Private Sampling with Identifiable Cheaters

César Sabater
INRIA Lille
Villeneuve d'Ascq, France
cesar.sabater@inria.fr

Florian Hahn
University of Twente
Enschede, Netherlands
f.w.hahn@utwente.nl

Andreas Peter
University of Oldenburg
Oldenburg, Germany
andreas.peter@uni-oldenburg.de

Jan Ramon
INRIA Lille
Villeneuve d'Ascq, France
jan.ramon@inria.fr

ABSTRACT

In this paper we study verifiable sampling from probability distributions in the context of multi-party computation. This has various applications in randomized algorithms performed collaboratively by parties not trusting each other. One example is differentially private machine learning where noise should be drawn, typically from a Laplace or Gaussian distribution, and it is desirable that no party can bias this process. In particular, we propose algorithms to draw random numbers from uniform, Laplace, Gaussian and arbitrary probability distributions, and to verify honest execution of the protocols through zero-knowledge proofs. We propose protocols that result in one party knowing the drawn number and protocols that deliver the drawn random number as a shared secret.

KEYWORDS

differential privacy, sampling, zero knowledge proofs, multiparty computation

1 INTRODUCTION

Nowadays, randomization is an important algorithmic technique. Its numerous applications include randomized algorithms, e.g., for many problems the simplest or most efficient known solution strategy is a randomized algorithm, and hiding information, e.g., in cryptography or in differential privacy. While true randomness is hard to achieve in most cases it is sufficient to be able to generate pseudo-random numbers. A wide range of approaches exist to generate pseudo-random numbers of good quality.

The situation becomes more complicated when we consider generating random numbers in the context of multi-party computation between parties which do not trust each other. We are particularly interested in algorithms which allow multiple parties to draw a random number from a specified probability distribution in such a way that all parties can be convinced that the number drawn is truly random and that either all parties, only one party, or none of the parties learn the drawn random number. This implies that no party should be able to influence the probability distribution or be able to predict or guess the random number.

Proceedings on Privacy Enhancing Technologies 2023(2), 361–383

https://doi.org/10.56553/popets-2023-0058

Such algorithms are particularly useful for differentially private federated machine learning using sensitive data from multiple data owners. In this setting, one would like to learn a statistical model $\mathcal{M}$ with parameters θ on the sensitive data of multiple data owners. Such model could reveal sensitive information and therefore one possible technique is to perturb the model before publication sufficiently such that it becomes differentially private [27], i.e., such that from the perturbed model $\hat{\mathcal{M}}$ with parameters $\hat{\theta}$ one cannot distinguish a change in a single individual. This can be achieved by drawing some noise η from an appropriate probability distribution, e.g., η often is a vector of Laplace or Gaussian random variables, and setting $\hat{\theta} = \theta + \eta$. In such scenario it is important nobody knows η as else that party could subtract η from the published $\hat{\theta}$ to obtain the sensitive model parameters θ. At the same time, all data owners want to be sure that η is drawn correctly: if anyone can bias the distribution of this noise, privacy may not be guaranteed anymore or the model parameters may be biased in a way similar as what one can achieve with data poisoning [49, 53].

In this paper we develop algorithms to verifiably draw random numbers. We consider uniform distributions, Laplace distributions, Gaussian distributions and arbitrary distributions. We develop strategies with three different privacy levels for the random number: strategies which verifiably draw a publicly known random number, strategies which verifiably draw a random number which is revealed to only one party and strategies which verifiably draw a random number and output it as a shared secret so that none of the parties knows the random number.

An important tool to prove correct behavior can be found in zero knowledge proofs (ZKP). These are cryptographic techniques that allow a party to prove statements without revealing anything else. Typically, one considers statements involving logical and arithmetic relations over private values which can be expressed using additions, multiplications and other elementary operations such as comparisons. For drawing from Laplace or Gaussian distributions, transcendental functions are needed. We work towards bridging this gap based on Cordic [52], a classic technique for computing such functions.

The main contributions of this paper can be summarized as follows: (1) we propose strategies to prove relationships involving logarithms or trigonometric functions in zero knowledge, (2) we propose and compare several strategies to let a party verifiably draw Gaussian random numbers, (3) we propose algorithms to let one party verifiably sample from the Laplace distribution and from an arbitrary distribution, (4) we propose algorithms to draw from

the Gaussian or Laplace distribution a random number represented as a shared secret.

The remainder of this paper is structured as follows:

After reviewing some common notations and concepts in Section 2, we formalize our problem statement in Section 3. Next, in Section 4 we discuss related work and in Section 5 we provide a high-level overview of our method. After that, in Section 6 we review the Cordic algorithm and adapt it for zero-knowledge proofs. In Sections 7 and 8 we apply these techniques for sampling from the Laplace and Gaussian distributions. To show how our methods work in practice, in Section 9 we provide an experimental comparison of the several possible strategies to sample from the Gaussian distribution. In Section 10 we discuss the application of our techniques to the problem of differentially private machine learning. Finally, in Section 11 we conclude and outline directions of future work.

2 PRELIMINARIES

We will denote the set of the first k positive integers by $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$. We denote the security parameter by λ. We say that a function is negligible in λ if, for each positive polynomial f, it is smaller than $\frac{1}{f(\lambda)}$ for sufficiently big λ. We omit λ sometimes in negligible functions when it is clear it refers to λ. $a \leftarrow_R S$ means that a is sampled uniformly at random from elements of S. For vectors $\mathbf{a} = (a_1, \ldots, a_k)$ and $\mathbf{b} = (b_1, \ldots, b_k)$, $\mathbf{a} + \mathbf{b}$ and $\mathbf{a} * \mathbf{b}$ are the element wise addition and product. $\mathbf{a}^{\mathbf{b}}$ is the multi-exponentiation $\prod_{i=1}^{k} a_i^{b_i}$. For a scalar s, $s + \mathbf{a} = (s, \ldots, s) + \mathbf{a}$, $s * \mathbf{a} = (s, \ldots, s) * \mathbf{a}$ and $\mathbf{a}^s = \mathbf{a}^{(s,\ldots,s)}$. The function $\text{sign}(x)$ is equal to 1 if $x \geq 0$ and to -1 otherwise.

2.1 Setting and Threat Model

We consider a set of n parties $\mathbf{P} = \{P_1, \ldots, P_n\}$. We assume parties have access to a public-key infrastructure which they can use to prove their identity when sending messages. Parties communicate through secure channels and have access to a public bulletin board that they can use to post messages. When a party sends a message to the bulletin board, it is forwarded to all other agents as when using a broadcast channel. In addition, all broadcasted messages remain publicly visible in the bulletin board, which allows to have publicly verifiable protocols.

We assume that a subset of parties $\mathbf{P}_{cor} \subset \mathbf{P}$ is corrupted and controlled by an adversary $\mathcal{A}$. $\mathcal{A}$ can make corrupted parties to deviate arbitrarily from the protocol and perform coordinated attacks. Our protocols are secure if at least one party is honest. The set $\mathbf{P}_{cor}$ of corrupted parties is assumed to be static, meaning that it does not change after the beginning of the execution.

We prove security in the simulation paradigm, using the model of *security with identifiable abort* [33] for the stand-alone setting [29]. This setting allows to obtain sequentially composable protocols in which parties are able to detect malicious actions and can in such cases abort the protocol. Deterrence measures may be in place to discourage parties from being detected as malicious. In fact, unless parties stop participating our protocols either complete successfully or abort with a proof that a specific party is a cheater, i.e., in case our protocols abort the message trace (which is kept on the bulletin board) allows for proving that a specific party did not follow the protocol. Assuming that adversaries will be deterred if they risk getting caught is a standard assumption that applies in many scenarios [3].

In parts of our protocols, we make use of specific Zero Knowledge Proofs that are non-interactive versions of compressed Σ-protocols whose security relies on the Random Oracle Model [6]. We provide a detailed description of our security framework and prove the security of our protocols in Appendix B.

2.2 Commitment Schemes

A *commitment scheme* allows for committing to values while keeping them hidden. We use the vector variant of the Pedersen commitment scheme [47]. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p exponential in λ in which the Discrete Logarithm Assumption (DLA) holds. The setup of the commitment scheme takes as input a string of length $O(1^\lambda)$ and outputs a vector $\mathbf{g} = (g_1, \ldots, g_k)$ of elements sampled at random from $\mathbb{G} \setminus \{1\}$. It is required that no pairwise discrete logarithm on the elements $g_1, \ldots, g_k$ is known, which can be guaranteed without a trusted party as the setup only requires public randomness. A commitment $P \in \mathbb{G}$ of a vector $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{Z}_p^k$ satisfies $P = \mathbf{g}^{\mathbf{x}}$. We say that $\mathbf{x}$ is an *opening* of P. The scheme is *binding* as no computationally bounded adversary can find two openings $\mathbf{x}$ and $\mathbf{x}'$ of P such that $\mathbf{x} \neq \mathbf{x}'$ except with probability negligible in λ. If $\mathbf{x} = (\hat{\mathbf{x}}, r)$, where $\hat{\mathbf{x}}$ is the data and r is sampled uniformly at random from $\mathbb{Z}_p$, P is uniformly distributed in $\mathbb{G}$ and therefore does not reveal any information about $\hat{\mathbf{x}}$. This is known as the *hiding* property. In our protocols, one coordinate of $\mathbf{g}$ is always reserved for randomness. The scheme is also *homomorphic* as, given commitments P and Q of $\mathbf{x}$ and $\mathbf{y}$ respectively, PQ is a commitment of $\mathbf{x} + \mathbf{y}$.

2.3 Arithmetic Circuits

An *arithmetic circuit* (or just *circuit*) $C : \mathbb{Z}_p^k \to \mathbb{Z}_p^s$ is a function that only contains additions and multiplications modulo p. In the following sections we will define circuits using the notation

$$C(a; i_1, \ldots, i_k) := \begin{bmatrix} o_1 \\ \vdots \\ o_s \end{bmatrix},$$

where $(i_1, \ldots, i_k)$ is the input, $(o_1, \ldots, o_s)$ is the output, and a are constants that may change the circuit structure, for example, in the case we are defining a family of similar circuits.

2.4 Compressed Σ-Protocols

We will prove statements about private committed values using Zero Knowledge Proofs [31]. In such proofs, for a NP relation $\mathcal{R}$, a *prover* $\mathcal{P}$ interacts with a *verifier* $\mathcal{V}$ to prove, for a public *statement* a, the knowledge of a private *witness* w such that $(a; w) \in \mathcal{R}$. At the end of the interaction, $\mathcal{V}$ either *accepts* or *rejects* the proof. ZKPs are (1) *complete*, as $\mathcal{V}$ always accepts a proof of an honest $\mathcal{P}$, (2) *sound*, as a proof of a dishonest $\mathcal{P}$ is rejected except with negligible probability and (3) *zero knowledge*, as no information other than $(a; w) \in \mathcal{R}$ is revealed by the protocol. The ZKPs that we use are also called *zero knowledge arguments*, as they are sound if $\mathcal{P}$ is computationally bounded. Additionally, they rely on the DLA.

The ZKPs we use are called compressed Σ-protocols [2]. Particularly, we use Protocol Π_{cs} of [2], that proves the nullity of the output of arithmetic circuits in $\mathbb{Z}_p$ applied to private inputs. Let $\mathbb{G}, \mathbb{Z}_p$, and $\mathbf{g}$ be as defined for commitments, then for any circuit $C : \mathbb{Z}_p^k \to \mathbb{Z}_p^s$, by applying Π_{cs} to C we obtain a complete, sound and zero knowledge proof for the relation

$$\{(P \in \mathbb{G}; \mathbf{x} \in \mathbb{Z}_p^k) : P \text{ is a commitment of } \mathbf{x} \wedge C(\mathbf{x}) = 0)\}.$$

While Π_{cs} is an interactive protocol between $\mathcal{P}$ and $\mathcal{V}$, it can be turned into a non-interactive proof using the Strong Fiat-Shamir heuristic [8]. By this transformation, ZKPs can be generated offline by $\mathcal{P}$ and later be verified by any party.

Let m be the number of multiplication gates of C, then the proof generated by the execution of Π_{cs} has a size of $2\lceil\log(k+2m+4)\rceil - 1$ elements of $\mathbb{G}$ and 6 elements of $\mathbb{Z}_p$. To generate such proof, the dominant computations are modular exponentiations in $\mathbb{G}$ (GEX). $\mathcal{P}$ performs $5k + 8m + 2\lceil\log_2(k+2m+4)\rceil + 6$ GEX, and the verification cost is of $k + 2m + 2\lceil\log_2(k + 2m + 4)\rceil - 1$ GEX.

We provide a detailed explanation of compressed Σ-protocols, their cost and some optimizations in Appendix A.

2.5 Secret Sharing

Consider again n parties $\{P_i\}_{i=1}^n$. For a positive prime p, group $\mathbb{Z}_p$, and a number $a \in \mathbb{Z}_p$, one can generate an additive secret share for a by drawing a random vector $(a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ subject to the constraint that $\sum_{i=1}^n a_i = a \bmod p$. We then denote this sharing of a as $[\![a]\!] = (a_1 \ldots a_n)$. The process of computing and revealing a from the sharing $[\![a]\!]$ is called opening the sharing $[\![a]\!]$. If every party P_i only receives a_i (for $i \in [n]$), then if not all parties collude each party can only see at most $n-1$ uniformly randomly distributed numbers, and hence has no information about the value of a.

If for one or more values a sharing is available, it is possible to perform various operations on them without revealing any new information, see [24] for an overview. If $[\![a]\!] = (a_1 \ldots a_n)$ is a sharing of a and $[\![b]\!]$ is a sharing of b, then $[\![a+b]\!] = (a_1+b_1 \ldots a_n+b_n)$ is a sharing of $a+b$. Given a sharing $[\![a]\!]$ of a and a public constant c, then $[\![ca]\!] = (ca_1 \ldots ca_n)$ is a sharing of ca. For multiplying two sharings, one can use pre-computed triples of sharings $([\![x]\!], [\![y]\!], [\![z]\!])$ with x and y random and $xy = z$. Given such triple, and two sharings $[\![a]\!]$ and $[\![b]\!]$ which one wants to multiply, one can compute $[\![d]\!] = [\![a]\!] - [\![x]\!]$ and $[\![e]\!] = [\![b]\!] - [\![y]\!]$ and open both $[\![d]\!]$ and $[\![e]\!]$. Then, a sharing of $c = ab$ is obtained by $[\![c]\!] = [\![z]\!] + d[\![y]\!] + e[\![x]\!] + de$. Several approaches have been proposed to generate such triples of sharings $([\![a]\!], [\![b]\!], [\![c]\!])$ efficiently, typically involving a somewhat homomorphic encryption (SHE) scheme with distributed decryption, where the parties can generate random sharings $[\![a]\!]$ and $[\![b]\!]$ uniformly at random, encrypt them, multiply them and decrypt the product in a distributed way to obtain $[\![c]\!]$ [24].

We'll adopt a number of ideas from [23]. In particular, we will represent sharings in binary form, denoting by $BITS(x, (x^{(i)})_{i=0}^{l-1})$ the relation $[\![x]\!] = \sum_{i=0}^{l-1} [\![x^{(i)}]\!] 2^i$ with $x^{(i)} \in \{0, 1\}$. The protocol Bit-add$((\![x^{(i)}]\!])_{i=0}^{l-1}; (\![y^{(i)}]\!])_{i=0}^{l-1})$ returns $(\![z^{(i)}]\!])_{i=0}^{l-1}$ such that there holds $z = x + y$ for $BITS(x, (\![x^{(i)}]\!])_{i=0}^{l-1})$, $BITS(y, (\![y^{(i)}]\!])_{i=0}^{l-1})$ and $BITS(z, (\![z^{(i)}]\!])_{i=0}^{l-1})$. To implement it, let $c^{(-1)} = 0$. For $i = 0 \ldots l-1$:

$$[\![z^{(i)}]\!] = \text{Bit-xor}([\![x^{(i)}]\!], \text{Bit-xor}([\![y^{(i)}]\!], [\![c^{(i-1)}]\!]))$$

where Bit-xor$(a, b) = (a-b)^2$; and $[\![c^{(i)}]\!] = [\![x^{(i)}]\!][\![y^{(i)}]\!] + [\![c^{(i-1)}]\!]((1 - [\![x^{(i)}]\!])[\![y^{(i)}]\!] + [\![x^{(i)}]\!](1 - [\![y^{(i)}]\!]))$. After this loop, return $(\![z^{(i)}]\!])_{i=1}^{l}$.

2.6 Random Numbers

A (secure) *pseudo-random number generator* (PRG) is a function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ for some polynomial p with $p(k) > k$ such that for any randomized polynomial time algorithm $A : \{0,1\}^{p(k)} \to \{0,1\}$ there holds

$$|P_{x \leftarrow_R \{0,1\}^k}(A(G(x)) = 1) - P_{x \leftarrow_R \{0,1\}^{p(k)}}(A(x) = 1)| \leq \mu(k)$$

for some function μ negligible in k. In other words, a PRG is a function which takes a string x as input and outputs a longer string $G(x)$ which cannot be distinguished from a random sequence by a polynomial time algorithm.

3 PROBLEM STATEMENT

We call π a *sampling protocol* over a domain $\mathcal{X}$ if π is a randomized multi-party protocol which outputs sequences of elements of $\mathcal{X}$. We consider sampling protocols which take only one input per party at the beginning of the protocol. In particular, let $\mathbf{P} = \{P_i\}_{i=1}^n$ be the set of n parties which participate to a sampling protocol π, and let s_i be the input (also called *seed*) of party P_i (for $i \in [n]$). We denote the output of π by $\pi(\mathbf{s})$ where $\mathbf{s} = (s_i)_{i=1}^n$ is the vector of seeds. We assume that there is some increasing polynomial $p : \mathbb{N} \to \mathbb{N}$ such that if $\mathbf{s} \in \{0,1\}^{k \times n}$ then $\pi(\mathbf{s}) \in \mathcal{X}^{p(k)}$. Let $\mathbf{s}_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{|s|})$ denote the vector $\mathbf{s}$ without the i-th component.

Definition 1 (Correct Sampling). For a multi-party protocol π, we say a party is *honest* if it follows the steps of protocol π correctly and does not collude with other parties. We say that a sampling protocol π *correctly samples* from a probability distribution $\mathcal{D}$ if there is a function μ with $\mu(k)$ negligible in k such that for every run of π by parties $\mathbf{P} = \{P_i\}_{i=1}^n$ among which there is at least one $i \in [n]$ such that party P_i is honest, for every $\mathbf{s}_{-i} \in \{0,1\}^{k \times (n-1)}$, for any probabilistic polynomial time algorithm $A : \{0,1\}^{k(n-1)} \times \mathcal{X}^{p(k)} \to \{0,1\}$, there holds that either π finishes correctly and

$$|P_{s_i \leftarrow_R \{0,1\}^k}(A(\mathbf{s}_{-i}, \pi(\mathbf{s})) = 1) - P_{t \leftarrow_R \{0,1\}^{k(n-1)}, x \leftarrow_R \mathcal{D}^{p(k)}}(A(t, x) = 1)| \leq \mu(k),$$

or π aborts and detects a party that attempted to cheat, where $\mathcal{D}^{p(k)}$ draws vectors from $\mathcal{X}^{p(k)}$ whose components are independently distributed according to $\mathcal{D}$.

In other words, if there is at least one honest party, then π acts as a (generalized) PRG even if all parties except that honest party would disclose their seeds. As a result, as soon as a single party is honest it can trust that any output of π used by any party is pseudo-random and no party could predict it in advance. We denote the fact that x is correctly drawn from $\mathcal{D}$ by $x \leftarrow_R^* \mathcal{D}$.

We say a protocol π *verifiably samples* from $\mathcal{D}$ if π correctly samples from $\mathcal{D}$ and after every execution of π the value of x is uniquely defined given the union of the information obtained by all parties and the information published by π is sufficient to convince any party that x has been correctly drawn. We denote the fact that x is verifiably drawn from $\mathcal{D}$ by $x \leftarrow_R^V \mathcal{D}$.

In this paper, we will often informally consider both discrete and continuous probability distributions, and $P_{\mathcal{D}}$ then either represents a probability mass or probability density according to the context. As computers work with finite precision, we will eventually discretize up to some parameter-defined precision. While in the end all distributions will be discrete, we will use the continuous representations whenever this simplifies the explanation.

In the sequel, unless made explicit otherwise, we'll assume there are n parties among whom at least one is honest, and that $\mathcal{D}$ is a publicly agreed probability distribution. Also, to simplify the explanation we will often describe protocols generating just one random number, the extension to streams of random numbers is then straightforward.

We can distinguish several types of verifiable sampling protocols, depending on how they output the sampled number x. For a verifiable sampling protocol π, we say it is a

- **public draw** if after running π the value of x is published.
- **private draw** if after running π exactly one party knows x, but the other parties have no information on x next to the prior distribution $\mathcal{D}$.
- **hidden draw** if after running π the parties have received an additive secret share $(x_1, \ldots, x_n)$ for x, but still no party has any information about x next to the prior distribution $\mathcal{D}$.

In this paper, we study the problem of finding efficient verifiable sampling protocols of each of the three above types given the probability distribution $\mathcal{D}$.

This problem is reasonably straightforward if $\mathcal{D}$ is the uniform distribution over the integers in the interval $[0, L)$ for some $L > 0$:

Protocol 1 (Public uniform sampling). *For each $i \in [n]$ let party P_i generate its own random number r_i uniformly distributed over $[0, L)$ from its own secret seed s_i and publish a commitment C_i to it. Then, all parties open their commitment, i.e., they publish r_i and the randomness associated to the commitment to prove that C_i was a commitment to r_i. Finally, all parties compute publicly $\sum_{i=1}^{n} r_i \bmod L$.*

It is easy to see Protocol 1 draws r verifiably: if at least one party P_i is honest, it has generated a uniformly distributed number r_i and r is also uniformly distributed because non-honest parties P_j cannot change their r_j as a function of other parties because they start with a commitment on their r_j. Note that Protocol 1 is a generalization for multiple parties of [10]. We present the protocol in more detail and prove its security in Appendix B.3.

Protocol 2 (Private uniform sampling). *One can sample a vector of k numbers private to P_1 as follows: P_1 draws uniformly at random a vector $\mathbf{a} = (a_1, \ldots, a_k) \in [0, L)^k$ and publishes a vector commitment C to it. Then, all parties generate jointly a public random number $r \in [0, L)$ with Protocol 1. P_1 expands r to random numbers $(r_1, \ldots, r_k) \in [0, L)^k$ using a PRG. Finally, for $i \in [k]$, P_1 computes $u_i = a_i + r_i \bmod L$ and performs a zero knowledge proof of the modular sum for each u_i. $(u_1, \ldots, u_k)$ is a vector of private uniform random numbers.*

Again, it is easy to see that $(u_1, \ldots, u_k)$ is drawn verifiably. This protocol has many aspects in common with the Augmented Coin-Tossing protocol defined in [29, Section 7.4.3.5]. We provide a proof of the security of Protocol 2 in Appendix B.4.

Protocol 3 (Hidden uniform sampling). *For each $i \in [n]$ let party P_i generate its own random number r_i uniformly distributed over $[0, L)$ and publish a commitment C_i to it. Then, they consider $(r_1, \ldots, r_n)$ as a secret share of the random number $r = \sum_{i=1}^{n} r_i \bmod L$.*

After running Protocol 3, if there is a honest party, r is fixed and follows the right probability distribution, and as not all parties collude no party knows more about r than that it follows the uniform distribution over $[0, L)$.

The problem of finding efficient verifiable sampling protocols becomes more challenging when $\mathcal{D}$ is not the uniform distribution, but a more general distribution such as a normal distribution or a Laplace distribution. Even for single party computation there sometimes exist multiple approaches with varying cost and precision.

4 RELATED WORK

Below we describe lines of work that are related to ours.

Multiparty Computation Between Unreliable Participants. The seminal work of [10] proposed the first protocol to sample a public random bit (i.e. tossing a coin) between two parties that do not trust each other. Subsequent works such as [13] proposed protocols to perform coin tossing between an arbitrary number of parties.

The work of [20] proved that in the malicious model without aborts it is impossible that a multiparty protocol is guaranteed to finish correctly and perform an unbiased coin toss if the number of malicious users is half or more of the total of participants. For such cases, there is no other possibility than providing weaker security guarantees. In the framework of *malicious security with abort*[29], protocols either end correctly or are aborted by malicious parties. This could lead to bias in the computations if a protocol is restarted after an abort and the adversary speculatively chooses when allow the protocol to finish correctly. To prevent this, a possible solution is to identify and punish malicious parties that cause aborts. The work of [3] proposes covert security, where cheating adversaries can get caught with certain probability. This is weaker than malicious security with abort, but allows cheaters to be detected. A stronger notion is *malicious security with identifiable abort*[33], where a party that cheats causes the protocol to abort with overwhelming probability and, in addition, the cheater is identified. Our work fits in that framework.

If deterrence measures are strong enough, this could be sufficient to discourage malicious behavior. Otherwise, if corrupted peers are willing to sacrifice themselves at any cost, other measures can be taken to attenuate the bias as much as possible [5, 43].

The work [30] proposes a method to securely perform a wide family of randomized computations (related to interactive games) over private data and private random numbers, using zero knowledge proofs to verify correctness. They prove that this is secure in the ideal paradigm without abort if the majority of parties is honest.

Sampling From Gaussians and Other Popular Non-Uniform Distributions. Distributions such as the Gaussian distribution, the Laplace distribution, the Poisson distribution or the exponential distribution are important in the field of statistics. Algorithms to securely draw from such distributions have applications in federated machine learning. Several contributions concern the problem of verifiable

noise for differential privacy [35, 50] and hence can benefit from secure drawing.

Even in the semi-honest model where parties follow the specified protocol, drawing hidden random numbers is sometimes non-trivial. For example, in [19] one needs to make a sum of statistics and a Laplacian-distributed noise term, hence the authors propose a protocol where parties generate random numbers summing to a Laplacian distributed value which can then be included in a secure aggregation without being revealed.

In [26], protocols are proposed to generate secret-shared samples for Gaussian, Exponential and Poisson distributions. For the Gaussian distribution, their approach generates samples by averaging uniform seeds, a method which we call the Central Limit Theorem (CLT) approach. We compare the CLT approach with our approaches in Section 9. Even if more than a decade has passed since [26], recent contributions still resort to these techniques to generate Gaussian samples among unreliable participants. For example, recent protocols use the technique of [26] by adapting it to generate private draws from the Exponential distribution [45] and to sample hidden draws from the Binomial distribution [9]. The work of [41] proposes techniques to securely sample from the geometrical and Gaussian distributions, both building on [26], and studies them in the light of differentially private memory access patterns. In addition, [41] defines an extension of the malicious security model which includes information leakage, as measured in differential privacy, and proves the security of their protocols within this model.

In our work, we propose new techniques for privately drawing from the Gaussian distribution and show that all our techniques for all but the lowest precision requirements outperform the technique of [26], which is the most efficient method known so far. The same dynamics are at play for exponential, Poisson and Laplace distributions. Compared to the techniques in [26], our methods have a better complexity as a function of the precision parameter. We extend our methodology to hidden draws of Gaussian, Laplacian and Arbitrary distributions. Achieving sufficient precision when sampling is important for both the statistical quality and the security of the algorithms [42].

Implementation of Math Functions Using Cryptographic Primitives. Using secret sharing techniques, there is a large body of work on how to compute math functions such as square roots, logarithms and trigonometric [1, 4, 25, 32, 32, 37]. However they usually rely on splines or other approximation techniques that approximate functions by splitting the domain and using low-degree polynomials for each part. Alternatively, they rely on rational approximations. Piecewise approximations require the use of conditionals which are expensive when computing with secret shares, and rational approximations only allow for a fixed precision. Our work uses iterative approximations which allow to customize the precision of the approximation and are easy to compute given that we avoid the use of comparison gates in our circuits. Furthermore, for the Gaussian distribution, piecewise approximations require an external method to sample from the tails of the distribution. We also show protocols for private sampling from Gaussian and Laplace distributions where we avoid the high cost of secret shared computation by letting one party perform the calculation and then prove correct behavior using compressed Σ-protocols.

Zero Knowledge Proofs for such functions, as we apply in our work, is a less explored technique. [54] proposes techniques to prove a limited set of relations involving common activation functions in machine learning.

5 METHOD

We start with discussing two generic approaches: a strategy based on the inverse cumulative probability distribution and a strategy based on table lookup.

5.1 Inverse Cumulative Probability Distribution

Assume $\mathcal{D}$ is a probability distribution on $\mathcal{X} \subseteq \mathbb{R}$. The cumulative probability distribution is defined as

$$F_{\mathcal{D}}(x) = \int_{t \leq x} P_{\mathcal{D}}(t)\,\mathrm{d}t$$

To the extent $\mathcal{D}$ is discrete, we can see $P_{\mathcal{D}}$ as a sum of scaled Dirac delta functions over which integration is possible and results in a sum. Then, the inverse $F^{-1}_{\mathcal{D}}$ is a function on the interval $(0, 1)$.

An approach to sampling from arbitrary distributions $\mathcal{D}$ on domains $\mathcal{X} \subseteq \mathbb{R}$, known as the inversion method, consists of sampling uniformly from the $(0, 1)$ interval and applying the inverse of the cumulative distribution function $F^{-1}_{\mathcal{D}}$. Indeed, if $t \leftarrow_R (0, 1)$, then $P(F^{-1}_{\mathcal{D}}(t) = x) = P_{\mathcal{D}}(x)$.

Public Sampling From an Arbitrary Distribution. This approach can easily be applied to draw random numbers publicly:

Protocol 4 (Public draw from arbitrary distribution). *Run protocol 1 to generate a public uniformly distributed random number r', and then publicly compute $r = F^{-1}_{\mathcal{D}}(r')$.*

Using the inversion method for private or hidden draws is more involved since one needs a multi-party algorithm to compute $F^{-1}_{\mathcal{D}}$ or a ZKP algorithm to prove to other parties that $F^{-1}_{\mathcal{D}}$ was applied correctly. In many practical cases, $F^{-1}_{\mathcal{D}}$ does not have a simple closed form. This especially holds for the Gaussian distribution which we will discuss in more detail in Section 8.

We can extend this method to multi-variate distributions. For example, consider a distribution $\mathcal{D}$ over $\mathbb{R}^2$. To sample a pair (x, y) according to $\mathcal{D}$, we first define $P_x(x) = \int P_{\mathcal{D}}(x, y)\,\mathrm{d}y$, apply the inversion method to draw a random number x according to P_x, and then define $P_{y|x}(y) = P_{\mathcal{D}}(y|x) = P_{\mathcal{D}}(x, y)/P_x(x)$ and apply again the inversion method to draw a random y.

5.2 Table Lookup

As pointed out above, practical inverse cumulative probability functions are often expensive to compute, especially in a secure multi-party setting. In such scenarios approaches such as the ones discussed in Sections 5.1 and 8 incur a high cost for each drawn random number. In this section we consider an approach based on *table lookup*. While the involved techniques are well-known, this approach is interesting as a baseline, especially as it has a number of

properties which are different from the other methods considered in this paper. In particular, the method studied here has a high pre-processing cost but then allows for drawing random numbers at a low constant cost per drawn random number.

Protocol 5 (Table-lookup private sampling).

- *Preprocessing. Let* $M \in \mathbb{N}$. *The parties publicly pre-compute the pairs* $\left(i, F_{\mathcal{D}}^{-1}\left(\frac{2i-1}{2M}\right)\right)$ *for all* $i \in [M]$ *and store them into a database* DB.
- *Sampling. Party* P_1 *privately draws using Protocol 2 a random number* r' *distributed uniformly in* $[M]$. *Then,* P_1 *sets* $r = F_{\mathcal{D}}^{-1}\left(\frac{2r'-1}{2M}\right)$, *publishes commitments to* r' *and* r, *and publishes a ZKP that* $(r', r) \in$ DB.

In Protocol 5 a zero knowledge set membership proof is needed. There is a large body of work on this topic since in [15] the first method was shown that has a large preprocessing cost (linear in M) but only a unit communication cost for proving membership. Several improvements have been proposed which vary in their assumptions and efficiency, [7] discusses some lines of recent work.

Only already storing the database DB may take a prohibitive amount of space if a high precision is needed, as M is exponential in the number of desired correct digits. As a result, this technique can only be used when the needed precision is not too high. If it is feasible, it is expensive for drawing only a few random numbers but it can become more efficient than other methods if a huge number of random numbers need to be drawn, as asymptotically the cost per sample will dominate.

5.3 Laplace Distribution

The Laplace distribution, denoted $Lap(b)$ is defined by

$$P_{Lap(b)}(x) = \exp(-|x|/b)/2b.$$

The cumulative distribution is

$$F_{Lap}(x) = 1/2 + \text{sign}(x)/2 - \text{sign}(x)\exp(-|x|/2).$$

To sample a number r from $Lap(b)$ it is convenient to separately draw the sign s and absolute value a of r. Then, $P(s = -1) = P(s = 1) = 1/2$ and $P(a) = \frac{1}{b}\exp\left(-\frac{a}{b}\right)$ and $P(a \leq t) = 1 - \exp\left(-\frac{t}{b}\right)$. In Section 7 we will describe protocols for both private and hidden Laplace-distributed draws.

5.4 Gaussian Distribution

The Gaussian distribution, denoted by $\mathcal{N}(\mu, \sigma^2)$, is defined by

$$P_{\mathcal{N}(\mu,\sigma^2)}(x) = \exp(-(x-\mu)^2/2\sigma^2)/\sqrt{2\pi}\sigma.$$

We will sometimes use the shorthand $P_{\mathcal{N}} = P_{\mathcal{N}(0,1)}$. The cumulative distribution is

$$F_{\mathcal{N}}(x) = (1 + \text{erf}(x/\sqrt{2}))/2 \tag{1}$$

where erf is the error function. There is no closed form for $P_{\mathcal{N}}$, $F_{\mathcal{N}}$ nor its inverse. In the single party setting multiple strategies have been investigated to sample from this important distribution:

- the Central Limit Theorem (CLT) approach, which consists of sampling repeatedly from a uniform distribution and computing the average, which is simple but requires $O(1/\Delta^2)$ time for a root mean squared error Δ,
- the Box-Müller method [12], that can obtain two Gaussian numbers from two uniform samples by the application of a closed form formula, but involves the computation of a square root, trigonometric functions and a logarithm,
- rejection sampling methods, such as the polar version of Box-Müller [36] or the Ziggurat Method [40] are efficient and highly accurate. While the former avoids the computation of trigonometric functions and leads to an efficient verifiable implementation, the latter uses several conditional branches which are expensive to prove in zero knowledge and requires an external method for sampling in the tails of the distribution,
- the inversion method for Gaussians that involves the approximation of the inverse error function erf^{-1}, which can be done with rational functions or Taylor polynomials, and
- the recursive method of Wallace [51], which is very popular for its efficiency, but requires as input a vector of already generated Gaussian samples to generate an output vector of the same size; furthermore, samples from input and output vectors are correlated, which deteriorates the statistical quality.

Before studying some of these in the multi-party setting, we will first provide Σ-protocols of relations involving approximations of certain elementary functions.

6 PROOFS OF ELEMENTARY FUNCTIONS

In this section, we construct zero knowledge proofs of statements that involve the approximation of elementary functions, i.e. sine, cosine, natural logarithm and square root. These functions can be numerically approximated using basic operations such as addition and multiplication. While classic cryptographic tools are used to prove statements over integers, we operate with real numbers which we approximate with fixed precision. Therefore, we use representations of integer multiples of $2^{-\psi}$ by multiplying our values with 2^{ψ} and rounding them deterministically to obtain elements of $\mathbb{Z}_p$. Negative numbers are represented in the upper half of $\mathbb{Z}_p$. For example, the number $a < 0$ is represented with $p + 2^{\psi}a$. The set of representable numbers is denoted by

$$\mathbb{Q}_{\langle p,\psi\rangle} = \{v \in \mathbb{Q} : 2^{\psi}v \in \mathbb{Z} \wedge -p/2 \leq 2^{\psi}v < p/2\}$$

which is closed under addition and multiplication modulo p (rounded up to $2^{-\psi}$). The encoding of $v \in \mathbb{Q}_{\langle p,\psi\rangle}$ is denoted by $\langle v\rangle = 2^{\psi}v \bmod p$.

We show circuits such that the nullity of their output is equivalent to the statements we want to prove. We will first construct circuits to describe low level statements and then use these as building blocks for higher level statements. In the end, we apply compressed Σ-protocols (see Section 2.4) to produce zero knowledge proofs of these circuits. For parameters $(a; b)$ of all circuits defined below, a always contains public constants and b private values.

We present in Section 6.1 circuits for proving various types of simple statements. In Section 6.2, we introduce Cordic, the core approximation algorithm. We implement circuits to prove its correct execution in Section 6.3, and details on how to expand its domain

of application, particularly for our sampling techniques, in Section 6.4.

6.1 Building Blocks

We introduce below proofs of basic statements that we will use to prove approximations, including the handling of some statements of numbers in $\mathbb{Q}_{\langle p,\psi\rangle}$. Note that additions, multiplication by an integer and range proofs port directly to $\mathbb{Z}_p$ by our encoding $\langle\cdot\rangle$. In Appendix A.5, we show that to prove that an integer $x \in \mathbb{Z}_p$ belongs to $[0, 2^k)$ we can use the circuit

$$C_{Ra}(k; x, \mathbf{x}) \coloneqq \begin{bmatrix} \mathbf{x} * (1 - \mathbf{x}) \\ x - \sum_{i=1}^{k} x_i 2^{i-1} \end{bmatrix}$$

where $\mathbf{x} = (x_1, \ldots, x_k)$ is the bit map of x. Here, $\mathbf{x} * (1 - \mathbf{x})$ is a vector with at position i the value $x_i(1-x_i)$, which is 0 if $x_i \in \{0, 1\}$. The second expression evaluates to 0 if $\mathbf{x}$ is indeed the correct bit map of x. Hence, the nullity of the circuit, i.e., its righthandside evaluating to the zero vector, proves $x \in [0, 2^k)$.

Generalized Range Proof. C_{Ra} can be used twice to prove membership in any range $[a, b] \subset \mathbb{Z}_p$. To prove $x \in [a, b]$ we use the circuit

$$C_{GRa}(a, b; x, \mathbf{s}_1, \mathbf{s}_2) \coloneqq \begin{bmatrix} C_{Ra}(\lfloor \log(b-a) \rfloor + 1; x - a, \mathbf{s}_1) \\ C_{Ra}(\lfloor \log(b-a) \rfloor + 1; b - x + a, \mathbf{s}_2) \end{bmatrix}$$

where $\mathbf{s}_1, \mathbf{s}_2 \in \{0, 1\}^{\lfloor \log(b-a) \rfloor + 1}$ the auxiliary bit vectors required for C_{Ra}.

(Right) Bit-Shift. For $a \in \mathbb{Q}_{\langle p,\psi\rangle}$ and an integer $k > 0$, a bit shift $a >> k$ is equal to the biggest value in $\mathbb{Q}_{\langle p,\psi\rangle}$ smaller than $a/2^k$. We have that $b = a >> k$ if $\langle a\rangle - 2^k\langle b\rangle \in [0, 2^k)$. For the vector $\mathbf{s}$ of the bit decomposition of $\langle a\rangle - 2^k\langle b\rangle$, the circuit is

$$C_{>>}(k; \langle a\rangle, \langle b\rangle, \mathbf{s}) \coloneqq C_{Ra}(k; \langle a\rangle - 2^k\langle b\rangle, \mathbf{s}).$$

Note that in the definition of $C_{>>}$, as in all subsequent circuits, the evaluation of inputs to sub-circuits such as C_{Ra} are computations performed within the circuit.

Approximate Product. For private $a, b, c \in \mathbb{Q}_{\langle p,\psi\rangle}$, it can be proven that c is the rounding of ab, that is by proving that $\langle c\rangle - \langle a\rangle\langle b\rangle + \langle 1/2\rangle \in [0, 2^\psi)$. For $\mathbf{s} \in \{0, 1\}^\psi$ the bitmap of $\langle c\rangle - \langle a\rangle\langle b\rangle + \langle 1/2\rangle$ our circuit is

$$C_{Prod}(\psi; \langle a\rangle, \langle b\rangle, \langle c\rangle, \mathbf{s}) \coloneqq C_{Ra}(\psi; \langle c\rangle - \langle a\rangle\langle b\rangle + \langle 1/2\rangle, \mathbf{s}).$$

Approximate Division. For private $a, b, c \in \mathbb{Q}_{\langle p,\psi\rangle}$, we prove that c is approximately a/b with error $2^{-\psi}$. We also require that $b \in [A, B]$ for public $A, B \in \mathbb{Q}_{\langle p,\psi\rangle}$. We prove that $\langle b\rangle\langle c\rangle - 2^\psi\langle a\rangle + \langle b\rangle \in [0, 2\langle b\rangle)$. Our range proofs require that the bounds are public, so we prove that $\langle b\rangle\langle c\rangle - 2^\psi\langle a\rangle + \langle b\rangle \in [0, 2^{s_b+1})$ and $2^\psi\langle a\rangle - \langle b\rangle\langle c\rangle \in [0, 2^{s_b+1})$ where $s_b = \lfloor \log_2(\langle B\rangle - \langle A\rangle) \rfloor + 1$. For $\mathbf{s_1}, \mathbf{s_2} \in \mathbb{Z}_p^{s_b+1}$ auxiliary bit vectors, the circuit is

$$C_{Div}(\psi, s_b; \langle a\rangle, \langle b\rangle, \langle c\rangle, \mathbf{s}_1, \mathbf{s}_2) \coloneqq \begin{bmatrix} C_{Ra}(s_b; \langle b\rangle\langle c\rangle - 2^\psi\langle a\rangle + \langle b\rangle, \mathbf{s_1}) \\ C_{Ra}(s_b; 2^\psi\langle a\rangle - \langle b\rangle\langle c\rangle, \mathbf{s}_2) \end{bmatrix}.$$

Exponentiation in $\mathbb{Z}_p$. Let $y, x \in \mathbb{Z}_p$ be private values with $x \in [0, 2^k)$ and $E \in \mathbb{Z}_p$ a public integer such that $E^x < p/2$. We prove that $y = E^x$. Let $\mathbf{x} \in \{0, 1\}^k$ the vector of bits of x, we prove that $y = \prod_{y=1}^{k} y_i$ where for $i \in \{1, \ldots, k\}$, y_i is equal to $E^{2^{i-1}}$ if $\mathbf{x}_i = 1$, or to 1 if $\mathbf{x}_i = 0$. The circuit is

$$C_{IEx}(k, E; x, y, \mathbf{x}) \coloneqq \begin{bmatrix} C_{Ra}(k; x, \mathbf{x}) \\ y - \prod_{i=1}^{k} 1 + \mathbf{x}_i(E^{2^{i-1}} - 1) \end{bmatrix}.$$

Modular Sum. We prove, for private $x, z \in \mathbb{Z}_p$ and public $y \in \mathbb{Z}_p$ such that all belong to $[0, M)$, that $z = x + y \bmod M$. Let $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{z}_1$ and $\mathbf{z}_2$ vectors of intermediate values for C_{GRa}, and let $b \in \{0, 1\}$, our circuit is

$$C_{Mod}(M, y; x, z, \mathbf{x}_1, \mathbf{x}_2, \mathbf{z}_1, \mathbf{z}_2, b) \coloneqq \begin{bmatrix} C_{GRa}(0, M-1; x, \mathbf{x}_1, \mathbf{x}_2) \\ C_{GRa}(0, M-1; z, \mathbf{z}_1, \mathbf{z}_2) \\ b(1-b) \\ z - (x + y - bM) \end{bmatrix}.$$

Ideas for C_{IEx} and C_{Mod} are taken from pages 112-115 of [16].

Private Magnitude Shift. Here, we prove that $y = x >> k$ for public K and private $k \le K$. Let $\mathbf{k}, \mathbf{k'}, \mathbf{k''} \in \{0, 1\}^K$ and $h \in \mathbb{Z}_p$ be intermediate values for range proofs and integer exponentiations and $\mathbf{I}_{>>} = (h, \mathbf{k}, \mathbf{k'}, \mathbf{k''})$, our circuit is

$$C_{P>>}(K; x, k, y, \mathbf{I}_{>>}) \coloneqq \begin{bmatrix} C_{IEx}(K, 2; k, h, \mathbf{k}) \\ C_{Ra}(K; x - hy, \mathbf{k'}) \\ C_{Ra}(K; h - x + hy - 1, \mathbf{k''}) \end{bmatrix}.$$

6.2 Cordic Algorithm

We use the Cordic algorithm [52] for approximations, which has long been state of the art for computations of elementary functions from simple operations [44]. Essentially, it uses the same core iteration algorithm, which only uses additions and bit-shifts, for all elementary function approximations. We will use Cordic parameterized for two settings described below, the first is used for sine and cosine and the second for square root and logarithm. In what follows, we only provide an algorithmic description of the Cordic algorithm as is needed in order to understand our extension to the zero knowledge setting in Section 6.3.

Setting 1 (Sine and Cosine). Let $\theta_0 = 0$. From input values X_0, Y_0, $\theta \in \mathbb{Q}_{\langle p,\psi\rangle}$, the following iterations are performed:

$$\begin{aligned} X_i &= X_{i-1} - \xi_i(Y_{i-1} >> i) \\ Y_i &= Y_{i-1} + \xi_i(X_{i-1} >> i) \\ \theta_i &= \xi_i \tan^{-1}(1 >> i) + \theta_{i-1} \end{aligned}$$

where $\xi_i \in \{-1, 1\}$ is equal to $\text{sign}(\theta - \theta_{i-1})$ and $\tan^{-1}(1 >> i)$ is taken from a precomputed table. Let ν be the total number of iterations, and let constants $K_{1,\nu} = \prod_{j=0}^{\nu} \sqrt{1 + (1 >> 2j)}$ and $K_1 = \lim_{\nu\to\infty} K_{1,\nu} \approx 1.6$. We have that

$$\lim_{\nu\to\infty} \begin{bmatrix} X_\nu \\ Y_\nu \\ \theta_\nu - \theta \end{bmatrix} = K_1 \begin{bmatrix} X_0 \cos\theta - Y_0 \sin\theta \\ X_0 \sin\theta + Y_0 \cos\theta \\ 0 \end{bmatrix}.$$

Recall the representation parameter ψ of $\mathbb{Q}_{\langle p,\psi\rangle}$ defined at the beginning of the section. By the convergence rate of Cordic, if $\psi \ge \nu + \lceil \log_2(\nu) \rceil + 1$, with input $X_0 = 1/K_{1,\nu}$, $Y_0 = 0$, and

$\theta \in [-\pi/2, \pi/2]$, then X_ν and Y_ν are approximations of $\sin(\theta)$ and $\cos(\theta)$ respectively with error at most $2^{1-\nu}$.

Setting 2 ($\ln(x)$ and $\sqrt{x}$). In this setting Cordic only takes two inputs $X_0, Y_0 \in \mathbb{Q}_{\langle p,\psi\rangle}$ and, with $\theta_0 = 0$, it performs the iterations

$$\begin{aligned} X_i &= X_{i-1} + \xi_i(Y_{i-1} >> F_i) \\ Y_i &= Y_{i-1} + \xi_i(X_{i-1} >> F_i) \\ \theta_i &= \xi_i \tanh^{-1}(1 >> F_i) + \theta_{i-1} \end{aligned}$$

with $\xi_i = \text{sign}(-Y_{i-1})$ and shift magnitude $F_i = i + 1 - k$ where the small value k is equal to the biggest integer such that $3^{k+1} + 2k - 1 \leq 2(i+1)$. Now let $K_{2,\nu} = \prod_{j=1}^{\nu} \sqrt{1 + (1 >> 2F_{j-1})}$ and $K_2 = \lim_{\nu\to\infty} K_{2,\nu} \approx 0.8$. In Setting 2, we have that

$$\lim_{\nu\to\infty} \begin{bmatrix} X_\nu \\ Y_\nu \\ \theta_\nu \end{bmatrix} = \begin{bmatrix} K_2\sqrt{X_0^2 - Y_0^2} \\ 0 \\ \tanh^{-1}\left(\frac{Y_0}{X_0}\right) \end{bmatrix}.$$

With ψ as in Setting 1, $x \in [\frac{1}{4}, 1)$ and fixing $X_0 = x + 1$, $Y_0 = x - 1$ we get by the identity $\ln(x) = 2\tanh^{-1}(\frac{x-1}{x+1})$ that θ_ν is an approximation of $\frac{1}{2}\ln(x)$ with error at most 2^{1-F_ν}. Similarly, for the same ψ and domain of x, $\sqrt{x}$ can be obtained by setting $(X_0, Y_0) = \left(x + \frac{1}{K_{2,n+1}^2}, x - \frac{1}{K_{2,n+1}^2}\right)$ with error at most 2^{1-F_ν}.

6.3 Cordic in Zero Knowledge

We first specify a set of statements that together are equivalent to a correctly performed Cordic computation. Note that the iterations in Settings 1 and 2 are very similar. Except for the correctness of the ξ_i values, they can be described by equations

$$\xi_i = -1 \vee \xi_i = 1 \quad \forall i \in \{1, \ldots, \nu\}, \tag{2}$$

$$Y_i = Y_{i-1} + \xi_i(X_{i-1} >> F_i) \quad \forall i \in \{1, \ldots, \nu\}, \tag{3}$$

$$X_i = X_{i-1} - m\xi_i(Y_{i-1} >> F_i) \quad \forall i \in \{1, \ldots, \nu\}, \tag{4}$$

$$\theta_\nu = \sum_{i=1}^{\nu} \xi_i \alpha_i, \tag{5}$$

where the constants in Setting 1 are $m = 1$, $F_i = i$ and $\alpha_i = \tan^{-1}(1 >> i)$ and in Setting 2, F_i is already defined, $m = -1$ and $\alpha_i = \tanh^{-1}(1 >> F_i)$. To prove the correct value of the ξ_i's, we avoid wide range checks at each iteration (on $\theta - \theta_i$ or Y_{i-1}), but instead we use properties of the convergence of Cordic: all of $\xi_1, \ldots, \xi_\nu \in \{-1, 1\}$ have been chosen correctly if

$$\theta_\nu - \theta \in [-\alpha_\nu, \alpha_\nu] \tag{6}$$

in Setting 1, and

$$Y_\nu \in [-2^{-F_{\nu-1}}, 2^{-F_{\nu-1}}] \tag{7}$$

in Setting 2.

We outline below the circuits that imply the above statements. Let $S \in \{1, 2\}$ be the Cordic setting that defines the involved constants. Let $\xi^* = (\xi_1, \ldots, \xi_\nu)$ and let

$$\mathbf{I} = (\langle X_i\rangle, \langle Y_i\rangle)_{i=1}^{\nu-1}, (\mathbf{s}_i, \mathbf{s}'_i, \langle X'_i\rangle, \langle Y'_i\rangle)_{i=0}^{\nu-1}, \xi^*)$$

be the vector of all intermediate values. The nullity of circuit

$$C_{Crd}(\nu, S; \langle X_0\rangle, \langle Y_0\rangle, \langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}) \coloneqq \begin{bmatrix} C_{>>}(F_i; \langle X_{i-1}\rangle, \langle X'_{i-1}\rangle, \mathbf{s}_{i-1}) & \forall i \in [\nu] \\ C_{>>}(F_i; \langle Y_{i-1}\rangle, \langle Y'_{i-1}\rangle, \mathbf{s}'_{i-1}) & \forall i \in [\nu] \\ (1 + \xi^*) * (1 - \xi^*) & \\ \langle Y_i\rangle - \langle Y_{i-1}\rangle - \xi_i\langle X'_{i-1}\rangle & \forall i \in [\nu] \\ \langle X_i\rangle - \langle X_{i-1}\rangle + m\xi_i\langle Y'_{i-1}\rangle & \forall i \in [\nu] \\ \langle\theta_\nu\rangle - \sum_{i=1}^{\nu} \xi_i\langle\alpha_i\rangle & \end{bmatrix}$$

is a proof of the core of the execution in eqs. (2) to (5). Here, for $i \in \{1, \ldots, \nu\}$, $\mathbf{s}_i, \mathbf{s}'_i \in \{0,1\}^{F_i}$ are auxiliary bit vectors to prove bit shifts of X_i and Y_i with result X'_{i-1} and Y'_{i-1} respectively.

We complete the above core circuit for Setting 1. Let

$$\gamma = \lfloor \log_2(2\langle\alpha_\nu\rangle)\rfloor + 1,$$

and let $\mathbf{s}_\alpha \in \{0,1\}^\gamma$ be the bit decomposition of $\langle\theta\rangle - \langle\theta_\nu\rangle$. The circuit

$$C_{Crd1}(\nu; \langle\theta\rangle, \langle X_0\rangle, \langle Y_0\rangle, \langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}, \mathbf{s}_\alpha) \coloneqq \begin{bmatrix} C_{Crd}(\nu, 1; \langle X_0\rangle, \langle Y_0\rangle, \langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}) \\ C_{Ra}(\gamma; \langle\theta\rangle - \langle\theta_\nu\rangle + \langle\alpha_\nu\rangle, \mathbf{s}_\alpha) \end{bmatrix}.$$

proves eqs. (2) to (6). Similarly, we extend the core circuit to a complete one for setting 2: for $\mathbf{s}_Y$ equal to the bit decomposition of Y_ν,

$$C_{Crd2}(\psi, \nu; \langle X_0\rangle, \langle Y_0\rangle, \langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}, \mathbf{s}_Y) \coloneqq \begin{bmatrix} C_{Crd}(\nu, 2; \langle X_0\rangle, \langle Y_0\rangle, \langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}) \\ C_{Ra}(\psi - F_\nu; \langle Y_\nu\rangle, \mathbf{s}_Y) \end{bmatrix}.$$

proves eqs. (2) to (5) and (7).

The instantiation of C_{Crd1} and C_{Crd2} for elementary functions is straightforward. Inputs that are intermediate values are defined as above. For trigonometric functions, we set $\mathbf{I}_T = (\langle\theta_\nu\rangle, \mathbf{I}, \mathbf{s}_\alpha)$ and use

$$C_{Tr}(\nu; \langle\theta\rangle, \langle c\rangle, \langle s\rangle, \mathbf{I}_T) \coloneqq C_{Crd1}(\nu; \langle\theta\rangle, \langle 1/K_{1,\nu}\rangle, 0, \langle s\rangle, \langle c\rangle, \mathbf{I}_T)$$

to compute $s = \sin(\theta)$ and $c = \cos(\theta)$. For the logarithm, let $\mathbf{I}_L = (\langle X_\nu\rangle, \langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}, \mathbf{s}_Y)$. Then,

$$C_{Log}(\psi, \nu; \langle x\rangle, \langle l\rangle, \mathbf{I}_L) \coloneqq \begin{bmatrix} C_{Crd2}(\psi, \nu; \langle x\rangle + \langle 1\rangle, \langle x\rangle - \langle 1\rangle, \mathbf{I}_L) \\ \langle l\rangle - 2\langle\theta_\nu\rangle \end{bmatrix}$$

proves $l = \ln(x)$. For the square root let $\mathbf{I}_S = (\langle Y_\nu\rangle, \langle\theta_\nu\rangle, \mathbf{I}, \mathbf{s}_Y)$, then

$$C_{Sqrt}(\psi, \nu; \langle x\rangle, \langle s\rangle, \mathbf{I}_S) \coloneqq C_{Crd2}(\psi, \nu; \langle x\rangle + \langle 1/4K_{2,\nu+1}^2\rangle, \langle x\rangle - \langle 1/4K_{2,\nu+1}^2\rangle, \langle s\rangle, \mathbf{I}_S)$$

proves $s = \sqrt{x}$.

Finally, we point out that, with minor adjustments to the above approximations, proofs of hyperbolic trigonometric functions and e^x can be obtained.

6.4 Extending the Domain

Here we extend the domain of approximations, which is necessary for our sampling applications. We sometimes do not define inputs such as bit vectors for range proofs and other intermediate values that are clear from the context or that are already defined in previous circuits.

Sine and Cosine. As shown, sine and cosine can be approximated in $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. For $Q \in \{1,2,3,4\}$ we use the identity

$$\sin\left(Q\frac{\pi}{2}+\theta'\right) = \begin{cases} \cos(\theta') & \text{if } Q=1 \\ -\sin(\theta') & \text{if } Q=2 \\ -\cos(\theta') & \text{if } Q=3 \\ \sin(\theta') & \text{if } Q=4 \end{cases}$$

extend the domain to $[0, 2\pi]$. Let $\mathbf{s}_\pi, \mathbf{s}'_\pi$ be bit vectors as needed for C_{GRa}, and $\mathbf{I}_{Tg} = (\langle\theta'\rangle, \langle s'\rangle, \langle c'\rangle, Q, \mathbf{s}_\pi, \mathbf{s}'_\pi, \mathbf{I}_T)$. Let

$$([j_1],[j_2],[j_3],[j_4]) = (\langle c'\rangle, -\langle s'\rangle, -\langle c'\rangle, \langle s'\rangle)$$

and

$$([k_1],[k_2],[k_3],[k_4]) = ([j_2],[j_3],[j_4],[j_1])$$

for $i \in \{1,2,3,4\}$ be literal variable replacements. Circuit

$$C_{TrG}(v; \langle\theta\rangle, \langle s\rangle, \langle c\rangle, \mathbf{I}_{Tg}) \coloneqq \begin{bmatrix} C_{Tr}(v; \langle\theta'\rangle, \langle s'\rangle, \langle c'\rangle, \mathbf{I}_T) \\ C_{GRa}(\langle -\frac{\pi}{2}\rangle, \langle\frac{\pi}{2}\rangle; \langle\theta'\rangle, \mathbf{s}_\pi, \mathbf{s}'_\pi) \\ \langle\theta\rangle - Q\langle\frac{\pi}{2}\rangle - \langle\theta'\rangle \\ \prod_{i=1}^4 (Q-i)^2 + ([j_i]-\langle s\rangle)^2 + ([k_i]-\langle c\rangle)^2 \end{bmatrix}$$

proves $s = sin(\theta)$ and $c = \cos(\theta)$ in the extended domain.

Natural Logarithm. We extend the domain of $\ln(x)$ to $(0,1)$. For $x' \in [\frac{1}{2}, 1)$ and non-negative integer e such that $x = 2^{-e}x' \in (0,1)$. We prove that $l = \ln(x') - e\ln(2) = \ln(x)$. Let $\mathbf{I}_{Lg} = (e, h, \mathbf{e}, \mathbf{s}_{x'}, \langle x'\rangle, \langle l'\rangle, \mathbf{I}_L)$, then

$$C_{LogG}(\psi, v; \langle x\rangle, \langle l\rangle, \mathbf{I}_{Lg}) \coloneqq \begin{bmatrix} C_{Log}(\psi, v; \langle x'\rangle, \langle l'\rangle, \mathbf{I}_L) \\ C_{Ra}(\psi-1; \langle x'\rangle - \langle 0.5\rangle, \mathbf{s}_{x'}) \\ C_{IEx}(\psi-1, 2; e, h, \mathbf{e}) \\ h\langle x\rangle - \langle x'\rangle \\ \langle l\rangle - \langle l'\rangle + e\langle\ln(2)\rangle \end{bmatrix}$$

proves our approximation.

Square Root. Now, for a public bound $B > 0$ and a private $x \in [0, B]$, we prove that $s = \sqrt{x}$. Let $\gamma = \lfloor \log_2(B) \rfloor + 1$. We choose $x' \in [\frac{1}{2}, 1)$ and an integer $e \in [-\psi, \gamma]$ such that $x = 2^e x'$, and we have that

$$\sqrt{x} = \begin{cases} 2^{e/2}\sqrt{x'} & \text{if } e \text{ is even} \\ 2^{(e+1)/2}\sqrt{x'/2} & \text{if } e \text{ is odd.} \end{cases}$$

We break the proof in several circuits to handle different cases. For that we use bit variables as flags to decide which computation will be proven. Let $n_e \in \{0,1\}$ be the "negativity flag" of e and $e' \geq 0$ such that $e = (1-n_e)e'$. Let $i_e \in \{0,1\}$ be the "parity flag" of e, such that $e = 2f - i_e$ for an integer f. We also define $f' \geq 0$ such that $f = (1-n_e)f'$. We first handle the relations between x, x', $s = \sqrt{x}$ and $s' = \sqrt{x'/(1+i_e)}$ when e is non-negative, or equivalently, when $n_e = 0$. Let $\mathbf{I}_{D1} = (l, \mathbf{f}, \mathbf{I}_{>>})$, then our circuit is

$$C_{SDom1}(B; \langle x\rangle, \langle s\rangle, \langle x'\rangle, \langle s'\rangle, i_e, e', f', \mathbf{I}_{D1}) \coloneqq \begin{bmatrix} C_{P>>}(\gamma; \langle x\rangle, e'+i_e, \langle x'\rangle, \mathbf{I}_{>>}) \\ e' - 2f' + i_e \\ C_{IEx}(\gamma, 2; f', l, \mathbf{f}) \\ \langle s\rangle - \langle s'\rangle l \end{bmatrix}.$$

Similarly, for $\mathbf{I}_{D2} = (h, \mathbf{e}, \mathbf{I}'_{>>})$ the case when e is negative is described by

$$C_{SDom2}(\psi; \langle x\rangle, \langle s\rangle, \langle x'\rangle, \langle s'\rangle, i_e, e', f', \mathbf{I}_{D2}) \coloneqq \begin{bmatrix} C_{IEx}(\psi-1, 2; e'-i_e, h, \mathbf{e}) \\ \langle x'\rangle - h\langle x\rangle \\ e' - 2f' - i_e \\ C_{P>>}(\psi-1; \langle s\rangle, f', \langle s'\rangle, \mathbf{I}'_{>>}) \end{bmatrix}.$$

Now we describe the main circuit. For $\mathbf{I}_D = (\langle x'\rangle, \langle s'\rangle, i_e, e', f')$ and $\mathbf{I}_{Sg} = (n_e, \mathbf{s}_{x'}, \mathbf{I}_S, \mathbf{I}_D, \mathbf{I}_{D1}, \mathbf{I}_{D2})$ vectors of intermediate values, we prove $s = \sqrt{x}$ with

$$C_{SqrtG}(\psi, v, B; \langle x\rangle, \langle s\rangle, \mathbf{I}_{Sg}) \coloneqq \begin{bmatrix} C_{Sqrt}(\psi, v; \langle x'\rangle, \langle s'\rangle, \mathbf{I}_S) \\ i_e(1-i_e) \\ n_e(1-n_e) \\ i_e * C_{Ra}(\psi-2; \langle x'\rangle - \langle 0.25\rangle, \mathbf{s}_{x'}) \\ (1-i_e) * C_{Ra}(\psi-1; \langle x'\rangle - \langle 0.5\rangle, \mathbf{s}_{x'}) \\ (1-n_e) * C_{SDom1}(B; \langle x\rangle, \langle s\rangle, \mathbf{I}_D, \mathbf{I}_{D1}) \\ n_e * C_{SDom2}(\psi, \langle x\rangle, \langle s\rangle, \mathbf{I}_D, \mathbf{I}_{D2}) \end{bmatrix}.$$

While the circuit above is easier to read, the practical implementation contains a number of further optimizations to reduce the number of multiplications. In particular, additional variables are introduced to avoid multiplying flags such as i_e with larger vectors such as the output of a C_{Ra} circuit. This introduces additional variables, e.g., $i_e * C_{Ra}(\psi-2; \langle x'\rangle - \langle 0.25\rangle, \mathbf{s}_{x'})$ would become $i_e(x' - x'_{aux})$ and $C_{Ra}(\psi-2; \langle x'_{aux}\rangle - \langle 0.25\rangle, \mathbf{s}_{x'_{aux}})$.

7 THE LAPLACE DISTRIBUTION

7.1 Private Laplace Sampling

Protocol 6 (private drawing from Laplace). *First, party P_1 privately draws s_0 and a' uniformly at random in $[0, L)$ with L sufficiently large (Protocol 2). Then, P_1 computes $a = -b\log(1-a'/L)$, $s = 2(s_0 \bmod 2) - 1$ and $r = sa$, and provides a ZKP for these relations (in Section 6.3 we showed a ZKP for the logarithm function, in Section 6.1 for approximate division).*

As Protocol 2 verifiably draws random numbers uniformly and for the other computations in Protocol 6 a ZKP is provided, Protocol 6 verifiably draws random numbers from the $Lap(b)$ distribution.

An alternative method could be based on work by [26] (see also [19] for related ideas). In particular, [26] proposes a technique to sample directly from the exponential distribution using a range of independent biased coin flips. The main advantage of our method is that we only need one uniformly sampled public random number, which strongly reduces the communication cost.

This remark also holds for the protocol of Laplace hidden draws which we will present in Section 7.2 below.

7.2 Hidden Laplace Sampling

We can also make hidden draws from the Laplace distribution, i.e., drawing a Laplace-distributed random number r as a secret share $[\![r]\!]$. For this, we build on the basic operations discussed in Section 2.5.

First, we observe that one can sample a sign s uniformly from $\{-1, 1\}$ as follows: the parties apply Protocol 3 to draw a secret

shared random number uniformly distributed in $\mathbb{Z}_p$, obtaining the sharing $[\![t]\!]$, next they multiply the sharing with itself to obtain $[\![t^2]\!]$ and open $[\![t^2]\!]$ to reveal t^2, and finally they multiply $[\![t]\!]$ with the public constant $1/\sqrt{t^2}$ to obtain $[\![s]\!] = \left[\!\left[t/\sqrt{t^2}\right]\!\right] \in \{-1, 1\}$. Drawing a secret shared random bit $b \in \{0, 1\}$ is then just drawing a sign $[\![s]\!]$ and computing $[\![b]\!] = ([\![s]\!] + 1)/2$ (this is protocol RAN_2 in [23]).

The Cordic algorithm for logarithm computation described in Section 6.2 requires only additions, bit shifts and comparisons (when setting $\xi_i = \text{sign}(-Y_{i-1})$). While it does not directly use multiplications, implementations of bit operations and comparisons, e.g., as in [23], often are using multiplications so the use of multiplications cannot be fully avoided. Alternative strategies to compute the logarithm suffer from similar challenges.

As in Section 7.1, we want to draw a number a' uniformly from $[0, L)$, compute $a = -b\log(1 - a'/L)$ and multiply it with a random sign s to get the random number *as* distributed according to $Lap(b)$. To compute $\log(x)$, Cordic expects $x \in [1/4, 1)$, so before applying Cordic we may need to scale its input to fit this interval.

We set $L = 2^l$ for some sufficiently large integer l and generate a' as an l-bit number, i.e., $[\![a']\!] = \sum_{i=0}^{l-1} \left[\!\left[a^{(i)}\right]\!\right] 2^i$ where $a^{(i)}$ are random bits.

We can find the highest zero bit of a' as follows: set $h_l = 0, h'_l = 1$ and $a^{(-1)} = 0$, and for $i = l - 1 \ldots - 1$, set $[\![h'_i]\!] = [\![h'_{i+1}]\!][\![1 - h_{i+1}]\!]$ and $[\![h_i]\!] = [\![h'_i]\!]\left[\!\left[1 - a^{(i)}\right]\!\right]$. The meaning of h_i then is 'bit i is the highest 0-bit', and the meaning of h'_i is 'the bits higher than bit i are all ones'. Exactly one h_i equals 1 and all others are 0 among $i = -1 \ldots l - 1$. We then can write $\log(1 - a'/L) = a^{\text{h}} + \log(x)$ with $[\![a^{\text{h}}]\!] = \sum_{i=0}^{l-1} [\![h_i]\!] \log(2^{i+1-l})$ and $BITS(x, (x^{(i)})_{i=0}^{l})$ where

$$([\![x^{(i)}]\!])_{i=0}^{l} = \textsc{Bit-add}(2^l, ([\![x_-^{(i)}]\!])_{i=0}^{l-1})$$

and $[\![x_-^{(i)}]\!] = \sum_{j=0}^{l-1} [\![h_j]\!][\![a^{(i+j+1-l)}]\!]$ (with $a^{(i)} = 0$ for $i < 0$).

Now we can apply Cordic on x. Cordic needs additions (using the Bit-add protocol), bit shifts (moving bits to the right and duplicating the highest bit), the sign$(\cdot)$ function (check the highest bit) and negation (invert all bits and add 1).

Protocol 7 (hidden drawing from Laplace). *One can verifiably draw a hidden Laplace-distributed random number by following the steps explained above, and by providing ZKP for all computations. The ZKP are similar to those for private sampling, where parts of secret shares are transfered between parties, the parties can agree on the commitment which will represent the shared number.*

8 THE GAUSSIAN DISTRIBUTION

In this section we elaborate several strategies to sample from the Gaussian distribution.

In particular, we are interested in protocols such that upon termination one party has a private number $y \in \mathbb{Q}_{\langle p,\psi\rangle}$ and has provided a zero knowledge argument that $y \sim \mathcal{N}(\mu, \sigma^2)$ for some public μ and σ.

All methods require as a subprotocol sampling uniformly distributed numbers. Therefore all our protocols for private drawing numbers from the Gaussian distribution follow the same high-level structure:

(1) use Protocol 2 to verifiably draw uniformly distributed number(s),
(2) transform the uniformly distributed number(s) into Gaussian distributed number(s), and
(3) use an arithmetic circuit matching this transformation together with compressed Σ-protocols (see Section 2.4) to prove the transformation.

We implement ZKPs of the correct execution of Gaussian draws. The sampling methods we implement are (1) the central limit theorem approach (averaging over uniform samples), (2) the Box-Müller method [12], (3) the Polar Method [39], (4) the inversion method using a series expansion for erf^{-1} [18], and (5) the inversion method using a fractional polynomial to approximate erf^{-1} [28].

8.1 The Central Limit Theorem Method

The uniform distribution over the interval $[0, L)$ has variance $L^2/12$. Let a party privately draw N random numbers $\{x_i\}_{i=1}^{N}$ uniformly distributed over $[0, L)$, and compute $x = \mu + \frac{\sigma\sqrt{12}}{L\sqrt{N}} \sum_{i=1}^{N} \left(x_i - \frac{L}{2}\right)$. Then, x is $\mathcal{N}(\mu, \sigma^2)$ distributed. For a ZKP for this relation between x and the x_i we only need the homomorphic property of the Pedersen commitment (for the additions and the multiplication with a constant) and a range proof (for the rounding). This method is essentially the technique of [26] to sample from the Gaussian distribution.

8.2 The Box-Müller Method

The Box-Müller method [12] consists of drawing two uniform samples U_1 and U_2 in the interval $(0, 1)$ and to compute

$$\rho = \sqrt{-2\ln(U_1)}, \quad X_1 = \rho\cos(2\pi U_2) \quad \textit{and} \quad X_2 = \rho\sin(2\pi U_2).$$

Then, X_1 and X_2 are distributed according to $\mathcal{N}(0, 1)$. Now we use circuits of elementary functions defined in Section 6 to construct our proof. Recall parameters ψ and the number of Cordic iterations ν defined therein. Let

$$\begin{aligned}\mathbf{I}_{BM} = (&\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \langle s\rangle, \langle c\rangle, \mathbf{I}_{Tg}, \langle a_\pi\rangle, \langle X'_1\rangle, \langle X'_2\rangle \\ &\mathbf{I}_{Sg}, \langle\rho\rangle, \mathbf{I}_{Lg}, \langle l\rangle, \mathbf{a}'_1, \mathbf{a}'_2, \mathbf{U}'_1, \mathbf{U}'_2, \mathbf{a}''_1, \mathbf{a}''_2, \mathbf{U}''_1, \mathbf{U}''_2)\end{aligned}$$

be a vector containing all intermediate values of the computation. Then the approximation circuit is

$$C_{BM}(\psi, \nu, z_1, z_2; \langle U_1\rangle, \langle U_2\rangle, \langle X_1\rangle, \langle X_2\rangle, \mathbf{I}_{BM}) := \begin{bmatrix} C_{Mod}(2^\psi - 1, z_1; a_1, \langle U_1\rangle - 1, \mathbf{a}'_1, \mathbf{a}'_2, \mathbf{U}'_1, \mathbf{U}'_2) \\ C_{Mod}(2^\psi - 1, z_2; a_2, \langle U_2\rangle - 1, \mathbf{a}''_1, \mathbf{a}''_2, \mathbf{U}''_1, \mathbf{U}''_2) \\ C_{LogG}(\psi, \nu; \langle U_1\rangle, \langle l\rangle, \mathbf{I}_{Lg}) \\ C_{SqrtG}(\psi, \nu, 2\psi\ln(2); -2\langle l\rangle, \langle\rho\rangle, \mathbf{I}_{Sg}) \\ C_{Prod}(\psi; 2\langle\pi\rangle, \langle U_2\rangle, \langle a_\pi\rangle, \mathbf{s}_1) \\ C_{TrG}(\nu; \langle a_\pi\rangle, \langle s\rangle, \langle c\rangle, \mathbf{I}_{Tg}) \\ C_{Prod}(\psi; \langle\rho\rangle, \langle c\rangle, \langle X'_1\rangle, \mathbf{s}_2) \\ C_{Prod}(\psi; \langle\rho\rangle, \langle s\rangle, \langle X'_2\rangle, \mathbf{s}_3) \end{bmatrix}.$$

Private values a_1 and a_2 and public challenges z_1 and z_2 are used in the modular proofs of circuit C_{Mod} to generate U_1 and U_2 with Protocol 2. To obtain a sample with standard deviation different than 1, the resulting samples can be scaled with an extra C_{Prod} circuit. Different mean requires an extra addition gate.

8.3 The Polar Box-Müller Method

The polar method [36] is an optimization of Box-Müller that avoids the computation of sine and cosine by the use of rejection sampling. It samples two uniform values V_1 and V_2 in the $(-1, 1)$ interval, and keeps the result only if $0 < V_1^2 + V_2^2 \leq 1$. Otherwise V_1 and V_2 are re-sampled. For non rejected V_1 and V_2 it computes

$$\alpha = V_1^2 + V_2^2, \quad Y_1 = V_1\sqrt{-2\ln(\alpha)/\alpha}, \quad \textit{and} \quad Y_2 = V_2\sqrt{-2\ln(\alpha)/\alpha}.$$

Y_1 and Y_2 have distribution $\mathcal{N}(0, 1)$.

In the private sampling, if V_1 and V_2 are rejected, the prover can just reveal them and start new uniform draws until acceptance, when it proves the correctness of accepting pairs. As in Box-Müller, parameters ν and ψ define our elementary function approximations, and a_1, a_2, z_1, z_2 are used to generate V_1 and V_2 in Protocol 2. Let

$$\begin{aligned}\mathbf{I}_{Pol} = (&\langle s\rangle, \mathbf{I}_{Sg}, \langle d\rangle, \langle\alpha\rangle, \langle l\rangle, \mathbf{I}_{Lg}, \mathbf{s}_6, \mathbf{s}_5, \mathbf{s}_4, \mathbf{s}_3, \mathbf{s}_2, \mathbf{s}_1, \langle\alpha\rangle,\\ &\langle V_2'\rangle, \langle V_1'\rangle, \mathbf{v}', \mathbf{v}, \mathbf{a}_1', \mathbf{a}_2', \mathbf{V}_1', \mathbf{V}_2', \mathbf{a}_1'', \mathbf{a}_2'', \mathbf{V}_1'', \mathbf{V}_2'')\end{aligned}$$

be a vector of intermediate computation values, then the implemented circuit is

$$C_{Pol}(\psi, \nu, z_1, z_2; \langle V_1\rangle, \langle V_2\rangle, \langle Y_1\rangle, \langle Y_2\rangle, \mathbf{I}_{Pol}) \coloneqq \begin{bmatrix} C_{Mod}(2^{2\psi} - 2, z_1; a_1, \langle V_1\rangle + \langle 1\rangle - 1, \\ \mathbf{a}_1', \mathbf{a}_2', \mathbf{V}_1', \mathbf{V}_2') \\ C_{Mod}(2^{2\psi} - 2, z_2; a_2, \langle V_2\rangle + \langle 1\rangle - 1, \\ \mathbf{a}_1'', \mathbf{a}_2'', \mathbf{V}_1'', \mathbf{V}_2'') \\ C_{Prod}(\psi; \langle V_1\rangle, \langle V_1\rangle, \langle V_1'\rangle, \mathbf{v}) \\ C_{Prod}(\psi; \langle V_2\rangle, \langle V_2\rangle, \langle V_2'\rangle, \mathbf{v}') \\ \langle\alpha\rangle - \langle V_1'\rangle - \langle V_2'\rangle \\ C_{GRa}(1, \langle 1\rangle - 1; \langle\alpha\rangle, \mathbf{s}_1, \mathbf{s}_2) \\ C_{LogG}(\psi, \nu; \langle\alpha\rangle, \langle l\rangle, \mathbf{I}_{Lg}) \\ C_{Div}(\psi; -2\langle l\rangle, \langle\alpha\rangle, \langle d\rangle, \mathbf{s}_3, \mathbf{s}_4) \\ C_{SqrtG}(\psi, \nu, 2^{\psi+1}\psi\ln(2); \langle d\rangle, \langle s\rangle, \mathbf{I}_{Sg}) \\ C_{Prod}(\psi; \langle s\rangle, \langle V_1\rangle, \langle Y_1\rangle, \mathbf{s}_5) \\ C_{Prod}(\psi; \langle s\rangle, \langle V_2\rangle, \langle Y_2\rangle, \mathbf{s}_6) \end{bmatrix}.$$

To avoid multiple interactions due to rejection, Protocol 2 is set to draw a sufficiently large number of uniform samples such that at least one pair is not rejected with high probability. The cost of the extra samples is negligible. To obtain a distribution with different mean and variance, we scale V_1 and V_2 with addition and product.

8.4 The Inversion Method

Inverting eq. (1) we get

$$F_{\mathcal{N}}^{-1}(x) = \sqrt{2}\,\mathrm{erf}^{-1}(2x - 1).$$

There are many numerical strategies to approximate either erf or erf^{-1}, of which we implemented two.

A first strategy due to [18] is to use the series

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\sum_{l=0}^{\infty}\frac{(-1)^l x^{2l+1}}{l!(2l+1)}.$$

However, its approximation error gets larger as x gets bigger. Therefore, when x is large, it is better approximate $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$ using the series

$$\mathrm{erfc}(x) = \frac{e^{-x^2}}{x\sqrt{\pi}}\left(\sum_{l=0}^{L-1}\frac{(-1)^l(2l-1)!!}{(2x^2)^l}\right) + R_L(x),$$

where

$$R_L(x) \leq \frac{e^{-x^2}}{x\sqrt{\pi}}\frac{(2L-1)!!}{(2x^2)^L}$$

is the remainder, $l!! = 1$ for $l < 1$ and $(2l-1)!! = \prod_{i=1}^{l}(2i-1)$. The number of terms L of the series is tuned for minimal error. If we set B to be the maximum error of our approximation, we use erf if $x < \sqrt{\frac{\ln(1/B)}{2} + 0.788}$ and erfc otherwise. The ZKP of this method proves that $y = \mathrm{erf}(x)$ or $1 - y = \mathrm{erfc}(x)$ depending on the domain of x.

A second strategy is proposed in [28] and uses a rational approximation. Therein, erf^{-1} is computed by

$$\mathrm{erfinv}_{\mathrm{SP}}(x) = \begin{cases} xp_1(w) & \text{if } w \leq 5 \text{ (central region)} \\ xp_2(s) & \text{if } w > 5 \text{ (tail region)} \end{cases}$$

where $w = -\log(1-x^2)$, $s = \sqrt{w}$ and p_1 and p_2 are two polynomials of degree 8. We use Cordic, product and range ZKPs to prove its computation.

8.5 Hidden Drawing

For the several strategies for sampling the Gaussian distribution described above, one can construct a protocol based on secret sharing for hidden sampling. Similar considerations apply as for the discussion in Section 7.2. As an example, we show a protocol using the Central Limit Theorem approach.

Protocol 8. *Let $N/12$ be a power of 4. For $i = 0 \ldots l-1$ and $j = 1 \ldots N$, draw a random bit sharing $[\![x_j^{(i)}]\!]$. This yields N random numbers $\{x_j\}_{j=1}^{N}$ in the interval $[0, 2^l)$. For $j \in [N]$ and $i = l \ldots l + \log_2 N$, set $x_j^{(i)} = 0$. Let $([\![y_1^{(i)}]\!])_{i=0}^{l+\log_2 N} = ([\![x_1]\!])_{i=0}^{l+\log_2 N}$ and for $j = 2 \ldots N$ let*

$$([\![y_j^{(i)}]\!])_{i=0}^{l+\log_2 N} = \textsc{Bit-add}(([\![y_{j-1}^{(i)}]\!])_{i=0}^{l+\log_2 N}, ([\![x_j^{(i)}]\!])_{i=0}^{l+\log_2 N}).$$

Finally let $[\![r]\!] = \sum_{i=\log_2(N/12)/2}^{l+\log_2 N} y_N^{(i)} 2^{i-\log_2(N/12)/2}$. Then, r approximates $\mathcal{N}(2^l\sqrt{3N}, 2^l)$. The computations can be made verifiable using a ZKP where parties send a number to each other can agree on using the same commitment.

9 EVALUATION

In this section, we present an empirical comparison of several methods to privately sample from the Gaussian distribution. We will publish code to reproduce all experiments together with the final version of this paper.

9.1 Setup

We evaluate the costs of the methods presented in Section 8. Namely, the Central Limit Theorem approach (CLT), the Box-Müller (BM), the Polar Method (PolM) and the inversion method. In the latter, we evaluate the two described strategies: using series (InvM-S) and rational approximations (InvM-R). Samples are generated for the $\mathcal{N}(0, 1)$ distribution.

We evaluate the cost of each method instantiated for several parameters against the statistical quality of the generated samples. For the computational cost we measure the exponentiations in $\mathbb{G}$ (GEX), which dominate the computation. The total communication

cost is the number of elements of $\mathbb{G}$ and $\mathbb{Z}_p$ sent (see Section 2.4). To measure the statistical quality, we generate 10^7 samples and measure the Mean Squared Error (MSE) from the ideal Gaussian CDF.

The varying parameter for BM and PolM is the number of iterations ν of their Cordic approximations, which is chosen between 2 and 14. For CLT we vary the number of averaged uniform terms between 2 and 400. For InvM-R, the number of terms of the approximation series is changed in order to obtain different approximations with errors between 0.5 and 2^{-20}. The rational approach InvM-R has no varying parameter. The representation parameter ψ which defines $\mathbb{Q}_{\langle p,\psi\rangle}$ is chosen to be the smallest as allowed by BM, PolM, InvM-S due to approximation constraints, and for CLT is set to optimize the quality/cost tradeoff .

9.2 Results

Figure 1 shows the communication costs, i.e., the number of elements of $\mathbb{G}$ required to prove one Gaussian draw. Note that, as described in Section 2.4, 6 elements of $\mathbb{Z}_p$ must be added to obtain the final cost. If high precision isn't important, CLT performs well, but in general PolM, BM and Inv-R give the best precision for a given computational investment. The Inv-R method is much simpler but can't be tuned to other precisions.

Figure 2 shows the number of GEX to prove (by the party who draws the number) or to verify (by another party) one Gaussian draw. Here too, BM and PolM are the most efficient methods as soon as a good statistical quality is required. We note that, as the communication cost is logarithmic in the number of inputs and multiplication gates, several parameter settings give different points in Figure 2 but may have the same communication cost, so in Figure 1 we just show the proof with best statistical quality.

For illustration, if we implement Pedersen commitments using the secp256k1 [1] elliptic curve, we obtain 128 bit security and an element of $\mathbb{G}$ can be represented with 257 bits. One GEX using this curve takes no more than 30 microseconds on an Intel Core i7-6600U at 2.60 GHz CPU. With BM, PolM and InvM-R, a sample with MSE $< 2^{-20}$ requires less than 900 Bytes of communication. With PolM, such sample takes less than 360 milliseconds (ms) to prove and 75 ms for its verification. While CLT quickly gets very expensive, if quality is less important and an MSE $> 2^{-13}$ is satisfactory, it is the most efficient approach. A proof of a sample using CLT with MSE 0.01 can be generated in less than 10 ms, verified in 3 ms and has a size of 482 Bytes. We also note that it is possible to further optimize our implementation using special-purpose algorithms [46] to compute multiple exponentiations in the form $\mathbf{g}^{\mathbf{b}}$.

Finally, in Figure 3 we show the gap in the communication cost between our PolM and CLT sampling techniques implemented with classic (non-compressed) Σ-protocols [21, 22] to the presented compressed techniques.

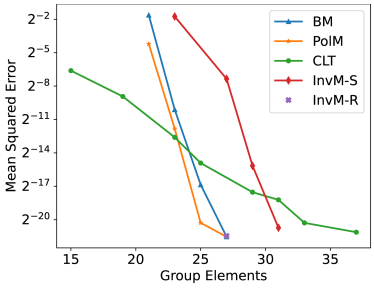
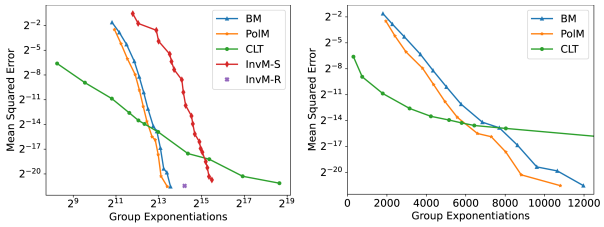
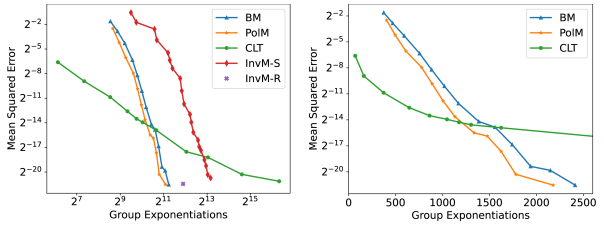


Figure 1: *(Comm. costs)* **Required Group Elements for one sample against the MSE for BM, PolM, InvM and CLT approaches.**



(a) Proving cost.



(b) Verification cost.

Figure 2: *(Comp. costs)* **Required group exponentiations for one sample against the MSE for BM, PolM, InvM and CLT approaches. Costs in the left side plots are in logarithmic scale. The right side plots are zooms of their left plots, in linear scale and with a detailed view on the most efficient methods.**

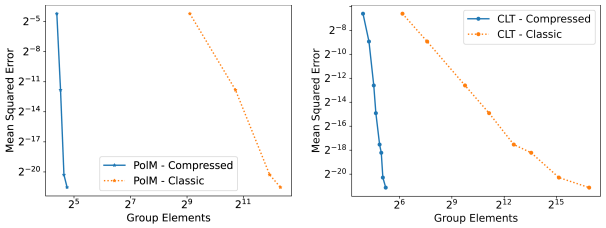


Figure 3: *(Comparison w. classic Σ-protocols)* **Required Group Elements for one sample against the MSE using compressed and classic Σ-protocols for PolM (left) and CLT (right).**

[1] See https://www.secg.org/SEC2-Ver-1.0.pdf and https://github.com/bitcoin-core/secp256k1

10 APPLICATION: DIFFERENTIALLY PRIVATE MACHINE LEARNING

An important application of verifiable sampling can be found in the field of federated machine learning under differential privacy. Consider parties $\mathbf{P} = \{P_i\}_{i=1}^n$ where each party P_i has some sensitive private data x_i. The parties $\mathbf{P}$ want to keep their data x_i private but want to collaborate to obtain statistical information θ of common interest. For example, assume that $x_i \in \mathbb{R}$ and the parties in $\mathbf{P}$ would like to compute $\theta = \frac{1}{n}\sum_{i=1}^n x_i$.

Even if no inputs nor intermediate results are revealed, computing and sharing the exact statistic θ may impact privacy. For example, suppose thet $x_i = 1$ if P_i likes a particular idea or $x_i = 0$ if P_i doesn't like it. It may turn out that no party likes the idea of interest, in which case we would get $\theta = 0$. If we publish that $\theta = 0$ no party can claim anymore that it liked the idea, so its privacy is lost. Statistical notions of privacy, such as differential privacy [27], add noise to guarantee the privacy of the individuals independently of the output. In particular, let θ be a function mapping datasets on values in $\mathcal{X}$. We say datasets are adjacent if they differ in the data of only one party. Then, we say a randomized algorithm A is (ϵ, δ)-differentially private (DP) if for any two adjacent datasets D_1 and D_2 and for any subset $X \in \mathcal{X}$, $P(A(D_1) \in X) \leq e^\epsilon P(A(D_2) \in X) + \delta$. A is ϵ-DP if it is $(\epsilon, 0)$-DP.

The most common strategy to make information DP before publication is to add noise from appropriately scaled Laplace or Gaussian distributions. For example, consider again the above example where the parties in $\mathbf{P}$ want to average their private x_i. Assume that $\forall i \in [n] : 0 \leq x_i \leq 1$. Then, for any $\epsilon > 0$, if we set $\hat{\theta} = \theta + \eta$ with $\eta \sim \text{Lap}(1/\epsilon)$ there holds that $\hat{\theta}$ is ϵ-DP. Alternatively, for any $\epsilon > 0$ and $\delta > 0$, if we set $\hat{\theta} = \theta + \eta$ with $\eta \sim \mathcal{N}(0, 2\ln(1.25/\delta)/\epsilon^2)$ there holds that $\hat{\theta}$ is (ϵ, δ)-DP.

It is important that no party knows the added noise η, because knowing both $\hat{\theta}$ and η would allow to reconstruct $\theta = \hat{\theta} - \eta$. We present two protocols, one privately drawing random numbers, which produces a less accurate result, and one based on the more expensive hidden drawing which has optimal precision.

Protocol 9 (DP learning using private sampling). *Let $\mathbf{P}_{cor} \subset \mathbf{P}$ be the set of corrupted parties of size at most ρn (with $0 \leq \rho < 1$). As described in Section 8, let all parties P_i ($i \in [n]$) privately verifiably draw a Gaussian random number $\eta \sim \mathcal{N}(0, \sigma^2/n(1-\rho))$, where noise with variance σ^2 on θ would be sufficient to achieve the desired privacy level. Then, securely sum $\hat{\theta} = \frac{1}{n}\sum_{i=1}^n (x_i + \eta_i)$ and publish $\hat{\theta}$.*

Even if the corrupted parties would collect all noise they have contributed $\eta_{coll} = \sum_{i \in \mathbf{P}_{cor}} \eta_i$ and subtracts it from $\hat{\theta}$ to obtain $\theta_{coll} = \hat{\theta} - \eta_{coll}/n$, then there is still Gaussian noise with variance $\frac{\sigma^2}{1-\rho} - \rho\frac{\sigma^2}{1-\rho} = \sigma^2$ left on their best estimation of θ. This strategy, which adopts some ideas from [26], works best for Gaussian noise, as the sum of Gaussian distributions is again a Gaussian distribution.

Protocol 10 (DP learning using hidden sampling). *Let all parties P_i (with $i \in [n]$) represent their private number x_i as a shared secret. Let the parties next together verifiably drawn a hidden random number η, i.e., a random number they obtain only as a shared secret. Finally, let them sum the secret shares and reveal $\hat{\theta} = \sum_{i=1}^n x_i + \eta$*

The advantage of this protocol is that no parties see η or parts of it, so it is impossible to get back towards the sensitive statistic θ. On the other hand, the full computation needs to be performed through multi-party computations, e.g., using shared secrets, which is clearly more expensive than the ZKPs which are needed in Protocol 9, especially as compressed Σ-protocols allow for ZKPs of size only logarithmic in the circuit size while calculations on secret shares have a linear communication cost.

As such drawing of random noise is a basic building block and needs to be performed repeatedly by secure federated differentially private machine learning algorithms, being able to draw from these probability distributions with low communication cost is essential to make algorithms more efficient.

Issues of Finite Precision. The work of [42] shows the impact that finite precision approximation of continuous distributions can have on DP guarantees. As explained in [42, Sec. 5.2], these vulnerabilities can be overcome by appropriately adjusting the precision and truncating the outcome after the noise is added to private values. This can be achieved in our protocols by using the correct precision parameters and range proofs.

To overcome vulnerabilities of finite precision approximations, other lines of work have explored the use of discrete distributions [17, 34]. These methods require in expectation comparable computational effort as our protocols. However, sampling from a discrete distribution requires in the worst case many iterations which results in longer zero-knowledge proofs.

11 CONCLUSION

We have presented novel methods for drawing random numbers in a verifiable way in a public, private and hidden setting. We applied the ideas to the Laplace and Gaussian distribution, and evaluated several alternatives to sample from the Gaussian distribution.

We see several interesting directions for future work. First, we hope to develop novel strategies to let our methods scale better when in the course of an algorithm many random numbers are needed. Second, we would like to develop new methods which allow for more efficient sampling in the hidden setting where the random numbers are output as shared secrets. In particular, our current methods based on generic secret sharing techniques require multiple rounds of computation and communication, it may be possible to develop more efficient special-purpose strategies.

ACKNOWLEDGMENTS

We thank Christian Weinert for fruitful discussions. This project was partially supported by ANR project ANR-20-CE23-0013 'PMR', the 'Chair TIP' project funded by ANR and MEL, and the Horizon Europe TRUMPET project grant no. 101070038.

REFERENCES

[1] Abdelrahaman Aly and Nigel P. Smart. 2019. Benchmarking Privacy Preserving Scientific Operations. In *Applied Cryptography and Network Security (Lecture Notes in Computer Science)*, Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung (Eds.). Springer International Publishing, Cham, 509–529.

[2] Thomas Attema and Ronald Cramer. 2020. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Advances in Cryptology – CRYPTO 2020 (Lecture Notes in Computer Science)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 513–543. https://doi.org/10.1007/978-3-030-56877-1_18

[3] Yonatan Aumann and Yehuda Lindell. 2010. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology* 23, 2 (2010), 281–343.

[4] Fattaneh Bayatbabolghani, Marina Blanton, Mehrdad Aliasgari, and Michael Goodrich. 2017. Secure fingerprint alignment and matching protocols.

[5] Amos Beimel, Eran Omri, and Ilan Orlov. 2010. Protocols for Multiparty Coin Toss with Dishonest Majority. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 538–557.

[6] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security (CCS '93)*. Association for Computing Machinery, New York, NY, USA, 62–73. https://doi.org/10.1145/168588.168596

[7] Danial Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. 2019. Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. Cryptology ePrint Archive, Report 2019/1255. https://ia.cr/2019/1255.

[8] David Bernhard, Olivier Pereira, and Bogdan Warinschi. 2012. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *Advances in Cryptology – ASIACRYPT 2012 (Lecture Notes in Computer Science)*, Xiaoyun Wang and Kazue Sako (Eds.). Springer, Berlin, Heidelberg, 626–643. https://doi.org/10.1007/978-3-642-34961-4_38

[9] Ari Biswas and Graham Cormode. 2022. Verifiable Differential Privacy For When The Curious Become Dishonest. https://doi.org/10.48550/ARXIV.2208.09011

[10] Manuel Blum. 1983. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News* 15, 1 (Jan. 1983), 23–27. https://doi.org/10.1145/1008908.1008911

[11] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In *Advances in Cryptology – EUROCRYPT 2016 (Lecture Notes in Computer Science)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Berlin, Heidelberg, 327–357. https://doi.org/10.1007/978-3-662-49896-5_12

[12] G. E. P. Box and Mervin E. Muller. 1958. A Note on the Generation of Random Normal Deviates. *Annals of Mathematical Statistics* 29, 2 (June 1958), 610–611. https://doi.org/10.1214/aoms/1177706645 Publisher: Institute of Mathematical Statistics.

[13] A.Z. Broder and D. Dolev. 1984. Flipping Coins In Many Pockets (Byzantine Agreement On Uniformly Random Values). In *25th Annual Symposium on Foundations of Computer Science, 1984.* IEEE, Singer Island, FL, USA, 157–170. https://doi.org/10.1109/SFCS.1984.715912

[14] Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, 315–334. https://doi.org/10.1109/SP.2018.00020

[15] Jan Camenisch, Rafik Chaabouni, and abhi shelat. 2008. Efficient Protocols for Set Membership and Range Proofs. In *Advances in Cryptology - ASIACRYPT 2008 (Lecture Notes in Computer Science)*, Josef Pieprzyk (Ed.). Springer, Berlin, Heidelberg, 234–252. https://doi.org/10.1007/978-3-540-89255-7_15

[16] Jan Camenisch and Markus Michels. 1999. Proving in Zero-Knowledge that a Number is the Product of Two Safe Primes. In *Advances in Cryptology — EUROCRYPT '99 (Lecture Notes in Computer Science)*, Jacques Stern (Ed.). Springer, Berlin, Heidelberg, 107–122. https://doi.org/10.1007/3-540-48910-X_8

[17] Clément L Canonne, Gautam Kamath, and Thomas Steinke. 2020. The Discrete Gaussian for Differential Privacy. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., Virtual, 15676–15688. https://proceedings.neurips.cc/paper/2020/file/b53b3a3d6ab90ce0268229151c9bde11-Paper.pdf

[18] S. Chevillard. 2012. The functions erf and erfc computed with arbitrary precision and explicit error bounds. *Information and Computation* 216 (July 2012), 72–95. https://doi.org/10.1016/j.ic.2011.09.001

[19] Seung Geol Choi, Dana Dachman-Soled, Mukul Kulkarni, and Arkady Yerukhimovich. 2020. Differentially-Private Multi-Party Sketching for Large-Scale Statistics. Cryptology ePrint Archive, Paper 2020/029. https://eprint.iacr.org/2020/029 https://eprint.iacr.org/2020/029.

[20] R Cleve. 1986. Limits on the Security of Coin Flips When Half the Processors Are Faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (Berkeley, California, USA) *(STOC '86)*. Association for Computing Machinery, New York, NY, USA, 364–369. https://doi.org/10.1145/12130.12168

[21] Ronald Cramer. 1997. *Modular Design of Secure yet Practical Cryptographic Protocols*. Ph. D. Dissertation. University of Amsterdam. https://ir.cwi.nl/pub/21438

[22] Ronald Cramer and Ivan Damgård. 1998. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free?. In *Advances in Cryptology — CRYPTO '98 (Lecture Notes in Computer Science)*, Hugo Krawczyk (Ed.). Springer, Berlin, Heidelberg, 424–441. https://doi.org/10.1007/BFb0055745

[23] I. Damgard, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft. 2006. Unconditionally secure constant rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC 2006 (LNCS, Vol. 3876)*. Springer, New York, NY USA, 285–304.

[24] Ivan Damgard, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In *ESORICS (LNCS, Vol. 8134)*. Springer, Egham, UK, 1–18.

[25] Vassil Dimitrov, Liisi Kerik, Toomas Krips, Jaak Randmets, and Jan Willemson. 2016. Alternative Implementations of Secure Real Numbers. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, New York, NY, USA, 553–564. https://doi.org/10.1145/2976749.2978348

[26] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *Advances in Cryptology - EUROCRYPT 2006*, Serge Vaudenay (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 486–503.

[27] Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.

[28] Mike Giles. 2012. Approximating the erfinv function. In *GPU Computing Gems Jade Edition*. Elsevier, Amsterdam, Netherlands, 109–116.

[29] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, Cambridge, England.

[30] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '87)*. Association for Computing Machinery, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18, 1 (Feb. 1989), 186–208. https://doi.org/10.1137/0218012 Publisher: Society for Industrial and Applied Mathematics.

[32] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. 2022. LLAMA: A Low Latency Math Library for Secure Inference. Cryptology ePrint Archive, Paper 2022/793. https://eprint.iacr.org/2022/793 https://eprint.iacr.org/2022/793.

[33] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. 2014. Secure Multi-Party Computation with Identifiable Abort. In *Advances in Cryptology – CRYPTO 2014*, Juan A. Garay and Rosario Gennaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–386. The full version can be found at https://eprint.iacr.org/2015/325.pdf..

[34] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, Virtual, 5201–5212. https://proceedings.mlr.press/v139/kairouz21a.html

[35] Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2021. Preventing Manipulation Attack in Local Differential Privacy Using Verifiable Randomization Mechanism. In *Data and Applications Security and Privacy XXXV*, Ken Barker and Kambiz Ghazinour (Eds.). Springer International Publishing, Cham, 43–60.

[36] R. Knop. 1969. Remark on algorithm 334 [G5]: normal random deviates. *Commun. ACM* 12, 5 (1969), 281. Publisher: ACM New York, NY, USA.

[37] Manuel Liedel. 2012. Secure Distributed Computation of the Square Root and Applications. In *Information Security Practice and Experience*, Mark D. Ryan, Ben Smyth, and Guilin Wang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 277–288.

[38] Lindell. 2003. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology* 16, 3 (June 2003), 143–184. https://doi.org/10.1007/s00145-002-0143-7

[39] G. Marsaglia and T. A. Bray. 1964. A Convenient Method for Generating Normal Variables. *SIAM Rev.* 6, 3 (July 1964), 260–264. https://doi.org/10.1137/1006063 Publisher: Society for Industrial and Applied Mathematics.

[40] George Marsaglia and Wai Wan Tsang. 2000. The ziggurat method for generating random variables. *Journal of statistical software* 5, 8 (2000), 1–7.

[41] Sahar Mazloom and S. Dov Gordon. 2018. Secure Computation with Differentially Private Access Patterns. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 490–507. https://doi.org/10.1145/3243734.3243851

[42] Ilya Mironov. 2012. On Significance of the Least Significant Bits for Differential Privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 650–661. https://doi.org/10.1145/2382196.2382264

[43] Tal Moran, Moni Naor, and Gil Segev. 2009. An Optimally Fair Coin Toss. In *Theory of Cryptography*, Omer Reingold (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[44] Jean-Michel Muller. 2016. *Elementary Functions: Algorithms and Implementation* (3 ed.). Birkhäuser Basel, Basel, Switzerland. https://doi.org/10.1007/978-1-4899-7983-4

[45] Gonzalo Munilla Garrido, Johannes Sedlmeir, and Matthias Babel. 2022. Towards Verifiable Differentially-Private Polling. In *Proceedings of the 17th International Conference on Availability, Reliability and Security* (Vienna, Austria) *(ARES '22)*.

Association for Computing Machinery, New York, NY, USA, Article 6, 11 pages. https://doi.org/10.1145/3538969.3538992
[46] Bodo Möller. 2001. Algorithms for Multi-exponentiation. In *Selected Areas in Cryptography (Lecture Notes in Computer Science)*, Serge Vaudenay and Amr M. Youssef (Eds.). Springer, Berlin, Heidelberg, 165–180. https://doi.org/10.1007/3-540-45537-X_13
[47] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91 (Lecture Notes in Computer Science)*, Joan Feigenbaum (Ed.). Springer, Berlin, Heidelberg, 129–140. https://doi.org/10.1007/3-540-46766-1_9
[48] David Pointcheval and Jacques Stern. 1996. Security Proofs for Signature Schemes. In *Advances in Cryptology — EUROCRYPT '96*, Ueli Maurer (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 387–398.
[49] Meital Ben Sinai, Nimrod Partush, Shir Yadid, and Eran Yahav. 2014. Exploiting Social Navigation. https://doi.org/10.48550/ARXIV.1410.0151
[50] Georgia Tsaloli and Aikaterini Mitrokotsa. 2019. Differential Privacy meets Verifiable Computation: Achieving Strong Privacy and Integrity Guarantees. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE 2019 - Volume 2: SECRYPT, Prague, Czech Republic, July 26-28, 2019*. SciTePress, Prague, Czech Republic, 425–430. https://doi.org/10.5220/0007919404250430
[51] C. S. Wallace. 1996. Fast pseudorandom generators for normal and exponential variates. *ACM Trans. Math. Software* 22, 1 (March 1996), 119–127. https://doi.org/10.1145/225545.225554
[52] J. S. Walther. 1971. A unified algorithm for elementary functions. In *Proceedings of the May 18-20, 1971, spring joint computer conference (AFIPS '71 (Spring))*. Association for Computing Machinery, New York, NY, USA, 379–385. https://doi.org/10.1145/1478786.1478840
[53] Simon Weckert. 2020. Google Maps hacks. Retrieved Dec 6, 2022 from http://www.simonweckert.com/googlemapshacks.html
[54] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. 2021. VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service. *IEEE Transactions on Parallel and Distributed Systems* 32, 10 (2021), 2524–2540. https://doi.org/10.1109/TPDS.2021.3068195

A COMPRESSED Σ-PROTOCOLS

In this appendix, we explain all necessary notions to understand compressed Σ-protocols [2]. In Appendix A.1, we explain the basic concepts of ZKP. Classic approaches to construct ZKP of linear relations are shown in Appendix A.2 and techniques to compress the communication cost in Appendix A.3. In Appendix A.4 we explain how linear proofs can be used to construct ZKP involving circuit computations. Finally, Appendix A.5 has an application for range proofs and Appendix A.6 discusses the costs of the techniques.

A.1 Zero Knowledge Proofs and Arguments

An interactive proof of knowledge (PoK) for an NP relation $\mathcal{R}$ is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ in which $\mathcal{P}$ tries to prove to $\mathcal{V}$ that they know a *witness* w such that $(a, w) \in \mathcal{R}$ for a public statement a. At the end of the protocol, $\mathcal{V}$ either accepts or rejects the proof. We denote by $(a; w)$ a member of a relation or an input of a protocol, using a semicolon to separate the public statement a from the private witness w. The tuple of all messages in a proof is called the *conversation* or *transcript*. Proofs may satisfy the following properties:

- **Completeness:** a proof is *complete* if $\mathcal{V}$ always accepts the proof when $(a; w) \in \mathcal{R}$ and $\mathcal{P}$ knows w.
- **Soundness:** a proof is *sound* if any prover whose proof for statement a is accepted by the verifier knows a valid witness w such that $(a; w) \in \mathcal{R}$ with overwhelming probability. The notion of soundness we use is called *witness extended emulation* [38]. Proofs that are sound only if the prover is computationally bounded are also called *arguments*.
- **Zero Knowledge:** a proof is *zero knowledge* if its transcript reveals no or negligible information about the witness other than its validity.

A.2 Σ-Protocols for Linear Relations

We now show how to prove linear relations over secret committed values. Recall the definition of Pedersen vector commitments presented in Section 2.2, which defines the commitment domain $\mathbb{Z}_p$, our underlying cryptographic group $\mathbb{G}$ and vector of elements $\mathbf{g}$. As explained in Section 2.2, we reserve one of the components of $\mathbf{g}$ for randomness so that commitments are hiding. For $L : \mathbb{Z}_p^k \to \mathbb{Z}_p$ a linear function in $\mathbb{Z}_p$, that is $L(x_1, \ldots, x_k) = a_1x_1 + \cdots + a_kx_k$ for coefficiens $a_1, \ldots, a_k \in \mathbb{Z}_p$, a vector commitment $P \in \mathbb{G}$ and a value $y \in \mathbb{Z}_p$, a prover $\mathcal{P}$ proves to know an opening $\mathbf{x} \in \mathbb{Z}_p^k$ of P such that $L(\mathbf{x}) = y$. We formally describe our linear relation by

$$\mathcal{R}_L = \{(P \in \mathbb{G}, y \in \mathbb{Z}_p; \mathbf{x} \in \mathbb{Z}_p^k) : P = \mathbf{g}^{\mathbf{x}} \wedge y = L(\mathbf{x})\}.$$

Note that in our case $\mathbf{x}$ has a coordinate reserved for randomness, so in valid relations the correspondent coefficient of L must be 0. However, in later auxiliary protocols we will not impose such restriction. To provide a zero knowledge proof for $\mathcal{R}_L$ we use a family of zero knowledge proofs called Σ-protocols [21] and its compressed version proposed in [2]. They provide soundness under the Discrete Logarithm Assumption (DLA). Protocol Π_0 below describes a classic proof of $\mathcal{R}_L$ originally stated for more general types of commitments defined in [21, 22] and which we instantiate for the Pedersen scheme. Π_0 takes as input $(P, y; \mathbf{x})$:

Protocol $\Pi_0(P, y, \mathbf{x})$**:**

(1) $\mathcal{P}$ computes: $\mathbf{r} \leftarrow_R \mathbb{Z}_p^k$, $A = \mathbf{g}^{\mathbf{r}}$, $t = L(\mathbf{r})$
(2) $\mathcal{P}$ sends to $\mathcal{V}$: A, t
(3) $\mathcal{V}$ sends to $\mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p$
(4) $\mathcal{P}$ sends to $\mathcal{V}$: $\mathbf{z} = c\mathbf{x} + \mathbf{r}$
(5) $\mathcal{V}$: if $\mathbf{g}^{\mathbf{z}} = AP^c$ and $L(\mathbf{z}) = cy + t$ then **accept**, else **reject**.

Theorem 2. *Π_0 is a complete, sound and zero knowledge proof of $\mathcal{R}_L$.*

We provide an intuition of how these properties are obtained. Completeness follows directly from the homomorphic property. For soundness, consider a prover $\mathcal{P}^*$ that by following Π_0 can produce an accepting transcript $((A, t), c_1, \mathbf{z}_1)$ with significant probability. Then it is shown that also with non-negligible probability, by using $\mathcal{P}^*$'s strategy many times, another accepting transcript of the form $((A, t), c_2, \mathbf{z}_2)$ can be produced, where the first message is equal in both transcripts and such that $c_1 \neq c_2$. Now, since both transcripts are accepting, we have $\mathbf{g}^{\mathbf{z}_1} = AP^{c_1}$ and $\mathbf{g}^{\mathbf{z}_2} = AP^{c_2}$ so $\mathcal{P}^*$ can efficiently compute the witness $\mathbf{x} = \frac{\mathbf{z}_1 - \mathbf{z}_2}{c_1 - c_2}$ such that $\mathbf{g}^{\mathbf{x}} = P$. Acceptance also implies that $L(\mathbf{z}_1) = c_1y + t$ and $L(\mathbf{z}_2) = c_2y + t$ and it follows that $L(\mathbf{x}) = y$. Therefore is $\mathbf{x}$ a valid witness. Either $\mathcal{P}^*$ already knew it or they can efficiently compute the discrete logarithms between components of $\mathbf{g}$, which is a contradiction due to the DLA.

Zero knowledge is obtained by showing that all information seen in the proof is random that does not depend on the secrets. By only knowing the statement (P, y) one can compute $\mathbf{z}' \leftarrow_R \mathbb{Z}_p$, $c' \leftarrow_R \mathbb{Z}_p$, $A' = P^{-c'}\mathbf{g}^{\mathbf{z}'}$ and $t' = L(\mathbf{z}') - c'y$, and the transcript $((A', t'), c', \mathbf{z}')$ has the same distribution as a conversation between

an honest prover and a honest verifier. Note that, as it is computed in reverse, $((A', t'), c', \mathbf{z}')$ cannot be efficiently produced in the actual protocol by a dishonest prover. Note that this reasoning only holds if the verifier is honest and generates its messages uniformly at random. To circumvent this, we implement our proofs transforming them into non-interactive proofs using the Strong Fiat-Shamir heuristic [8], where the verifier is replaced by a hash function and therefore without the need of involving a trusted party. The construction is secure under the Random Oracle Model [6].

A.3 Compression Mechanism

Now we reduce the communication cost of Π_0 using ideas of compressed Σ-protocols [2] and also present in [11, 14]. The transfer in Π_0 is dominated by the third message of the protocols in Step 4, with size of k elements of $\mathbb{Z}_p$. It can be reduced if instead of sending $\mathbf{z}$, $\mathcal{P}$ proves that $(AP^c, cy + t; \mathbf{z}) \in \mathcal{R}_L$ which would imply the condition tested in Step 5. Note that this proof does not need to be zero knowledge as $\mathbf{z}$ is originally revealed in Π_0. We first present Π_1, a proof of $\mathcal{R}$ that halves communication cost by "folding" $\mathbf{z}$ before sending it. Next, we show how to use this protocol to reduce cost of Π_0. By assuming that k is even, we define $\mathbf{g}_L = (g_1, \ldots, g_{k/2}) \in \mathbb{G}^{k/2}$ and $\mathbf{g}_R = (g_{(k/2)+1}, \ldots, g_k) \in \mathbb{G}^{k/2}$ and analogously for $\mathbf{x}_L \in \mathbb{Z}_p^{k/2}$ and $\mathbf{x}_R \in \mathbb{Z}_p^{k/2}$. We also define $L_L : \mathbb{Z}_p^{k/2} \to \mathbb{Z}_p$ and $L_R : \mathbb{Z}_p^{k/2} \to \mathbb{Z}_p$ such that $L_L(\mathbf{a}) = L(\mathbf{a}, 0)$ and $L_R(\mathbf{a}) = L(0, \mathbf{a})$. We use, an additional group element $\hat{g} \in \mathbb{G}$ generated in the same way as the components of $\mathbf{g}$.

Protocol $\Pi_1(P, y; \mathbf{x})$:

(1) $\mathcal{P}$ computes: $A = \mathbf{g}_R^{\mathbf{x}_L} \hat{g}^{L_R(\mathbf{x}_L)}$, $B = \mathbf{g}_L^{\mathbf{x}_R} \hat{g}^{L_L(\mathbf{x}_R)}$
(2) $\mathcal{P}$ sends to $\mathcal{V}$: A, B
(3) $\mathcal{V}$ sends to $\mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p$
(4) $\mathcal{P}$ sends to $\mathcal{V}$: $\mathbf{z} = \mathbf{x}_L + c\mathbf{x}_R$
(5) $\mathcal{V}$: if $(\mathbf{g}_L^c * \mathbf{g}_R)^{\mathbf{z}} \hat{g}^{cL_L(\mathbf{z}) + L_R(\mathbf{z})} = A(P\hat{g}^{L(\mathbf{x})})^c B^{c^2}$ then **accept**, else **reject**

Π_1 is a complete and sound proof of $\mathcal{R}_L$ with half the communication of Π_0. The communication of Step 4, can be further reduced by applying Π_1 recursively until the size of $\mathbf{z}$ is sufficiently small. Let $\Pi_B \diamond \Pi_A$ be the interactive proof obtained executing Π_A except for the last message and then executing Π_B. Now we can define to $\Pi_c = \Pi_1 \diamond \ldots \diamond \Pi_1 \diamond \Pi_0$ where the $\diamond$ is applied $\log_2(k) - 2$ times. Note that k requires to be a power of 2, but padding vectors with 0's is sufficient to fix this. Presented with more detail, it is proven in Theorem 3 of [2], that Π_c is a complete, sound and zero knowledge protocol for $\mathcal{R}_L$. Completeness is straightforward, and for zero knowledge it is sufficient to see that Π_0 is already zero knowledge, and the rest of the protocol only reveals as much as Π_0. Soundness follows from similar ideas than those shown for Theorem 2.

Amortization techniques can be applied to prove many nullity checks, where the prover claims for linear relations $L_1, \ldots, L_r$ that $L_i(\mathbf{x}) = 0$ for all $i \in \{1, \ldots, r\}$. For that, $\mathcal{V}$ sends a random value $\rho \leftarrow \mathbb{Z}_p$ and then $\mathcal{P}$ and $\mathcal{V}$ execute Π_c on input $(P, \sum_{i=1}^r \rho^{i-1} L_i, 0; \mathbf{x})$. If $L(\mathbf{x}) = 0$ then $L_i(\mathbf{x}) = 0$ for all i with overwhelming probability $1 - (r-1)/p$. Amortized nullity checks also hold when replacing linear forms by affine forms $\Phi_1, \ldots, \Phi_r$ where each one is the application of a linear form plus a constant. We denote this protocol by Π_N and its input by $(P, (\Phi_1, \ldots, \Phi_r); \mathbf{x})$. A prover can prove the opening of an affine map $\Phi : \mathbb{Z}_p^k \to \mathbb{Z}_p^r$ to $\mathbf{y} = (y_1, \ldots, y_r) \in \mathbb{Z}_p^r$ by running Π_N on input $(P, (\Phi_1 - y_1, \ldots, \Phi_r - y_r); \mathbf{x})$ where $\Phi_1, \ldots, \Phi_r : \mathbb{Z}_p^k \to \mathbb{Z}_p$ are the affine forms that compose Φ. The communication cost of these protocols sends $r - 1$ elements of $\mathbb{Z}_p$ more than Π_c, which account for the size of $\mathbf{y}$.

A.4 Proving Multiplications and Circuits

Now, we show the idea of [2] to prove multiplicative relations only with black-box access to Π_N. For a set committed triplets

$$(\alpha_1, \beta_1, \gamma_1), \ldots, (\alpha_m, \beta_m, \gamma_m),$$

$\mathcal{P}$ proves to $\mathcal{V}$ that $\alpha_i \beta_i = \gamma_i$ for all $i \in \{1, \ldots, m\}$. Let $\alpha = (\alpha_1, \ldots, \alpha_m)$, $\beta = (\beta_1, \ldots, \beta_m)$ and $\gamma = (\gamma_1, \ldots, \gamma_m)$.

Protocol Π_M**:**

(1) $\mathcal{P}$: samples random polynomials $f(X), g(X)$ in $\mathbb{Z}_p$ of degree at most m that define a secret sharing over α and β by fixing $f(i) = \alpha_i$ and $g(i) = \beta_i$ for all $i \in \{1, \ldots, m\}$ and sampling $f(0), g(0) \leftarrow_R \mathbb{Z}_p$.
(2) $\mathcal{P}$: computes the product polynomial $h(X) = f(X)g(X)$ which has degree at most $2m$.
(3) $\mathcal{P}$ sends to $\mathcal{V}$: a vector commitment of
$$\mathbf{x} = (\alpha, \beta, f(0), g(0), h(0), \ldots, h(2m)).$$
Note that $\gamma = (h(1), \ldots, h(n))$.
(4) $\mathcal{V}$ sends to $\mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p \setminus \{0, \ldots, m\}$
(5) $\mathcal{P}$ and $\mathcal{V}$ run Π_N to prove that $f(c), g(c), h(c)$ open to some points u, v and w respectively. This is possible by Lagrange interpolation, where by having sufficient points of f,g and h, which are in the commitment of $\mathbf{x}$, they can be evaluated its domain by applying an affine form to $\mathbf{x}$
(6) $\mathcal{V}$: if $uv = w$ then **accept** else **reject**

As before, completeness is straightforward. Zero knowledge follows from fact that f, g and h are random polynomials and their evaluations do not reveal information and that Π_N is zero knowledge. If the multiplicative relations does not hold in all triplets, the probability that $uv = w$ is negligible. From that and the soundness of Π_N, soundness is obtained.

Now, let $C : \mathbb{Z}_p^k \to \mathbb{Z}_p^s$ be a circuit, i.e. a function that only contains addition and multiplication gates in $\mathbb{Z}_p$. For a vector commitment P, we adapt ideas from Π_M to construct a proof that $\mathcal{P}$ knows a opening $\mathbf{x} \in \mathbb{Z}_p^k$ of P such that $C(\mathbf{x}) = 0$. Suppose that C has m multiplication gates. We enumerate the multiplication gates from 1 to m. For $i \in \{1, \ldots, m\}$, let α_i and β_i be inputs of the ith multiplication gate, and γ_i its output. Let α,β and γ as defined in the multiplication protocol. It is not necessary to commit to α and β as a commitment of them can be obtained from affine forms on inputs $(\mathbf{x}, \gamma)$ which are only dependent on C. Similarly, the output of C can be computed from an affine map $\omega : \mathbb{Z}_p^{k+2m} \to \mathbb{Z}_p^s$ that takes as input $(\mathbf{x}, \gamma)$. When committing to $\mathbf{x}$ and γ, the prover only needs to prove that multiplication gates hold and that ω opens to 0. This can be done with amortized nullity checks. Let $[\mathbf{a}]$ be a hiding vector commitment of $\mathbf{a}$ i.e., $[\mathbf{a}] = \mathbf{g}^{(\mathbf{a}, r)}$ where $r \in \mathbb{Z}_p$ is chosen at random.

Protocol $\Pi_{cs}(C)$**:**

(1) $\mathcal{P}$: Computes f, g and h from α, β as in Steps 1 and 2 of Π_M.

(2) $\mathcal{P}$ sends to $\mathcal{V}$: $[\mathbf{y}]$, where

$$\mathbf{y} = (\mathbf{x}, f(0), g(0), h(0), h(1), \ldots, h(2m)) \in \mathbb{Z}_p^{k+2m+3}.$$

(3) $\mathcal{V}$ sends to $\mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p \setminus \{1, \ldots, m\}$.
(4) $\mathcal{P}$ sends to $\mathcal{V}$: $z_1 = f(c)$, $z_2 = f(c)$, $z_3 = f(c)$.
(5) $\mathcal{P}$ and $\mathcal{V}$ run Π_N with input

$$([\mathbf{y}], (\omega, f(c) - z_1, g(c) - z_2, h(c) - z_3); \mathbf{y}),$$

where linear forms are obtained by Lagrange interpolation as in Step 5 of Π_M.
(6) $\mathcal{V}$: if $z_1 z_2 = z_3$ then **accept** else **reject**

Note that $[\mathbf{y}]$, additionally to α and β, is an implicit commitment to $\gamma = (h(1), \ldots, h(m))$. Properties of completeness, zero knowledge and soundness, which can be found in Theorem 4 of [2], follow from the same arguments as the multiplication protocol.

A.5 Range Proofs

A straightforward application of circuit proofs are range proofs. Namely, for a secret $x \in \mathbb{Z}_p$ and an integer $k < \log_2(p)$, that $x \in [0, 2^k)$. For that you commit to the first k bits $b_1, \ldots, b_k$ of x. Then, for $\mathbf{b} = (b_1, \ldots, b_k)$, a circuit proof for circuit

$$C_{Ra}(x, \mathbf{b}) := \begin{bmatrix} \mathbf{b} * (1 - \mathbf{b}) \\ x - \sum_{i=1}^{k} 2^{i-1} b_i \end{bmatrix}$$

implies the range constraint. Note that in C_{Ra} the output of the multiplication gates is 0. Therefore, in protocol Π_{cs}, it is not necessary to include in $\mathbf{y}$ the elements $h(1), \ldots, h(k)$, which reduces the proof cost.

A.6 Cost of Proofs

We now briefly summarize communication and computational cost of the proofs. As previously stated in the Appendix, k is the number of inputs and m of multiplication gates in circuits.

Theorems 3 and 4 of [2] give the communication costs for Π_c and Π_{cs} respectively. As we use versions of the protocols transformed by the Fiat-Shamir heuristic, the verifier does not send any message. Therefore, the proof sizes are $2\lceil \log(k+1) \rceil$ elements of $\mathbb{G}$ and 3 elements of $\mathbb{Z}_p$ for Π_c, and $2\lceil \log(k+2m+4) \rceil - 1$ elements of $\mathbb{G}$ and 6 elements of $\mathbb{Z}_p$ for Π_{cs}.

For the computational cost, we count the amount of group exponentiations in $\mathbb{G}$ (GEX) as they dominate the work. In protocol Π_0, $\mathcal{P}$ performs k GEX to compute A, and $\mathcal{V}$ performs $k+1$ GEX, which corresponds to the verification of Step 5. In Π_1, $\mathcal{P}$ performs $k+2$ GEX to compute A and B, and $\mathcal{V}$ does $k/2 + 4$ GEX in the verification of Step 4. Π_c is a composition of one instance of Π_0 and $\mu = \lceil \log_2(k) \rceil - 2$ instances of Π_1. After the first Π_1 proof, k halves at each instance of Π_1. Additionally, $\mathcal{P}$ and $\mathcal{V}$ have to compute $\mathbf{g}' = \mathbf{g}_L^c + \mathbf{g}_R$ after the first Π_1 to update parameters for each of following sub-protocols. $\mathcal{V}$ avoids each of the verification checks except for the last one, which requires a constant amount of GEX. Therefore, $\mathcal{P}$ performs $k + 2 + \sum_{i=1}^{\mu} \frac{k}{2^{i-1}} + \frac{k}{2^i} = 4k + 2\mu - 10$ GEX and $\mathcal{V}$ does $3 + \sum_{i=1}^{\mu} \frac{k}{2^i} + 2 = k + 2\mu - 1$ GEX. Protocol Π_N requires the same amount of GEX than Π_c.

In Π_{cs}, $\mathcal{P}$ is required to compute a (hiding) commitment of $\mathbf{y} \in \mathbb{Z}_p^{k+2m+3}$, which costs $l = k + 2m + 4$ GEX. Then $\mathcal{P}$ and $\mathcal{V}$ engage in Π_N for an affine form of l inputs. The final costs for Π_{cs} are then $5k + 8m + 2\lceil \log_2(k + 2m + 4) \rceil + 6$ GEX for $\mathcal{P}$ and $k + 2m + 2\lceil \log_2(k + 2m + 4) \rceil - 1$ GEX for $\mathcal{V}$. For a proof of membership to the range $[0, 2^k)$, as discussed in Appendix A.5, $h(1), \ldots, h(k)$ are not included in $\mathbf{y}$ which then has $2k + 4$ elements. The costs are of $9k + 2\lceil \log_2(2k + 5) \rceil + 11$ GEX for $\mathcal{P}$, and $2k + 2\lceil \log_2(2k + 5) \rceil$ GEX for $\mathcal{V}$.

We apply the same optimization done for range proofs to all of our circuits. That is, multiplication gates that are expected to be equal to 0 are not included in $\mathbf{y}$. Therefore, for circuits with k inputs, m multiplication gates, and m_0 multiplication gates that will be equal to 0,

- $\mathcal{P}$ performs $5k + 8m - 4m_0 + 2\lceil \log_2(k + 2m - m_0 + 4) \rceil + 6$ GEX
- $\mathcal{V}$ performs $k + 2m - m_0 + 2\lceil \log_2(k + 2m - m_0 + 4) \rceil - 1$ GEX.

B SECURITY OF OUR PROTOCOLS

In this section, we prove the security of the protocols presented in the main paper. In the sequel, we will often restate in more detail and more formally the protocols. We consider a set of n parties $\mathbf{P} = \{P_1 \ldots P_n\}$. We will denote by P_{-i} the set of all parties except i, i.e., $P_{-i} = \mathbf{P} \setminus \{P_i\}$. We assume that a subset of parties $\mathbf{P}_{cor} \subset \mathbf{P}$ is corrupted and controlled by an adversary $\mathcal{A}$. The set $\mathbf{P}_{cor}$ of corrupted parties is static, i.e. does not change after the beginning of the execution.

For the description of our protocols, we will denote the fact that a party A sends a message M to a party B by "$A \rightarrow B$: M". Recall that we use a bulletin board for communication. Among others, this means that when a protocol contains a broadcast instruction a single message is sent from one party to all others, in practice by sending it from that party to the bulletin board, which forwards it to all other agents. We will also use hiding vector Pedersen commitments defined in Section 2.2. Recall the finite groups $\mathbb{Z}_p$ and $\mathbb{G}$ defined therein. For any integer $k > 0$, we will denote by $Com(\mathbf{x}; r) \in \mathbb{G}$ the commitment of the k-dimension vector $\mathbf{x} \in \mathbb{Z}_p^k$ with randomness $r \in \mathbb{Z}_p$.

We describe our security framework in Appendix B.1. In Appendix B.2, we describe our model of compressed Σ-protocols in our security analysis. In appendices B.3 and B.4 we prove the security of Protocols 1 and 2 respectively. We conclude by discussing the security of our protocols for public and private draws from some other distributions in Appendix B.5.

B.1 Security Definitions

We prove security in the simulation paradigm, using the model of *malicious security with identifiable abort* [33] in the stand-alone setting [29]. Therefore, we assume parties are able to detect malicious actions and can in such cases abort the protocol. Deterrence measures may be in place to discourage parties from being detected as malicious. In fact, unless parties stop participating our protocols either complete successfully or abort while detecting that a specific party is a cheater.

We start by introducing the key concepts of multiparty computation under the model of security with identifiable abort in the stand-alone setting (see [33, App. B of the full version]). A multiparty computation between our n parties in $\mathbf{P}$ is a protocol that computes a stochastic process $\mathcal{F}: (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, which

is also called an *ideal functionality*, where for all $i \in \{1, \ldots, n\}$, the ith component of the input and output of $\mathcal{F}$ maps respectively to the private input and output of party P_i.

In the simulation paradigm, a multiparty protocol securely computes an ideal functionality in the presence of malicious adversaries if any possible malicious behavior in the protocol caused by colluding malicious parties is not more harmful than what such party can cause in the *ideal model* as defined below.

Ideal Model. In this model, it is assumed that there exists a trusted party that computes $\mathcal{F}$. A malicious adversary $\mathcal{S}$ controls a set of corrupted parties $\mathbf{P}_{cor}$. The ideal execution comprises 7 phases and goes as follows:

(1) **Inputs:** Each party P_i receives a private input u_i. Additionally, $\mathcal{S}$ has an auxiliary input u^* which represents the extra knowledge other than its regular input.
(2) **Send inputs to the trusted party:** All honest parties send their input to the trusted party, while parties controlled by $\mathcal{S}$ might deviate and send what $\mathcal{S}$ wishes.
(3) **Early abort or corrupted input:** $\mathcal{S}$ can send $\texttt{abort}_i$ instead of a valid input, which means that some corrupted party P_i performed an early abortion or sent a corrupted input. In that case, the trusted party sends $\texttt{abort}_i$ (choosing the index i deterministically if many parties aborted or sent a corrupted input) to all honest parties and halts.
(4) **Detect cheating parties:** During the execution, $\mathcal{S}$ can send $\texttt{abort}_i$ to the trusted party, which means that a corrupted party P_i attempted to cheat. In that case, the trusted party sends $\texttt{abort}_i$ to all parties (i.e. the attempt of cheating is detected).
(5) **Trusted party answers the adversary:** If no $\texttt{abort}_i$ is sent, then the trusted party sends to $\mathcal{S}$ the outputs of $\mathcal{F}$ of the corrupted parties. After receiving them, $\mathcal{S}$ can send $\texttt{abort}_i$ to the trusted party or instruct it to continue.
(6) **Trusted party answers the honest parties:** If $\mathcal{S}$ instructed the trusted party to continue, then the latter sends their outputs of $\mathcal{F}$ to the honest parties.
(7) **Output:** The honest parties always output what the trusted party sent to them. $\mathcal{S}$ outputs any arbitrary computable function of the inputs $\{u_i\}_{i \in \mathbf{P}_{cor}}$, the auxiliary input u^* and the messages obtained from the trusted party.

Let $\bar{u} = (u_1 \ldots u_n)$ be the vector of inputs of all parties and u^* the auxiliary input of $\mathcal{S}$. Recall that λ is the security parameter. We denote by $\text{IDEAL}_{\mathcal{F},\mathcal{S}(u^*),\mathbf{P}_{cor}}(\bar{u}, \lambda)$ to the vector of outputs of parties in the above execution.

The Real Model. The real model of an n-party protocol Π describes its execution in the presence of non-uniform probabilistic polynomial time advesary $\mathcal{A}$ that corrupts a set of parties $\mathbf{P}_{cor}$. Parties in $\mathbf{P} \setminus \mathbf{P}_{cor}$ behave as described by Π. We denote by $\text{REAL}_{\Pi,\mathcal{A}(u^*)}(\bar{u}, \lambda)$ the vector of outputs of parties in the real execution of Π with input $\bar{u}$ and where $\mathcal{A}$ has auxiliary input u^*.

We formalize the notion of security with identifiable abort [33, Def. 16 of the full version] below.

Definition 3 (Security with identifiable abort in the stand-alone setting). Let λ be the security parameter and $\mathcal{F} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party ideal functionality. A protocol Π *securely computes $\mathcal{F}$ with identifiable abort* if for every non-uniform probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real model, there exist a non-uniform PPT adversary $\mathcal{S}$ in the ideal model such that the distributions of

$$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(u^*),\mathbf{P}_{cor}}(\bar{u}, \lambda)\}_{\bar{u}, u^* \in (\{0,1\}^*)^{n+1}, \lambda \in \mathbb{N}}$$

and

$$\{\text{REAL}_{\Pi,\mathcal{A}(u^*),\mathbf{P}_{cor}}(\bar{u}, \lambda)\}_{\bar{u}, u^* \in (\{0,1\}^*)^{n+1}, \lambda \in \mathbb{N}}$$

are computationally indistinguishable.

Hybrid Model. In addition to the security definition, the simulation paradigm facilitates tools to prove security of protocols which use sub-protocols already known to be secure. Given functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_k$, where k is polynomial in λ, it allows to define a protocol in which both parties can send messages to each other and place "ideal calls" to a trusted party that computes $\mathcal{F}_i$ for $i \in \{1, \ldots, k\}$. In these ideal calls, parties can send their input to the trusted party and wait for the output of $\mathcal{F}_i$. However, (1) they cannot send any messages between each other after invoking an ideal functionality and before its response is returned by the trusted party and (2) functionalities cannot be called concurrently. In other words, functionalities can only be *sequentially composed* with all other interactions. We denote such model as the $(\mathcal{F}_1, \ldots, \mathcal{F}_k)$*-hybrid model.* When we describe a protocol in the $(\mathcal{F}_1, \ldots, \mathcal{F}_k)$-hybrid model, we say that $\mathcal{F}_1, \ldots, \mathcal{F}_k$ are *hybrid functionalities.*

Sequential Composition. Consider the protocol Π and functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_k$ as defined above and let $\rho_1, \ldots, \rho_k$ be protocols. We define the protocol $\Pi^{\rho_1,\ldots,\rho_k}$ to be the protocol in the ideal model that behaves exactly as Π in real messages, but for all $i \in \{1, \ldots, k\}$ each ideal call to $\mathcal{F}_i$ is replaced by the execution of protocol ρ_i. We now state conditions such in which sequential composition is secure.

Theorem 4 (Sequential composition [29]). *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be secure multiparty functionalities and let $\rho_1, \ldots, \rho_k$ be protocols that securely compute $\mathcal{F}_1, \ldots, \mathcal{F}_k$ respectively with identifiable abort. Let $\mathcal{G}$ be a multiparty functionality and let Π be a multiparty protocol that securely computes $\mathcal{G}$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_k)$-hybrid model with identifiable abort. Then $\Pi^{\rho_1,\ldots,\rho_k}$ securely computes $\mathcal{G}$ in the real model with identifiable abort.*

We note that the model of [33] is defined for more general types of composition. Nevertheless, it is compatible with the definition described above.

Assumptions on Adversaries. We can see $\mathcal{A}$ as a deterministic algorithm with a special input which is a random tape of uniformly distributed bits. In the ideal model, we can also rewind $\mathcal{A}$ to a previous state in execution, where the random tape also rewinds.

B.2 Compressed Σ-Protocols as Ideal Functionalities

We will use compressed Σ-protocols implemented with the Fiat-Shamir heuristic which are proven secure in the random oracle model [48]. They have been proven secure against (by definition) malicious provers and there is no interaction with malicious verifiers. Therefore we can consider them as secure and abstract them as hybrid functionalities. In particular, when we write $P \to \mathcal{F}_\Sigma^R$:

$(x; w)$; $\mathcal{F}_\Sigma^R \to O : [b, x']$, we mean that $\mathcal{F}_\Sigma^R$ gets as input from party P the public data x and secret witness w, gets from all other relevant parties as input the empty string, and returns to all parties in O the same pair $[b, x']$ where x' is the data provided as input by P and b is 1 if $(x; w) \in R$ (the proof succeeds) and 0 otherwise.

B.3 Proof of Protocol 1

Let U be the random variable uniformly distributed over the interval $[0, L)$ for $L \leq p$. We consider the *ideal functionality*

$$\mathcal{F}_{p1}(\{\} \dots \{\}) = (U \dots U),$$

i.e., $\mathcal{F}_{p1}$ takes as input from each of the n parties an empty string and outputs to every party the same uniformly distributed U.

Protocol. The protocol Π_{p1} is explained below.

Protocol Π_{p1}:
Security Parameter: λ
Hybrid Functionality sub-protocols: Functionalities $\mathcal{F}_\Sigma^{R_1}$ and $\mathcal{F}_\Sigma^{R_2}$ are zero knowledge proofs of respectively

- $R_1 = \{(C; x, r) : C = Com(x, r)\}$
- $R_2 = \{(C, x; r) : C = Com(x, r)\}$ (only r is secret),

Protocol:

(1) For $i = 1 \dots n$:
- P_i : choose $x_i \in [0, L), r_i \in \mathbb{Z}_p$ at random
- P_i : compute the commitment $C_i = Com(x_i, r_i)$
- P_i : broadcast C_i
- $P_i \to \mathcal{F}_\Sigma^{R_1} : (C_i; x_i, r_i)$
- $\mathcal{F}_\Sigma^{R_1}$: broadcast $[b_i, C_i']$
- P_{-i}: if $b_i \neq 1$ or $C_i' \neq C_i$, detect P_i as a cheater and abort

(2) For $i = 1 \dots n$:
- $P_i \to \mathcal{F}_\Sigma^{R_2} : (C_i, x_i; r_i)$
- $\mathcal{F}_\Sigma^{R_2}$: broadcast $[b_i', C_i'', x_i']$
- P_{-i}: if $x_i \notin [0, L)$, $b_i' \neq 1$ or $C_i'' \neq C_i$, detect P_i as a cheater and abort

(3) For $i = 1 \dots n$:
- **output** $\sum_{i=1}^n x_i' \bmod L$.

We state the security of protocol Π_{p1} in the theorem below.

Theorem 5 (Security of Π_{p1}). *Let Com be a computationally binding and perfectly hiding commitment scheme and let $\mathcal{F}_\Sigma^{R_1}$ and $\mathcal{F}_\Sigma^{R_2}$ be secure multiparty functionalities of computationally sound zero-knowledge proofs of relations R_1 and R_2 respectively. Then, Protocol Π_{p1} securely computes $\mathcal{F}_{p1}$ in the $(\mathcal{F}_\Sigma^{R_1}, \mathcal{F}_\Sigma^{R_2})$-hybrid model with identifiable abort if at least one party is honest.*

Proof. It is clear that Π_{p1} securely computes $\mathcal{F}_{p1}$ in the honest-but-curious setting.

Below, we will use $\mathcal{A}$ to denote a non-uniform probabilistic polynomial-time (PPT) adversary that controls covert parties.

In Figure 4, we define four very similar algorithms $\mathcal{S}^v$, $v \in \{0, 1, 2, 3\}$ where $\mathcal{S}^0$ is a simulator and for which we will prove that their ideal execution output is indistinguishable from the hybrid execution output. Each simulator $\mathcal{S}^v$ will internally run a copy of $\mathcal{A}$ which we will denote by $\mathcal{A}^v$. Without loss of generality, we can assume that at the points where all parties send messages, the messages of the honest parties arrive first, as everything an adversary can infer from the messages of a subset of honest parties it can also infer from the messages of all honest parties.

To see that the simulation of $\mathcal{S}^0$ is indistinguishable from the $(\mathcal{F}_\Sigma^{R_1}, \mathcal{F}_\Sigma^{R_2})$-hybrid model with adversary $\mathcal{A}$, we define our simulators such that

- The outputs of $\mathcal{S}^0$ and $\mathcal{S}^1$ are identically distributed, because their only difference is the moment on which z is chosen uniformly at random (and hence independently from other variables).
- The outputs of $\mathcal{S}^1$ and $\mathcal{S}^2$ are identically distributed, the only difference is that $\mathcal{S}^1$ first draws z in line 28 and then computes $x_{i'}$ from it, while $\mathcal{S}^2$ first draws $x_{i'}$ and then computes z from it.
- The outputs of $\mathcal{S}^2$ and $\mathcal{S}^3$ are indistinguishable. All inputs to $\mathcal{A}$ are the same, except for a commitment $Com(0, r_{i'})$ versus a commitment $Com(x_{i'}, r_{i'})$. As the commitment scheme is hiding and $r_{i'}$ is chosen independetly, the distributions of these commitments are indistinguishable and for any $\mathcal{A}^3$ getting input $Com(x_{i'}, r_{i'})$ there is an $\mathcal{A}^2$ producing a computationally indistinguishable output.
- The output of $\mathcal{S}^3$ is identically distributed as the output of Π_{p1} in the hybrid model.

□

Vectorized Version. We can now consider the ideal functionality

$$\mathcal{F}_{p1}^{(k)}(\{\} \dots \{\}) = ((U_1 \dots U_k) \dots (U_1 \dots U_k)),$$

i.e., $\mathcal{F}_{p1}^{(k)}$ takes as input from each of the n parties an empty string and outputs to every party the same vector $(U_1 \dots U_k)$ where every U_i is uniformly distributed in $[0, L)$ independently for $i = 1 \dots k$. We can then similarly construct a protocol $\Pi_{p1}^{(k)}$ that draws a vector z uniformly distributed in $[0, L)^k$.

In this protocol we will use a pseudo-random number generator as we defined in 2.6. We use a similar but equivalent definition which is more convenient in our proof, in particular, for some polynomial p, this is a function $G : \{0, 1\}^q \to [0, L)^{p(q)}$ such that for any randomized polynomial time algorithm $A : [0, L)^{p(q)} \to \{0, 1\}$ there holds that $|P_{x \leftarrow_R \{0,1\}^q}(A(G(x)) = 1) - P_{x \leftarrow_R [0,L)^{p(q)}}(A(x) = 1)| \leq \mu(q)$ with μ a negligible function. We choose q sufficiently large such that $p(q) \geq k\lceil \log(L) \rceil$ and $\mu(q)$ is sufficiently small.

We describe this protocol in the hybrid model using the ideal functionality $\mathcal{F}_{p1[0,2^q)}$, which is a variant on the $\mathcal{F}_{p1}$ functionality we introduced above where L is set to 2^q. The protocol $\Pi_{p1}^{(k)}$ is described below:

1: **Protocol** $\Pi_{p1}^{(k)}$
2: **Security parameter:** λ
3: **Hybrid functionality sub-protocols:**
- $\mathcal{F}_{p1[0,2^q)}$: draws a single public random number in $[0, 2^q)$.

4: **Protocol:**
5: All parties collaboratively perform:
6: $z' = \mathcal{F}_{p1[0,2^q)}(\{\} \dots \{\})$

Phase 1: choosing z (see also phase 4)
  if $v = 0$ **then**
    $\mathcal{F}_{p1} \rightarrow \mathcal{S}^0 : z$ ($\mathcal{S}^0$ invokes trusted party delivering ideal functionality)
  end if
Phase 2: $\mathcal{S}^v$ simulates honest parties:
  $\mathcal{S}^v : i' = \max\{i \mid P_i \in \mathbf{P} \setminus \mathbf{P}_{cor}\}$
  for $P_i \in (\mathbf{P} \setminus \mathbf{P}_{cor}) \setminus \{P_{i'}\}$ **do**
    $\mathcal{S}^v$: choose x_i randomly in $[0, L)$ and r_i randomly in $\mathbb{Z}_p$
    $\mathcal{S}^v$: $C_i = Com(x_i, r_i)$; $[b_i, C_i'] = \mathcal{F}_{\sum}^{R_1}(C_i; x_i, r_i)$
    $\mathcal{S}^v \rightarrow \mathcal{A}^v : [b_i, C_i']$ (as if it is sent by P_i)
  end for
  if $v \in \{0, 1, 2\}$ **then**
    draw $r_{i'}$ randomly; $C_{i'} = Com(0, r_{i'})$; $[b_{i'}, C_{i'}'] = [1, C_{i'}]$
  else
    draw $x_{i'}$ and $r_{i'}$ randomly; $C_{i'} = Com(x_{i'}; r_{i'})$; $[b_{i'}, C_{i'}'] = \mathcal{F}_{\sum}^{R_1}(C_{i'}; x_{i'}, r_{i'})$
  end if
  $\mathcal{S}^v \rightarrow \mathcal{A}^v : [1, C_{i'}']$ (as if it is the answer of $\mathcal{F}_{\sum}^{R_1}(C_{i'}; x_{i'}, r_{i'})$)
Phase 3: $\mathcal{S}^v$ simulates $\mathcal{A}^v$:
  for $i \in \mathbf{P}_{cor}$ **do**
    $\mathcal{A}^v \rightarrow \mathcal{S}^v : C_i$
    $\mathcal{A}^v \rightarrow \mathcal{F}_{\sum}^{R_1} : (C_i; x_i, r_i)$; $\mathcal{S}^v$ records C_i, x_i and r_i.
    $\mathcal{F}_{\sum}^{R_1} \rightarrow \mathcal{S}^v : [b_i, C_i']$ (as if sent to members of $\mathbf{P} \setminus \mathbf{P}_{cor}$)
    If $C_i' \neq C_i$ or $C_i \neq Com(x_i, r_i)$ detect P_i as cheater
  end for
Phase 4: choosing z and $x_{i'}$
  if $v \in \{0, 1\}$ **then**
    if $v = 1$ **then**
      $\mathcal{S}^1$ chooses z uniformly at random
    end if
    $x_{i'} = z - \sum_{i \neq i'} x_i \bmod L$
  else (i.e., if $v \in \{2, 3\}$)
    draw $x_{i'}$ uniformly at random and set $z = \sum_{i \in \mathbf{P}} x_i \bmod L$
  end if
Phase 5:
  for $P_i \in \mathbf{P} \setminus \mathbf{P}_{cor}$ **do**
    $\mathcal{S}^v$: set $x_i' = x_i$; $C_i'' = C_i$; $b_i' = 1$
    $\mathcal{S}^v \rightarrow \mathcal{A}^v$: $[b_i', C_i'', x_i']$ (as if it is sent by $\mathcal{F}_{\sum}^{R_2}$ in answer to $(x_i, C_i; r_i)$)
  end for
Phase 6: $\mathcal{S}^v$ continues simulation of $\mathcal{A}^v$
  for $P_i \in \mathbf{P}_{cor}$ **do**
    $\mathcal{A}^v \rightarrow \mathcal{F}_{\sum}^{R2} : (C_i'', x_i'; r_i')$; $\mathcal{S}^v$ records C_i'', x_i' and r_i'
    $\mathcal{F}_{\sum}^{R2} \rightarrow \mathcal{S}^v : [b_i', C_i'', x_i']$
    If $x_i' \neq x_i$ or $C_i'' \neq C_i$ or $b_i' = 0$ detect P_i as cheater
  end for
Phase 7: Output
  $\mathcal{A}^v \rightarrow \mathcal{S}^v : z'$ (the output of $\mathcal{A}^v$)
  output z'

Figure 4: Simulators $\mathcal{S}^v$, for $v \in \{0, 1, 2, 3\}$.

For $P_i \in \mathbf{P}$:
  P_i: Output $z = (z_1 \ldots z_k)$ with z_j containing bits

$$(j-1)\lceil \log(L) \rceil + 1 \ldots j\lceil \log(L) \rceil$$

of $G(z')$ for $j = 1 \ldots k$

Theorem 6 (Security of $\Pi_{p1}^{(k)}$). *Let $\mathcal{F}_{p1[0,2^q)}$ be a secure multiparty functionality and G be a PRG as defined above. Then, Protocol $\Pi_{p1}^{(k)}$ securely computes $\mathcal{F}_{p1}^{(k)}$ in the $\mathcal{F}_{p1[0,2^q)}$-hybrid model with identifiable abort if at least one party is honest.*

Proof. It follows from the definition of random number generator that the output of the protocol is indistinguishable from the output of the ideal functionality when all parties are honest-but-curious.

The protocol consists of first a call to a secure sub-protocol and then a public computation which each agent can do for himself without any interaction. It is hence easy to see that the protocol is secure. □

B.4 Proof of Protocol 2

Now we prove the security of Protocol 2, which performs a private uniform draw. Without loss of generality, we assume that the drawn sample should be private to P_1. The ideal functionality $\mathcal{F}_{p2} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ is defined as

$$\mathcal{F}_{p2}(\{\} \dots \{\}) = ((z, r_z), C_z, \dots, C_z)$$

where $C_z = Com(z, r_z)$, z is uniformly randomly distributed in $[0, L)^k$ and r_z is uniformly randomly distributed in $\mathbb{Z}_p$ (as a consequence of the distributions of z and r_z, C_z is uniformly distributed over $\mathbb{G}$). The pair (z, r_z) is private to P_1 while C_z is known by all parties. We will describe a detailed version of Protocol 2 in the hybrid model. Our hybrid functionalities will be $\mathcal{F}_{p1}$ and $\mathcal{F}_{p1}^{(k)}$, which correspond to Protocol 1, and $\mathcal{F}_{\Sigma}^{R_m}$, which, for some modulus $L \leq p$ is the ideal functionality of a zero knowledge proof for relation

$$\begin{aligned} R_m = \{(y \in [0,L)^k, r_y \in \mathbb{Z}_p, C_x \in \mathbb{G}, C_z \in \mathbb{G}; \\ x \in [0,L)^k, r_x \in \mathbb{Z}_p, z \in [0,L)^k, r_z \in \mathbb{Z}_p) : \\ C_x = Com(x, r_x) \wedge C_z = Com(z, r_z) \wedge x \in [0, L) \\ \wedge z = x + y \bmod L \wedge r_z = r_x + r_y \bmod \mathbb{Z}_p\}. \end{aligned}$$

Such proof can be performed by applying the techiques decribed in Section 6.1. In Figure 5, we define the protocol Π_{p2} which describes Protocol 2 in more detail.

Theorem 7. *Let Com be a computationally binding and perfectly hiding commitment scheme and let $\mathcal{F}_{\Sigma}^{R_m}$ be a secure multiparty functionality of a computationally sound zero-knowledge proof of relation R_m. Let $\mathcal{F}_{p1}^{(k)}$ and $\mathcal{F}_{p1[0,2^q)}$ be secure multiparty functionalities as defined above. Then, protocol Π_{p2} securely computes $\mathcal{F}_{p2}$ in the $(\mathcal{F}_{p1}^{(k)}, \mathcal{F}_{p1\mathbb{Z}_p}, \mathcal{F}_{\Sigma}^{R_m})$-hybrid model with identifiable abort if at least one party is honest.*

Proof. First note that, except for P_1, the other parties are only supposed to perform ideal calls and do not interact with each other. We will consider first the most difficult case in which P_1 is among the corrupted parties controlled by $\mathcal{A}$. In Figure 6, we define our adversary $\mathcal{S}$ that simulates the output of $\mathcal{A}$ in the ideal model.

We show that the ideal and hybrid execution outputs are indistinguishable. We first start by analyzing the view of $\mathcal{A}$ in the hybrid protocol and when interacting with $\mathcal{S}$.

- Since $\mathcal{A}$ has not seen any other message, the first commitment C_x of $\mathcal{A}$ is the same in both hybrid and ideal executions
- y and r_y have the same distribution in the hybrid and ideal model as they are part of an ideal call. Therefore, also the C_z broadcasted by $\mathcal{A}$ follows the same distribution in the hybrid and ideal model.
- if $\mathcal{A}$ cheats in C_z (i.e., C_z is not a commitment according to relation R_m), it gets caught both in the hybrid and ideal executions
- before rewinding $\mathcal{A}$, $\mathcal{S}$ recovers x and r_x such that $C_x = Com(x, r_x)$ with overwhelming probability (this is because $\mathcal{A}$ cannot fake the zero knowledge proof in $\mathcal{F}_{\Sigma}^{R_m}$ except with negligible probability)
- in the ideal world, $\mathcal{S}$ sets y and r_y such that $z = x + y \bmod L$ and $r_z = r_x + r_y \bmod p$, where x and r_x are chosen by $\mathcal{A}$, z and y are uniformly randomly distributed in $[0, L)^k$ and r and r_y are uniformly randomly distributed in $\mathbb{Z}_p$. This is the same distribution as in the hybrid world.
- now $\mathcal{A}$ broadcast C_z'' and either is detected as cheater or $C_z'' = Com(z, r_z)$ with overwhelming probability both in the hybrid or ideal executions

Up to this point, from the view of $\mathcal{A}$ the transcripts simulated by $\mathcal{S}$ in the ideal execution and the transcript in the hybrid execution are indistinguishable. Since $\mathcal{A}$ runs in polynomial time, his output must be indistinguishable in both executions.

Now we analyze the case where P_1 is among then honest parties. Particularly, consider the worst case where all other parties are malicious and P_1 is the only honest party. This case is still easy since the only interaction of malicious users is to perform ideal calls to $\mathcal{F}_{p1}^{(k)}$, $\mathcal{F}_{p1\mathbb{Z}_p}$ and $\mathcal{F}_{\Sigma}^{R_m}$. The only role of $\mathcal{S}$ is to send inputs that might be consistent according to the behavior of $\mathcal{A}$. Even if all the verifiers are malicious, the honest party can still detect them. □

B.5 Non-Uniform Public and Private Draws

Public and Private draws from many distributions such as Gaussians or Laplace distributions can be performed by first drawing uniformly distributed samples with either Protocol 1 for public draws or Protocol 2 for private draws, then applying a non-interactive transformation. For private draws, this transformation is a non-interactive compressed Σ-protocol as described in Appendix B.2. For each technique, we explain the appropriate transformation in Sections 7.1, 8.1, 8.2, 8.3 and 8.4. This procedure is secure as it is essentially a secure drawing of a uniformly distributed random number as discussed in the previous sections followed by a private but verifiable single-party post-processing.

Protocol Π_{p2} (to generate single private sample):
Security Parameter: λ
Hybrid Functionalitiy sub-protocols:

- $\mathcal{F}_{p1}^{(k)}$ draws publicly a random number from $[0, L)^k$.
- $\mathcal{F}_{p1\mathbb{Z}_p}$ is a variant of $\mathcal{F}_{p1}$ that draws a random number from $\mathbb{Z}_p$ rather than from $[0, L)$.
- $\mathcal{F}_{\Sigma}^{R_m}$ performs a zero-knowledge proof for the relation R_m defined above

Protocol:
P_1:
    draw $x \in [0, L)^k$ and $r_x \in \mathbb{Z}_p$ uniformly at random
    compute $C_x = Com(x, r_x)$
    broadcast C_x
All parties in **P** collaboratively:
    call $\mathcal{F}_{p1}^{(k)}$ to obtain a public y uniformly distributed over $[0, L)^k$
    call $\mathcal{F}_{p1\mathbb{Z}_p}$ to obtain a public r_y uniformly distributed over $\mathbb{Z}_p$
P_1:
    Compute $z = x + y \bmod L$ and $r_z = r_x + r_y \bmod p$
    Compute $C_z = Com(z, r_z)$
    broadcast C_z
Parties perform an ideal call to $\mathcal{F}_{\Sigma}^{R_m}$:
    $P_1 \rightarrow \mathcal{F}_{\Sigma}^{R_m} : (y, r_y, C_x, C_z; x, r_x, z, r_z)$
    $\mathcal{F}_{\Sigma}^{R_m}$: broadcast $[b, y', r'_y, C'_x, C'_z]$
    P_{-1}: if $b = 0 \vee r_u \neq r'_y \vee y \neq y' \vee C_x \neq C'_x \vee C_Z \neq C'_z$, detect P_1 as a cheater and abort, otherwise continue the execution
P_1: **output** (z, r_z)
P_{-1}: **output** C_z

Figure 5: Protocol Π_{p2}, a detailed description of Protocol 2.

1: $\mathcal{S}$: internally run $\mathcal{A}$ until broadcast C_x (line 8 in Π_{p2})
2: $\mathcal{A} \to \mathcal{S} : C_x$
3: $\mathcal{S}$: continue the run of $\mathcal{A}$ until after the ideal call to $\mathcal{F}_{p1}^{(k)}$ (line 10 in Π_{p2})
4: $\mathcal{S}$: draw a random value $y \in [0, L)^k$
5: $\mathcal{S} \to \mathcal{A} : y$ (as if it came from $\mathcal{F}_{p1}^{(k)}$)
6: $\mathcal{S}$: continue the run of $\mathcal{A}$ until after the ideal call to $\mathcal{F}_{p1\mathbb{Z}_p}$ (line 11 in Π_{p2})
7: $\mathcal{S}$: draw a random value $r_y \in \mathbb{Z}_p$
8: $\mathcal{S} \to \mathcal{A} : r_y$ (as if it came from $\mathcal{F}_{p1\mathbb{Z}_p}$)
9: $\mathcal{A} \to \mathcal{S} : C_z$
10: $\mathcal{S}$: continue run of $\mathcal{A}$: $P_1 \to \mathcal{F}_{\Sigma}^{R_m} : (y', r'_y, C'_x, C'_z, x', r'_x, z', r'_z)$
11: $\mathcal{F}_{\Sigma}^{R_m}$: broadcast $[b, y', r'_y, C'_x, C'_z]$
12: if $b = 0 \vee y' \neq y \vee r'_y \neq r_y \vee C'_x \neq C_x \vee C'_z \neq C_z$, detect P_1 as a cheater
13: $\mathcal{S}$: invoke trusted party computing $\mathcal{F}_{p2}$
14: $\mathcal{F}_{p2}$ returns $((z, r_z) \in [0, L)^k \times \mathbb{Z}_p, (C_z, \ldots, C_z) \in \mathbb{G}^{|\mathbf{P}_{cor}|-1})$ to $\mathcal{S}$ and $(C_z, \ldots, C_z) \in \mathbb{G}^{|\mathbf{P}\setminus\mathbf{P}_{cor}|}$ to the honest parties
15: $\mathcal{S}$ sets $y = z - x \mod L$ and $r_y = r_z - r_x \mod p$
16: $\mathcal{S}$ rewinds $\mathcal{A}$ to before the invocation of $\mathcal{F}_{p1}^{(k)}$
17: $\mathcal{S}$ continues the internal run of $\mathcal{A}$, which performs ideal calls to $\mathcal{F}_{p1}^{(k)}$ and $\mathcal{F}_{p1\mathbb{Z}_p}$.
18: $\mathcal{S} \to \mathcal{A} : y, r_y$ (as if they were sent by $\mathcal{F}_{p1}^{(k)}$ and $\mathcal{F}_{p1\mathbb{Z}_p}$)
19: $\mathcal{A} \to \mathcal{S} : C''_z$; if $C''_z \neq C_z$, detect P_1 as a cheater
20: $\mathcal{S}$ continues to run $\mathcal{A}$: $P_1 \to \mathcal{F}_{\Sigma}^{R_m} : (y'', r''_y, C''_x, C''_z, x'', r''_x, z'', r''_z)$
21: $\mathcal{F}_{\Sigma}^{R_m}$: broadcast $[b', y'', r''_y, C''_x, C''_z]$
22: if $b' = 0 \vee y'' \neq y \vee r''_y \neq r_y \vee C''_x \neq C_x \vee C''_z \neq C_z$, detect P_1 as a cheater
23: output whatever $\mathcal{A}$ outputs

Figure 6: Simulator of advesary $\mathcal{A}$ in Protocol Π_{p2}, for the case when P_1 is corrupted.