 bytes of communication in the malicious and semi-honest settings, respectively. While Table 4 gives a detailed overview, we had to select a specific parameter set. Our implementation is tailored towards the 3-party setting, and in the remainder of this section, we aim to provide a nuanced view, showing the performance in relation to the approximation and privacy parameters λ and ε.

Evaluation within varying λ and ε In Figure 7, we show the amortized performance for different approximation accuracies. We denote the *DLap* in blue and $\mathcal{N}_{\mathbb{Z}}$ in red and fix $\varepsilon = 1$. The strength of our construction is that both distributions can be approximated with similar accuracy, given

specific table parameters and ε. This is due to the approximation error being an upper bound and the table settings not being as fine-grained. On the downside, we observe that while our approach is the fastest for the supported settings, it is limited in the accuracy achievable per privacy parameter ε, as illustrated in Figure 6.

While Table 4 and Figure 7 fix the privacy parameter, we present the impact of a changing ε in Figure 8. In this plot, we fix the approximation parameter to $\lambda = 40$, display $DLap$ in blue and $\mathcal{N}_{\mathbb{Z}}$ in red, and test settings from a rather extremely flat case of $\varepsilon = 0.001$ up to an extremely concentrated setting of $\varepsilon = 10$. Again we use the setting of $\sigma^2 = \frac{1}{\varepsilon^2}$ and $p_{DLap} = e^{-\varepsilon}$.

Limits of our Evaluation The strength of our construction is clearly outperforming all related works in the settings it supports. However, we are limited in the approximation error given a specific target distribution and table parameters. Setting, e.g., $\varepsilon = 0.001$, the closest approximation reached by the grid search in Figure 6 satisfies only an error bound of 2^{-55}. Given a more restrictive setting (i.e., where higher values of λ or lower values of ε are required), our construction cannot be used as is. Finally, our implementation and analysis assume a 3-party RSS as the underlying secret sharing scheme. In the following, we discuss the potential impact of applying our contributions in different party settings.

Analytic discussion of LUT sampling strategies Our quantitative evaluation shows that lookup table-based sampling is promising across the board, as demonstrated by Franzese et al., who excel in the setting of many parties, Kii et al., who are tailored to two parties, and our work, which is the most efficient in the three-party setting. In the following, we compare the individual approaches on a qualitative level. We begin by evaluating related LUT sampling in our setting and then proceed to estimate the cost of our method in the settings presented in the related work.

In Kii et al., N table evaluations are performed, and each table needs no more than $(1/\delta)^{1/N}$ elements, for sensitivity-1 queries. Transferred to our notation, we have $2^k \leq (1/2^{-\lambda})^{1/N} = 2^{\lambda/N}$ and consequently $k \leq \lambda/N$. The target distribution $DDLap$ exhibits a sub-optimal privacy-utility tradeoff that scales negatively with the increasing number of tables. Therefore, they limit the number of tables to $N = \{2, 3\}$, resulting in $k \leq \{20, 14\}$. Following the communication cost in Table 5, the evaluation of these tables would incur 1554 bits for $N = 2$ and 585 bits for $N = 3$, using our implementation. This roughly equals the cost of evaluating a single table with $k = 18$, where, according to Table 6, our method approximates the Gaussian and Laplace distributions with $\lambda = 40$ for privacy parameters $\varepsilon \geq 2^{-6}$ and $\varepsilon \geq 2^{-4}$, respectively.

For Franzese et al., we compute the necessary table size with the script provided by the authors[15] and compare to our method as in Figure 6. Computing the table sizes for $\sigma = \{2^0, 2^2, 2^4, 2^6, 2^8, 2^{10}\}$, and $\lambda = 40$, DE-Sample requires a λ tables of sizes $2^{\sigma+6}$ while our work uses a single table of size $2^{\sigma+12}$. Following again the communication costs of Table 5, our method will be significantly more efficient for all these parameters in our setting due to the linear cost impact of λ. In addition, the iterative evaluation of LUTs will require the evaluation of (partially) secret shared lookup tables. As a consequence, LUT evaluation is more expensive, and adding additional dimensions will yield fewer performance benefits.

This strongly indicates that our proposed method performs best in our MPC setting when employing the discrete Gaussian or Laplace distributions. Even if $DDLap$ is acceptable in some

[15] https://github.com/cleverhans-lab/Secure_Noise_Sampling_DP_CL/blob/main/dice_ensemble_code/dice_ensemble_approximation.py

given use case, for the ε parameter regimes reported in Kii et al., our method performs better. While performance in the popular 3-party honest majority setting serves as a relevant baseline, we now provide some intuitions on the expected suitability of our method in other settings.

In the two-party setting, we expect similar performance to Kii et al. using their instantiation of $N = \{2, 3\}$ for $k = 20$ and $k = 14$, respectively. Our method approximates the Laplace and Gaussian distributions well for privacy parameters $\varepsilon \geq 2^{-6}$ and $\varepsilon \geq 2^{0}$, respectively. The additional communication complexity of index sampling with its $(c-1)l$ bit multiplications is insignificant compared to the total communication cost[16] as the cost of an individual bit multiplication amortizes to below one bit, using recent results of Silent-OT [4,5,14]. Apart from directly applying the techniques of Kii et al. in the two-party setting, ABY2 [38] offers a sharing strategy that enables dot product evaluation with cost independent of vector size. While these capabilities potentially enable our total communication trade-offs, we leave a concrete evaluation for future work.

Applying our method in Mixed-Mode MPC [23], we first treat the cost of index sampling. With our maximum setting of $c = 12$ and $l = k$, we get at most $11k$ ciphertext-ciphertext multiplications among the parties. Letting P be the number of parties, the one-hot vector generation incurs $2^{k-\log P}$ such multiplications. We conclude that the total communication cost will again be dominated by the cost of vector generation and table evaluation. As outlined above, DE-Sample evaluates many smaller tables, whereas our method evaluates a single large table. We use the values for σ and the resulting table sizes as above. Without adaptation and within this parameter range, it seems that the exponential scaling of both the one-hot vector generation and the table evaluation will lead to worse performance for our method as $2^{\sigma+12} = 64 \cdot 2^{\sigma+6} > 40 \cdot 2^{\sigma+6}$. However, this does not take into account that while for computation this is true, for communication it is not. We provide performance numbers for the implementation of Franzese et al. in our benchmark environment, with different table dimensions, in Appendix C. We also present runtime and network performance for evaluating the respective tables for DE-Sample and our method in Figures 12 and 13. Our small-scale experiments indicate the following. First, for more concentrated distributions with $\sigma \leq 2^{8}$, our method appears to improve communication and is at least competitive in runtime. Second, the main drawback of our method is that, in addition to the privacy parameter, the security parameter also impacts the table size. Thus, DE-Sample runtime scales better for higher levels of ε and can achieve higher values of λ.

6 Conclusions and Open Directions

We present a new construction for multiparty noise sampling using LUTs and achieve substantial performance improvements over the state of the art, in the setting that we consider. The results presented however also 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 wide range of parameters and achieve negligible approximation errors. For such distributions, our protocol constitutes a substantial efficiency improvement over previous work, especially in terms of communication. All previous constructions have communication complexity roughly in the megabyte range, while our construction has an amortized cost of well under a kilobyte. Also, the efficiency of the protocol is independent of the target noise distribution (and thus privacy parameters), as long as the

[16] Note that in the related work, N tables are evaluated, giving us a significant margin in this estimation.

distribution can be approximated sufficiently well under the fixed choice of LUT size and index distribution.

Further, our implementation relies solely 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, although we leave doing so for future work.

Another promising avenue for future work is combining our techniques with those of Franzese et al.[20] or Kii et al.[32] and it is plausible that doing so may result in increased efficiency, especially in wider MPC settings and/or for other target distributions.

Acknowledgements

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. Finally, we extend our gratitude to the authors of [32] and [20] for taking their time to clarify questions regarding their publications.

References

1. Albers, S., Khan, A., Ladewig, L.: Best fit bin packing with random order revisited. Algorithmica **83**(9), 2833–2858 (2021). https://doi.org/10.1007/S00453-021-00844-5
2. Beimel, A., Nissim, K., Omri, E.: Distributed private data analysis: Simultaneously solving how and what. In: CRYPTO 2008. LNCS, vol. 5157, pp. 451–468. Springer (2008). https://doi.org/10.1007/978-3-540-85174-5_25
3. 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
4. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: Conference on Computer and Communications Security, CCS. pp. 291–308. ACM (2019). https://doi.org/10.1145/3319535.3354255
5. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Advances in Cryptology - CRYPTO. LNCS, vol. 11694, pp. 489–518. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_16
6. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS. ACM (2016), https://doi.org/10.1145/2976749.2978429
7. Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., Yalame, H.: FLUTE: fast and secure lookup table evaluations. In: S&P (2023), https://doi.org/10.1109/SP46215.2023.10179345
8. Cacchiani, V., Iori, M., Locatelli, A., Martello, S.: Knapsack problems - an overview of recent advances. part II: multiple, multidimensional, and quadratic knapsack problems. Comput. Oper. Res. **143**, 105693 (2022). https://doi.org/10.1016/J.COR.2021.105693, https://doi.org/10.1016/j.cor.2021.105693
9. Canetti, R.: Universally composable security. J. ACM **67**(5) (Sep 2020). https://doi.org/10.1145/3402457
10. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: CRYPTO 2015 (2015), https://doi.org/10.1007/978-3-662-48000-7_1
11. 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

12. 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
13. Chekuri, C., Khanna, S.: A polynomial time approximation scheme for the multiple knapsack problem. SIAM J. Comput. **35**(3), 713–728 (2005). https://doi.org/10.1137/S0097539700382820
14. Couteau, G., Rindal, P., Raghuraman, S.: Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In: Advances in Cryptology - CRYPTO. LNCS, vol. 12827, pp. 502–534. Springer (2021). https://doi.org/10.1007/978-3-030-84252-9_17
15. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: TCC (2005), https://doi.org/10.1007/978-3-540-30576-7_19
16. Dalskov, A.P.K., Escudero, D., Nof, A.: Fast fully secure multi-party computation over any ring with two-thirds honest majority. In: CCS 2022. pp. 653–666. ACM (2022). https://doi.org/10.1145/3548606.3559389
17. 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), https://eprint.iacr.org/2018/486.pdf
18. 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
19. 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
20. 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
21. Fu, Y., Wang, T.: Benchmarking secure sampling protocols for differential privacy. In: CCS '24 (2024). https://doi.org/10.1145/3658644.3690257
22. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: EUROCRYPT 2017. LNCS, vol. 10211, pp. 225–255 (2017). https://doi.org/10.1007/978-3-319-56614-6_8
23. Garg, R., Yang, K., Katz, J., Wang, X.: Scalable mixed-mode MPC. In: IEEE Symposium on Security and Privacy. pp. 523–541. IEEE (2024). https://doi.org/10.1109/SP54263.2024.00106
24. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. SIAM Journal on Computing (2012). https://doi.org/10.1137/09076828X
25. Heath, D., Kolesnikov, V., Ng, L.K.L.: Garbled circuit lookup tables with logarithmic number of ciphertexts. In: EUROCRYPT (2024), https://doi.org/10.1007/978-3-031-58740-5_7
26. Inusah, S., Kozubowski, T.J.: A discrete analogue of the laplace distribution. Journal of Statistical Planning and Inference (2006), https://doi.org/10.1016/j.jspi.2004.08.014
27. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: TCC (2013), https://doi.org/10.1007/978-3-642-36594-2_34
28. 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, https://doi.org/10.1002/ecjc.4430720906
29. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J. Comput. **3**(4), 299–325 (Dec 1974). https://doi.org/10.1137/0203025
30. 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
31. 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: Applied Cryptography and Network Security. pp. 229–249. Springer, Cham (2017)

32. Kii, M., Ichikawa, A., Miura, T.: Lightweight two-party secure sampling protocol for differential privacy. Proceedings on Privacy Enhancing Technologies **2025**, 23–36 (2025), https://doi.org/10.56553/popets-2025-0003
33. Meisingseth, F., Rechberger, C., Schmid, F.: Practical two-party computational differential privacy with active security. PoPETs (2025). https://doi.org/c, https://doi.org/https://doi.org/
34. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: CRYPTO 2009 (2009), https://doi.org/10.1007/978-3-642-03356-8_8
35. 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
36. Morita, H., Pohle, E., Sadakane, K., Scholl, P., Tozawa, K., Tschudi, D.: MAESTRO: multi-party AES using lookup tables. In: 34th USENIX Security Symposium (USENIX Security 25). USENIX Association (aug 2025), https://www.usenix.org/conference/usenixsecurity25/presentation/morita
37. Mullen, G.L., Panario, D.: Introduction to finite fields : Basic properties of finite fields. In: Handbook of Finite Fields, pp. 13–31. Discrete mathematics and its applications, CRC Press (2013)
38. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: improved mixed-protocol secure two-party computation. In: USENIX. pp. 2165–2182 (2021)
39. Shamir, A.: How to share a secret. Commun. ACM (1979). https://doi.org/10.1145/359168.359176
40. Wagh, S.: Pika: Secure computation using function secret sharing over rings. PoPETs **2022**(4), 351–377 (2022), https://doi.org/10.56553/popets-2022-0113
41. 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, 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 $\mathcal{F}_{\texttt{rand}}, \mathcal{F}_{\texttt{verify}}$, $\mathcal{F}_{\texttt{weakDotProduct}}/\mathcal{F}_{\texttt{weakMult}}, \mathcal{F}_{\texttt{RandOHV}}, \mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ and for all of them we use their formulations as in MAESTRO [36] so we refer the reader there for formal formulations and more detailed discussions. Informally, the functionalities may be understood as follows:

- $\mathcal{F}_{\texttt{rand}}$ samples a uniformly random value in a finite field and distributes shares of this value to all parties.
- $\mathcal{F}_{\texttt{weakDotProduct}}/\mathcal{F}_{\texttt{weakMult}}$ 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 $\mathcal{F}_{\texttt{weakMult}}$.
- $\mathcal{F}_{\texttt{verify}}$ 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.
- $\mathcal{F}_{\texttt{RandOHV}}$ 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.
- $\mathcal{F}_{\texttt{LUT}}^{[\![\cdot]\!]}$ 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.

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 as in MAESTRO [36] so we refer the reader there for details. The security of the protocols below rely solely on two elementary facts; Firstly, the adversary/environment 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, are correct and introduce no opportunity for cheating except for aborting.

B.1 Bernoulli Index Sampling

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: $([\![r_{l-1}^{j'}]\!], \ldots, [\![r_0^{j'}]\!]) \leftarrow \mathcal{F}_{\texttt{rand}}$ 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 $j \in \{l, \ldots k-1\}$.

3 : Verify all the multiplication triples via $\mathcal{F}_{\texttt{verify}}$.

4 : **return** $[\![i]\!]$

Fig. 9: The index sampling protocol π_I, where the bits of the index are either i.i.d. according to $Ber(2^{-c})$ or $Ber(1/2)$.

The protocol for Bernoulli index sampling is found in Figure 9. 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 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 9) securely realizes functionality $\mathcal{F}_{\texttt{Ber}}$ (Figure 3) 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.*

One-hot Vector Generation The protocol for sampling random one-hot vectors is given in 10. It is precisely the one from MAESTRO [36] 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 [31], where the correctness of the protocol and UC-security in the *arithmetic black-box model (ABB)* 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 ([36,31]). *Protocol $\pi_{\texttt{RndOhv}}$ (10) 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.2 Lookup Table Evaluation

The protocol for oblivious lookup table evaluations is given in Figure 11. It is the same as the protocol in MAESTRO [36] for two-dimensional LUT evaluations (their Protocol 10), except that we introduce one extra dimension, and that the dimensions may have different sizes. 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. It is similarly straight-forward to verify that the output is correct (i.e. is $L(i)$) given that the elementary operations are correct. Therefore, we omit the proof.

Lemma 3. *Protocol $\pi_{\texttt{LUT}}$ (Figure 11) 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.3 Security of Full Sampling Protocols

We now give the security proof of our sampling protocols.

Proof (Proof of 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 $\mathcal{F}_{\texttt{Ber}}(p_{Ber}, k, l)$ 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.

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 : $\quad M \leftarrow 2^j$

5 : $\quad$ **for** $j' = 0, \ldots, M/2 - 2$:

6 : $\quad\quad [\![e_{j'}]\!] \leftarrow \mathcal{F}_{\texttt{weakMult}}([\![r_j]\!], [\![v_{j'}]\!])$

7 : $\quad$ **endfor**

8 : $\quad [\![e_{M/2-1}]\!] \leftarrow \left(\bigoplus_{j'=0}^{M/2-2} [\![e_{j'}]\!]\right) \oplus [\![r_{j'}]\!]$

9 : $\quad [\![\mathbf{v}]\!] \leftarrow ([\![v_0]\!] \oplus [\![e_0]\!], \ldots, [\![v_{\frac{M}{2}-1}]\!] \oplus [\![e_{\frac{M}{2}-1}]\!], [\![e_0]\!], \ldots, [\![e_{\frac{M}{2}-1}]\!])$

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. 10: The protocol $\pi_{\texttt{RndOhv}}$ (from [36]) for sampling random one-hot vectors.

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.

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

Parameters: LUT size $N = 2^k = 2^{k_1+k_2+k_3}$, LUT $\texttt{L} \in \mathbb{Z}_{2^w}^N$
Input: Shared index $[\![i]\!]$ with $i \in \mathbb{Z}_N$.
Output: $[\![s := \texttt{L}(i)]\!]$.

---

1 : Sample 3 random one-hot vectors:

$(\{[\![\mathbf{r}]\!]\}, e^{(r)}), ([\![\mathbf{s}]\!]\}, e^{(s)}), ([\![]\!], \mathbf{e}^{(\mathbf{t})})$,

each from $\mathcal{F}_{\texttt{RandOHV}}(2^{k_i})$.

2 : $[\![\hat{\mathbf{r}}]\!] \leftarrow ([\![r_0]\!], \ldots, [\![r_{2^{k_1}-1}]\!], [\![s_0]\!], \ldots, [\![s_{2^{k_2}-1}]\!],$

$[\![t_0]\!], \ldots, [\![t_{2^{k_3}-1}]\!])$

3 : $c \leftarrow \mathit{Reconstruct}([\![i]\!] + [\![\hat{\mathbf{r}}]\!])$.

4 : $[\![\mathbf{v}]\!] \leftarrow (\underbrace{[\![e_0^{(r)}]\!], \ldots, [\![e_0^{(r)}]\!]}_{\times 2^{k_2+k_3}}, \ldots, [\![e_{2^{k_1}-1}^{(r)}]\!], \ldots, [\![e_{2^{k_1}-1}^{(r)}]\!])$

with $\mathbf{v}$ being of length N.

5 : $[\![\mathbf{v}']\!] \leftarrow (\underbrace{[\![e_0^{(s)}]\!], \ldots, [\![e_0^{(s)}]\!]}_{\times 2^{k_3}}, \ldots, [\![e_{2^{k_2}-1}^{(s)}]\!], \ldots, [\![e_{2^{k_2}-1}^{(s)}]\!])$

6 : $[\![\mathbf{v}'']\!] \leftarrow ([\![e_0^{(t)}]\!], \ldots, [\![e_{2^{k_3}-1}^{(t)}]\!], \ldots, [\![e_0^{(t)}]\!], \ldots, [\![e_{2^{k_3}-1}^{(t)}]\!])$

7 : $[\![\mathbf{w}]\!] \leftarrow (\texttt{L}_c \cdot [\![v_0]\!], \ldots, \texttt{L}_{c \oplus N-1} \cdot [\![v_{N-1}]\!])$

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. 11: MPC lookup protocol with $d = 3$. The protocol is an extension of Protocol 10 in MAESTRO[36], the only differences being that we have $d = 3$ (they have $d = 2$) and that we have dimensions of different sizes.

- 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 Results Continued

In addition to the results presented in Section 5.2, we provide further data in this section.

Costs of different dimension settings Our proposed construction uses multiple dimensions to reduce total communication. In table 5, we provide the optimal setup of dimensions for different sizes of the total table 2^k and the word size in the table w. In our experiments, we exclusively use $w = 8$ and $w = 16$. However, we have two slight deviations. First, the packing implementation requires $w \cdot 2^{k_3} \geq 64$, constraining $k_3 \geq 3$ and $k_3 \geq 2$ for $w = 8$ and $w = 16$, respectively. Second, we implemented only lookups with $d = \{2, 3\}$. Only a single entry is affected. For $k = 24$, we decompose it into $k_1 = k_2 = 9$ and $k_3 = 6$, incurring 1581 bits of total communication instead of the optimal 1566.

Microbenchmarks of DE-Sample In Section 5.2, we give an estimation of how our method may perform in different MPC settings. In particular, when comparing our work with that of Franzese et al. [20], we deviate from one of their arguments. Increasing the size of the table has an exponential impact on runtime but only a small linear impact on communication. As a result, the authors fix the table size to 2^{16} to accommodate all of their experiments. One of the main distinctions between our works is that we have a single large table, and they have many smaller tables. In Figures 12 and 13, we thus provide additional benchmarks for their implementation for different table sizes. In particular, we use the provided script to compute the minimum table DE-Sample and evaluate the resulting cost, as shown in blue. In red, we give the runtime and network costs of evaluating a table of size depicted in Figure 6.

D Details on Upper Bounding δ_{appr}

Generally when we compute the statistical distance between the target, Z, and the output of our protocol, Out_{π_Z}, we have one computed distance δ^{calc}_{appr} and two associated sources of numerical errors. The first is in our computation of f_Z for each z in the range and the second is for our calculation of the pmf of our output distribution. These two numerical errors combine into the full error of our calculation. Then there additionally is an error arising from truncating the target distribution, denoted δ_{trunc}. We perform our computations using the *fastnum* library[17], guaranteeing us that additions, subtractions, multiplications and divisions are performed with full n-bit precision for some n of our choosing. This means that such operations cause an additive error of at most 2^{-n}.

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

Table 5: Communication optimal setting with at most 4 dimensions for a given total LUT size of 2^k. Numbers in parentheses give individual dimensions in ascending order, the number below indicates communication in bits, and w indicates word size in the lookup table.

k	[36]	$w = k$	$w = 1$	$w = 8$	$w = 16$
12	$(6, 6)$ $114 + w$	$(5, 5, 2)$ 113	$(4, 4, 2, 2)$ 45	$(5, 5, 2)$ 93	$(6, 6)$ 130
13	$(6, 7)$ $177 + w$	$(5, 6, 2)$ 149	$(4, 5, 2, 2)$ 60	$(5, 6, 2)$ 124	$(6, 6, 1)$ 162
14	$(7, 7)$ $240 + w$	$(6, 6, 2)$ 185	$(5, 5, 2, 2)$ 75	$(6, 6, 2)$ 155	$(6, 6, 2)$ 195
15	$(7, 8)$ $367 + w$	$(6, 6, 3)$ 253	$(5, 5, 3, 2)$ 94	$(6, 6, 3)$ 190	$(6, 7, 2)$ 258
16	$(8, 8)$ $494 + w$	$(7, 7, 2)$ 321	$(5, 6, 3, 2)$ 125	$(6, 7, 3)$ 253	$(7, 7, 2)$ 321
17	$(8, 9)$ $749 + w$	$(7, 7, 3)$ 397	$(6, 6, 3, 2)$ 156	$(7, 7, 3)$ 316	$(7, 7, 3)$ 388
18	$(9, 9)$ $1004 + w$	$(7, 8, 3)$ 533	$(6, 6, 3, 3)$ 195	$(7, 7, 4)$ 387	$(7, 8, 3)$ 515
19	$(9, 10)$ $1515 + w$	$(8, 8, 3)$ 669	$(6, 7, 3, 3)$ 258	$(7, 8, 4)$ 514	$(8, 8, 3)$ 642
20	$(10, 10)$ $2026 + w$	$(8, 8, 4)$ 845	$(7, 7, 3, 3)$ 321	$(8, 8, 4)$ 641	$(8, 8, 4)$ 777
21	$(10, 11)$ $3049 + w$	$(8, 9, 4)$ 1117	$(7, 7, 4, 3)$ 392	$(8, 8, 5)$ 784	$(8, 9, 4)$ 1032
22	$(11, 11)$ $4072 + w$	$(9, 9, 4)$ 1389	$(7, 8, 4, 3)$ 519	$(8, 9, 5)$ 1039	$(9, 9, 4)$ 1287
23	$(11, 12)$ $6119 + w$	$(9, 9, 5)$ 1789	$(8, 8, 4, 3)$ 646	$(9, 9, 5)$ 1294	$(9, 9, 5)$ 1558
24	$(12, 12)$ $8166 + w$	$(9, 10, 5)$ 2333	$(8, 8, 4, 4)$ 789	$(9, 9, 5, 1)$ 1566	$(9, 10, 5)$ 2069

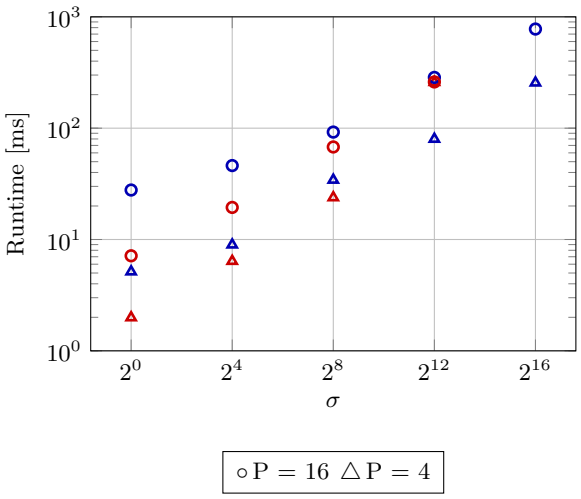

Fig. 12: Hypothetical runtime performance of our method, when employed in Mixed Mode MPC. Our strategy is depicted in red, while [20] is depicted in blue.

D.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

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

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, for each z, the numerical error in computing $f_Z(z)$ is upper bounded by $(2B+1)\binom{B}{\lceil B/2 \rceil}(B-1)2^{-n}$. Now we want an upper bound on $\binom{B}{\lceil B/2 \rceil}$

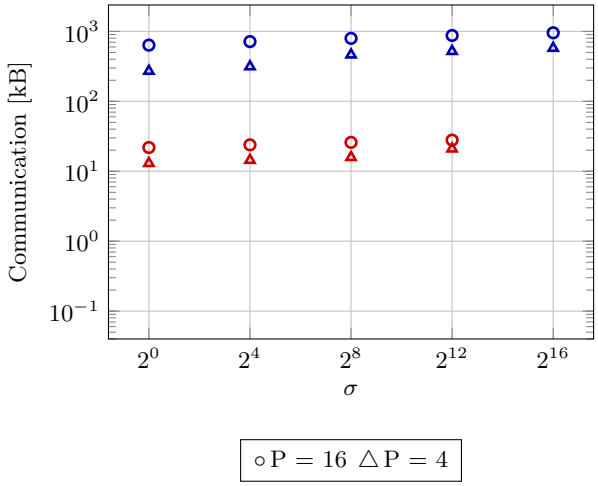

Fig. 13: Hypothetical communication cost of our method, when employed in Mixed Mode MPC. Our strategy is depicted in red, while [20] is depicted in blue.

in terms of negative powers of 2. For smaller values of B, we can simply compute an explicit upper bound but for larger B, this becomes troublesome due to the sizes of the numbers involved. We may instead use the general bound $\binom{a}{b} < (\frac{a \cdot e}{b})^b$, with e being Euler's number. Assuming that B is even, we thus get $\binom{B}{\lceil B/2 \rceil} < (\frac{B \cdot e}{B/2})^{B/2} = (2 \cdot e)^{B/2} < (2 \cdot 4)^{B/2} = (2^3)^{B/2} = 2^{3B/2}$. Whereas this is a non-tight upper bound, we cannot hope to get a much smaller value, since the general lower bound $\binom{a}{b} \geq (\frac{a}{b})^b$ gives $\binom{B}{\lceil B/2 \rceil} \geq 2^{B/2}$. This means that we, roughly, need to have $n \propto B$, which in our implementation becomes computationally infeasible for n larger than a few thousand. Luckily, most of the parameter values that we consider do not require such large B.

For instance, when we have $B \leq 255$ ($w \leq 8$), we get $\binom{B}{\lceil B/2 \rceil} \leq \binom{255}{127} < 2^{251}, (2B+1) \leq 511 < 2^9, B - 1 \leq 254 < 2^8$, so the numerical error is at most $2^9 \cdot 2^{251} \cdot 2^8 \cdot 2^{-n} = 2^{-n+268}$. This allows us to use $n = 512$. The largest B we use for the Laplace distribution is $2^{12} = 4096$ and then we get (by the upper bound above) $\binom{B}{\lceil B/2 \rceil} < (\frac{2^{12}}{2^{11}})^{3 \cdot 2^{11}/2} = 2^{3 \cdot 1024}$ and $(2B+1) < 2^{15}, B - 1 < 2^{12}$. This results in a numerical error no more than $2^{-n+3 \cdot 1024+15+12} = 2^{-n+3099}$ so using $n = 8192$ will be more than sufficient in this case. For $n = 8192$, our computations still run only for a few minutes. Our current implementation does not support a larger value of n but we do not see this as an inherent limit.

For δ^{calc}_{trunc}, we use the bound $F_{DLap(p)}(-B-1) = \frac{p^{-B-1}}{1+p} < p^{-B-1}$ from [26]. 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 $\delta^{calc}_{trunc} = 2(\Delta^{-1}\tilde{p})^{B+1}$ and have $\delta^{err}_{trunc} \leq 2\binom{B+1}{\lceil (B+1)/2 \rfloor} B 2^{-n}$, which for $B \leq 256$ is no more than $2^{-n+256+8+1} = 2^{-n+265}$ and for $B \leq 2^{12}$ no more than $2^{-n+3 \cdot 1024+14} = 2^{-n+3086}$.

We get the probability masses for each z by summing up the index probabilities for the elements in the LUT which are assigned to z. If we use K uniformly random bits in the generation of indices, then each index will have a probability which is a multiple of 2^{-K}. Thus, as long as $K \leq n$, each of these probabilities will be perfectly represented by value with n bits of precision. In our

Table 6: Benchmarks of the work of [20] providing context on the impact of k on the average runtime in [ms] and the average communication cost in kB.

k	P = 3		P = 4		P = 16	
	Time	Comm.	Time	Comm.	Time	Comm.
6	0.12	6.26	0.12	6.76	0.69	15.88
7	0.16	6.79	0.16	7.30	0.82	16.88
8	0.22	7.31	0.22	7.84	1.15	17.85
9	0.58	10.06	0.44	10.89	1.56	18.85
10	0.84	10.71	0.85	11.57	2.30	19.86
11	1.22	11.38	1.24	12.28	3.66	20.86
12	1.98	12.05	1.99	12.99	7.14	21.83
13	3.35	12.73	3.50	13.69	11.27	22.84
14	6.15	13.40	6.39	14.40	19.42	23.84
15	11.78	14.07	12.21	15.11	35.67	24.85
16	22.97	14.74	23.82	15.82	67.80	25.85
17	97.69	18.56	74.65	20.06	131.97	26.86
18	250.95	19.38	258.54	20.94	260.16	27.86

implementation, each index bit is generated by multiplying c uniformly random bits, and we have k index bits. Thus we get $K \leq c \cdot k$, and we have at most $k = 24, c = 12$, making $K \leq 288$ and we use $n \geq 512$, so there is no numerical error in the computation of our output distribution.

Thus the numerical error in computing the statistical distance is purely that of summing up the errors for our computations of $f_Z(z)$, plus the errors arising from truncation. That is, when $B \leq 255$,

$$\begin{aligned}\delta_{appr} &= \delta_{appr}^{calc} + (2B+1) \cdot 2^{-n+268} + \delta_{trunc}^{calc} + 2^{-n+265} \\ &\leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-n+268+1+9} + 2^{-n+265} \\ &\leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-n+279}.\end{aligned}$$

When we set $n = 512$, we get in the end

$$\delta_{appr} \leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-233}.$$

For $255 < B \leq 2^{12}$, we use $n = 8192$ and similarly get

$$\begin{aligned}\delta_{appr} &= \delta_{appr}^{calc} + (2B+1) \cdot 2^{-n+3099} + \delta_{trunc}^{calc} + 2^{-n+3086} \\ &\leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-n+3099+14} + 2^{-n+3086} \\ &\leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-n+3114} \\ &\leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-5078}.\end{aligned}$$

Consequently, we report on n as at least being 512 in the main body and similarly report that, for the discrete Laplace distribution,

$$\delta_{appr} \leq \delta_{appr}^{calc} + \delta_{trunc}^{calc} + 2^{-233}.$$

D.2 Bounding $\delta^{err}_{appr}, \delta^{err}_{trunc}$ for the Gaussian

The discrete Gaussian's pmf 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 high 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 has n bits of precision in multiplications and divisions , 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 $2^{-n+1}/(d_{tot}+(2B+1)2^{-n+1})+2^{-n} \leq 2^{-n+1}/(1+(2B+1)2^{-n})+2^{-n+1} \leq 2^{-n+1}+2^{-n+1} = 2^{-n+2}$. Thus the numerical error in computing $f_Z(z)$ is at most $(2B+1)\cdot 2^{-n+2}$ for each z.

For δ^{calc}_{trunc}, we use Proposition 25 in [11] 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 $\delta^{err}_{trunc} \leq 2^{-n+2}$.

As before with the discrete Laplace distribution, we have no numerical error in computing Out_{π_Z}. Therefore we get, with $B \leq 2^{16} = 65536$,

$$\begin{aligned}\delta^{err}_{appr} &\leq (2B+1)\cdot((2B+1)\cdot 2^{-n+2} + 2^{-n}) \\ &\leq (2^{17}+1)\cdot(2^{17}+2)\cdot 2^{-n+3} \leq 2^{18}\cdot 2^{18}\cdot 2^{-n+3} \\ &= 2^{-n+39}.\end{aligned}$$

Finally, we get, with $n = 512$,

$$\begin{aligned}\delta_{appr} &\leq \delta^{calc}_{appr} + 2^{-n+39} + \delta^{calc}_{trunc} + 2^{-n+2} \\ &\leq \delta^{calc}_{appr} + \delta^{calc}_{trunc} + 2^{-472}.\end{aligned}$$

Remark 1. The reason that the bound on numerical errors is so much better for the discrete Gaussian is that the distribution parameter (σ or p) is in the exponent rather than the base, as for the discrete Laplace distribution. This leads to larger B (and thus w) being available for the discrete Gaussian, which perhaps is counter-intuitive since it has lighter tails than the Laplace distribution. We note that this outcome is seemingly due to the method with which we compute the respective pmfs (having the parameter in the base seems to require a more error-prone numerical computation), and not due to how heavy the tails of the distributions are. If one requires even less concentrated Laplace distributions (necessitating a larger B) or larger λ than allowed by the

current analysis, one can in principle increase the bit-precision n, although this comes at a cost in computation time. Alternatively, one can seek even more effective ways of computing the pmf of the discrete Laplace distribution (leading to a smaller numerical error) or perform an analysis which exploits the way in which the package for fixed-number arithmetics represents numbers. We do not investigate these possibilities further since we, for our purposes, get satisfying results with the current method.