NEW DIRECTIONS IN SECURE MULTI-PARTY COMPUTATION: TECHNIQUES AND INFORMATION DISCLOSURE ANALYSIS

by

Alessandro Nicolo Baccarini

August 2024

A dissertation submitted to the

Faculty of the Graduate School of

the University at Buffalo, The State University of New York

in partial fulfilment of the requirements for the

degree of

Doctor of Philosophy

Department of Computer Science and Engineering

The dissertation of Alessandro Nicolo Baccarini was reviewed by the following:

Marina Blanton

Associate Professor of Computer Science and Engineering

Thesis Advisor, Chair of Committee

Shaofeng Zou

Assistant Professor of Electrical Engineering

Committee Member

Ziming Zhao

Assistant Professor of Computer Science and Engineering

Committee Member

To my parents and fiancée Rebecca,
for their eternal love and support.

Acknowledgments

This thesis is the culmination of a life-long journey filled with a broad range of emotions, experiences, and challenges. There are several individuals who deserve recognition for their integral roles in the successful completion of this dissertation.

First and foremost, I express my deepest gratitude to my advisor Dr. Marina Blanton. Her guidance, wisdom, and encouragement fostered my growth from a novice PhD student into an independent researcher. I wholeheartedly attribute my success to her mentorship philosophy, which I can only hope to emulate one day. I truly appreciate her patience and support throughout our many hours of discussion, as well as the privilege she provided to me to present our work in Switzerland and Portugal. I hold the utmost admiration, appreciation, and respect for her as a researcher, mentor, and colleague.

I thank my committee members for their time and guidance over the course of my PhD. I extend my gratitude to Dr. Shaofeng Zou for his supervision and guidance in our collaborative research projects contained within this dissertation. I thank Dr. Ziming Zhao for his thoughtful comments during my dissertation defense and oral qualifying exam. I also acknowledge my past and present fellow group members at UB, Dr. Chen Yuan and Dennis Murphy, for our collaborations and discussions.

I extend my deepest gratitude to all my family members who have supported me outside graduate school. I especially thank my parents for their love, support, and motivation throughout my

entire life: my mother Debra Baccarini and my late father Isidoro Baccarini. Their encouragement throughout my adolescence into early adulthood ultimately led me to pursue a PhD, for which I will be forever grateful. I thank my grandmother Norma Luongo for her love and our heartfelt conversations, and my uncle Stanley Luongo for always finding a way to make me laugh. I also thank my future in-laws Christopher and Lisa Vignogna, along with the entire Vignogna clan.

Finally, I thank my fiancée Rebecca Vignogna. She has been with me every step of the way, and my rock throughout this journey. She has endured many years of long-distance during our respective graduate educations, and I thank her for her love, patience, and motivation. She reassured me when I was stressed, knew how to make me laugh when I was upset, and inspired me to work hard every single day. I truly could not have done it without her, and I look forward to living the rest of our lives together as "Dr. and Dr."

Lastly, I thank our cat Matcha, whose company kept me sane through many hours of research, writing, and editing. This dissertation is as much his as it is mine.

Table of Contents

List of Tables

List of Figures

List of Protocols

Abstract

Secure multi-party computation (SMC) refers to the act of multiple participants jointly computing an arbitrary function on private inputs without disclosing any information beyond the output. SMC has grown in mainstream popularity with a wide range of applications in machine learning, healthcare, and, data analytics.

Among the available techniques, secret sharing offers competitive performance and is a popular choice in a variety of domains. While many schemes perform computation over a field, alternative techniques that operate over the ring $\mathbb{Z}_{2^k}$ (such as replicated secret sharing, or RSS) have been proposed and shown to substantially boost performance. The caveat associated with existing frameworks is that they are often bespoke three-party solutions, which do not easily generalize in the event the computational setup needs to be adjusted to accommodate more participants or a higher collusion threshold. Moreover, there is a gap in the literature for a ring-based framework that supports both integer and floating-point computation. The first aspect of this dissertation fills this gap by developing a comprehensive protocol suite in the semi-honest, honest majority setting based on RSS. We construct a set of elementary building blocks, which enable us to build more complex operations to ultimately support general-purpose computation. We demonstrate that our techniques are substantially faster than their field-based equivalents when instantiated with a different number of parties, and perform on par with or better than state-of-the-art techniques with constructions designed for a fixed number of parties.

In light of these and many other recent advancements of SMC, the literature lacks a means of measuring information disclosure from the output of secure function evaluations. In other words, how much information is leaked about private input(s) by virtue of releasing the output? The second aspect of this dissertation answers this question and more through our framework for measuring information disclosure of practically significant data analysis functions. Motivated by the City of Boston gender pay gap studies, we analyze the computation of the average salaries and quantify information disclosure about private inputs of one or more participants (the target) to an adversary via information-theoretic techniques (i.e., computing the entropy). We study a number of different distributions and experimental configurations and provide recommendations for real-world SMC deployments of the average salary computation. This approach is foundational when we pivot to more advanced statistical measures (such as the variance, maximum, and median), which introduce new challenges based on the absence of established closed-form expressions for the entropy. Fortunately, data-driven techniques circumvent these limitations and enable us to apply our analysis to this broader domain of functions, each of which displays unique and surprising behaviors.

Chapter 1

Introduction

Secure multi-party computation (SMC) and other privacy-preserving computing techniques have enabled many opportunities for practical deployment for data analysis on sensitive information. SMC has breached the domain of purely academic interest in recent years and garnered mainstream popularity. For example, healthcare institutions jointly contributing patient data for the purpose of developing breakthroughs in research and treatment [71]. Data privacy laws (such as HIPAA) impose strict guidelines for the dissemination of medical records and personally identifiable information which may hamper technological advancements. Secure computation techniques enable scientific professionals to conduct research to develop novel advancements in treatment and patient care while maintaining compliance with privacy regulations. More broadly, wider adoption of privacy-preserving technologies, and secure computation in particular, can lead to higher security standards and practices in broad aspects of our society. As such, the research community is constantly evolving and adapting to the public's privacy needs.

The field of secure computation has substantially matured in the last decade with advances in

overall performance, while concurrently developing tools to facilitate broader adaptation by general audiences. At the core, secure computation technologies rely on fundamental building blocks, which cultivates a vast range of applications in data analytics [158], medicine [72, 33, 124, 52], biometrics [32, 38, 37] and machine learning [160, 50, 130, 136, 49, 104, 60, 161, 79]. Many mainstream corporations such as Google and Apple have integrated secure computation techniques into their products [110, 162, 95, 30] and the number of start-up companies offering related products is expanding (see, e.g., [4, 1, 116, 2]). However, a number of fundamental questions still need to be addressed by the research community in order to make secure computing practices commonplace.

Data analytics encompasses a broad domain of computational technologies, ranging from intricate computations (such as modeling, data cleansing, and communication), to fundamental descriptive statistics such as the mean (average), standard deviation, and order statistics (maximum, minimum, median). This dissertation focuses on advancing secure computation techniques upon which a diverse range of applications are built, as well as developing a deeper understanding of nontrivial aspects of the field as a whole.

1.1 Challenge and Scope

Informally, SMC is the notion of multiple parties working together to compute an arbitrary function on secret inputs. No information is revealed other than the output of the function itself. Many options are available within SMC to realize specific security and performance guarantees. *Secret sharing* (SS) offers superior performance for arithmetic operations over other prevalent cryptographic tools such as homomorphic encryption. In this dissertation, we consider the assumption that all parties are *semi-honest*, meaning they will not deviate from a specified protocol but will at-

tempt to learn as much information as possible. Moreover, we assume there is an *honest majority*, such that only a minority of the participants can be corrupted.

Traditional multi-party computation techniques are performed over a field $\mathbb{F}_p$ with prime p. This makes frequent use of modulo reduction a necessity, increasing the cost of the computation. Alternative approaches have been developed to perform computation over rings such as $\mathbb{Z}_{2^k}$ [39, 21, 58, 62] offer a highly attractive performance for their inherent compatibility with native CPU instructions. Shamir SS [148] is a common and efficient choice in the literature for computation in the honest majority setting. Unfortunately, Shamir's construction is incompatible with ring computation due to the need for multiplicative inverses as part of polynomial interpolation. The current landscape of techniques in the semi-honest, honest majority setting which can perform computation over ring $\mathbb{Z}_{2^k}$ for some k are limited to a fixed number of parties, most commonly to 3 (see, e.g., [21, 117, 130, 50, 49]). This implies that the techniques do not easily generalize to a larger number of participants, should there be a need to adjust the computation setup, e.g., to permit the use of a higher collusion threshold. Moreover, most multi-party frameworks are limited to integer computation and do not consider floating-point operations. Recent approaches that attempt to bridge this gap [140, 34, 43, 44, 48, 46, 45, 105, 111, 101, 114, 120, 144, 139] are often limited to two- and three-party solutions. The earliest comprehensive floating-point framework [16] with support for an arbitrary number of parties used Shamir SS. Constructing a general-purpose computation framework based on replicated secret sharing (RSS) to support $n \geq 3$ computational parties constitutes the first aspect of this dissertation.

In light of these developments, an underlying question remains regarding whether a secure function evaluation is "safe." As stated, the cryptographic community established standard security definitions by the requirement that no information about private inputs is disclosed through-

out a function's evaluation. That is, given a function f evaluated on private inputs $x_1, x_2, \ldots$ coming from different sources, security is achieved if a participant does not learn more information than the function output and any information that can be deduced from the output and its private input. However, there are no constraints on the types of functions that can be evaluated in this framework. The information one participant can deduce from the output and their private input about another participant's private input is potentially large.

This problem is typically handled by assuming that the function being evaluated is agreed upon by and acceptable to the data owners as not to reveal too much information about private inputs. Our ability to evaluate functions in this aspect and determine what functions might be acceptable is currently limited. This introduces a number of nontrivial questions inherent to secure computation as a whole: how much information is leaked about private inputs from releasing the output of the computation? How should we quantify such a measure, such that a prospective participant can learn actionable information about the true risk associated with participating in a computation? In the event a function does leak information, what measures can we take to mitigate the disclosure to an acceptable level? Answering these questions for a specific class of functions with practical significance (such as in data analytics computations) is worthy of study, and constitutes the second aspect of this dissertation.

1.2 Dissertation Overview

Given the current landscape of SMC and opportunities to advance the state-of-the-art as outlined above, we distill the fundamental goal of this dissertation into the following objective:

To develop a comprehensive suite of RSS protocols in the semi-honest, honest majority setting over a ring for an arbitrary number of parties, and to design techniques for evaluating information disclosure from secure function evaluation.

This dissertation fills two vacancies in SMC literature. First, our framework serves as a competitive alternative to existing solutions designed for both secure integer and floating-point computation. We deepen our understanding of fundamental questions regarding information leakage inherent to secure computation, specifically in the context of descriptive statistical functions. We summarize the main results of the dissertation below.

Part I. Comprehensive n-party RSS framework over a ring. The first component of the dissertation pertains to developing a complete set of efficient RSS protocols over a ring in the semi-honest, honest majority setting for any number of parties. In Chapter 4, we design a comprehensive set of fundamental RSS protocols over an arbitrary finite ring for multiplication, share reconstruction, and entering private values into the computation. After establishing these building blocks, we narrow our focus to the ring $\mathbb{Z}_{2^k}$ to build higher-level protocols such as random bit generation, comparisons, and division. In this process, we develop a generalization to n parties of [34]'s optimized B2A protocol, which frequently appears in constructions. Extensive benchmarking demonstrates the clear advantage of RSS over field-based techniques, as well as presenting RSS as a compelling alternative to existing state-of-the-art solutions.

In Chapter 5, we further build upon our integer operations to construct protocols for floating-point operations. We utilize the seminal work of [16] in conjunction with the recent works of [140, 44] to develop fundamental floating-point arithmetic operations including addition, multiplication, division, and comparisons. We discuss any required modifications necessary to maintain compliance with the IEEE 754 standard for floating-point arithmetic.

Part II. Information disclosure analysis for secure function evaluations. The second aspect of this dissertation focuses on analyzing information disclosure from secure statistical function evaluations. In Chapter 7, we leverage the entropy-based definitions from [12] to formulate our metric for quantifying information disclosure and subsequently apply these definitions to functions with significant practical relevance. Chapters 8 and 9 contain a comprehensive case study of the average salary computation as used in the Boston gender pay gap study [113]. This analysis unveils a number of interesting properties about the average computation itself, including the independence of the adversary's inputs on the disclosure, the non-impact the choice of input distribution has on the leakage, and how increasing the number of participants lowers the overall disclosure.

After our extensive treatment of the average function, we broaden the scope of our analysis in Chapter 10 to encompass more complex functions and study the impact of potential mitigation techniques. Specifically, we analyze nontrivial statistical functions prevalent in data analysis, such as the standard deviation, median, and max/minimum. These functions pose more of a challenge to analyze analytically, due to their lack of closed-form expressions for the entropy. To overcome this limitation, we turn to data-driven techniques to estimate the information disclosure. Our findings reveal a broad range of interesting interactions between the functions, the distributions used to model participant inputs, and the computational configuration itself.

We conclude the dissertation in Chapter 11 and discuss open research problems.

Part I

A Replicated Secret Sharing Framework for an Arbitrary Number of Parties

Chapter 2

Related Work

2.1 Secret Sharing Schemes

Secret sharing [148, 31] is a popular choice for secure multi-party computation, and common options include Shamir SS [148], additive SS, and RSS [96] for three parties. Computation over rings, and specifically $\mathbb{Z}_{2^k}$, has recently gained attention in publications including [39, 21, 117, 58, 64, 62, 77, 10, 102, 60]. We can distinguish between three-party techniques based on RSS such as [39, 21, 117, 64, 77, 10, 102]; multi-party techniques based on additive SS such as [58, 62], often for the setting with no honest majority; and ad-hoc techniques for three or four parties that utilize one or more types of rings with constructions for specific applications such as [100] and others.

The first category is the closest to this work and includes Sharemind [39], a well-developed framework for three-party computation with a single corruption using custom protocols; Araki et al. [21] who use three-party with a single corruption to support arithmetic or Boolean circuits; and several compilers from passively secure to actively secure protocols [117, 64, 77, 10]. Dalskov et al. [61] also studied four-party computation with a single corruption. We are not aware of existing

multi-party techniques for an honest majority over a ring that extends beyond three parties or multi-party protocols based on RSS over a ring. While RSS provides the most utility for a small number of parties, we still find it desirable to support more participants and build additional techniques for this setting. For example, if our matrix multiplication protocol over a ring with three parties is 100 times faster than field-based computation, it will remain faster even if the work increases when the number of parties is larger than 3.

We rely on the results of Damgård et al. [62] for some of our protocols. While this work is for the SPD$\mathbb{Z}_{2^k}$ framework [58] in the malicious setting with no honest majority, once we develop elementary building blocks, the structure of higher-level protocols can remain similar. Composite protocols such as comparison, conversion, and truncation require a large number of random bits. We leverage the edaBit protocol from [79] to efficiently generate sets of binary and arithmetic shared bits. Their technique improves upon the daBit technique [143]. Rabbit [121] builds on daBits [143] and edaBits [79] and developed an efficient n-party comparison protocol by relying on commutativity of addition over fields and rings. Their protocol offers significant improvement over [79] in most adversarial settings over a field, but remains comparable with a passively secure honest majority over a ring.

2.2 Secure Floating-Point Arithmetic

We are not aware of an RSS-based framework that features floating-point multi-party protocols for ring computation. Nonetheless, the concept of fusing multi-party computation and floating point arithmetic has a rich history, with its genesis of Aliasgari et al.'s [16] landmark work. The authors were the first to design a complete suite of protocols for floating point arithmetic based

on [48]'s fixed-point protocols using Shamir secret sharing. This work also featured complex functions including square root, logarithm, and exponentiation. The techniques proposed were later implemented into the PICCO compiler [165]. MP-SPDZ [102, 9] is an aggregate framework with support for several SMC techniques and security settings, including RSS in the semi-honest, honest-majority setting. The software creates virtual machines for many protocol variants which are responsible for compiling user programs into secure machine code. The software realizes the floating-point specifications from [16]. While ring computation using RSS is supported by MP-SPDZ, it is limited to three computational parties.

SecFloat [140] is the most recent work to feature a two-party semi-honest floating-point framework over a ring with 2-out-of-2 additive secret sharing. The authors claim to offer the first fully IEEE 754-compliant SMC library, satisfying the core requirements for precision, rounding, and correctness. Their implementation was integrated into CrypTFlow's software package [112, 8] on top of SIRNN's [141] building blocks. SecFloat can support representations outside the standard such as Google's BFloat16 or NVIDIA's TensorFloat32, as realized in their later work [139]. Naturally, SecFloat's two-party construction accounts for a stronger setting (dishonest majority), but we emphasize our this work supports an arbitrary number of parties in the honest majority setting.

We acknowledge other recent works in the area not directly related to this dissertation but fall within the scope of multi-party techniques on floating-point inputs [34, 43, 44, 48, 46, 45, 105, 111, 101, 114, 16, 120, 144].

Chapter 3

Background

We consider a secure multi-party setting with n computational parties, out of which at most t can be corrupt. We work in the setting with an honest majority, i.e., $t < n/2$ and semi-honest participants and use simulation-based security, formulated as follows:

Definition 1

Let parties $P_1, \ldots, P_n$ engage in a protocol Π that computes function $f(\mathsf{in}_1, \ldots, \mathsf{in}_n) = (\mathsf{out}_1, \ldots, \mathsf{out}_n)$, where in_i and out_i denote the input and output of party P_i, respectively. Denote $\text{VIEW}_\Pi(P_i)$ as the view of participant P_i during the execution of protocol Π. More precisely, P_i's view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, \ldots, m_k$ passed between the parties during protocol execution: $\text{VIEW}_\Pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_k)$. Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the participants for $t < n$, $\text{VIEW}_\Pi(I)$ denote the combined view of participants in I during the execution of protocol Π (i.e., the union of the views of the participants in I), and $f_I(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ denote the projection of $f(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ on the coordinates in I (i.e., $f_I(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ consists of the i_1th, $\ldots$, i_tth element that $f(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ outputs).

We say that protocol Π is t-private in the presence of semi-honest adversaries if for each coalition of size at most t there exists a probabilistic polynomial time simulator S_I such that $\{S_I(\mathsf{in}_I, f_I(\mathsf{in}_1, \ldots, \mathsf{in}_n)), f(\mathsf{in}_1, \ldots, \mathsf{in}_n)\} \Leftrightarrow \{\mathrm{VIEW}_\Pi(I), (\mathsf{out}_1, \ldots, \mathsf{out}_n)\}$, where $\mathsf{in}_I = \bigcup_{P_i \in I}\{\mathsf{in}_i\}$ and $\Leftrightarrow$ denotes computational or statistical indistinguishability.

As customary with SS techniques, the set of computational parties does not have to coincide with (and can be formed independently of) the set of parties supplying inputs in the computation (input providers) and the set of parties receiving the output of the computation (output recipients). Then, if a computational party learns no output, the computation should reveal no information to that party. Consequently, if we wish to design a functionality that takes secret-shared input and produces shares of the output, any computational party should learn nothing from protocol execution.

3.1 Secret Sharing

A secret sharing scheme allows one to produce shares of secret x such that access to a predefined number of shares reveals no information about x. In the context of secure multi-party computation, each of the n participants receives one or more shares x_i and in the case of (n, t) threshold SS schemes, possession of shares stored at any t or fewer parties reveals no information about x, while access to shares stored at $t + 1$ or more parties allows for reconstruction of x. Of particular importance are linear SS schemes, which have the property that a linear combination of secret shared values can be performed locally on the shares. Examples of linear secret sharing schemes include additive secret sharing with $x = \sum_i x_i$ (as used in Sharemind [39] with $n = 3$ and in SPDZ [65] with any n), Shamir secret sharing which realizes (n, t) secret sharing with $t < n/2$ and

represents a share as evaluation of a polynomial on a distinct point, and replicated secret sharing, which we discuss next.

3.2 Replicated Secret Sharing

Our techniques utilize *replicated secret sharing* (RSS) [96] which has an associated access structure Γ. An access structure is defined by qualified sets $Q \in \Gamma$, which are the sets of participants who are granted access, and the remaining sets of the participants are called unqualified sets. In the context of this work, we only consider threshold structures in which any set of t or fewer participants is not authorized to learn information about private values (i.e., they form unqualified sets), while any $t+1$ or more participants are capable of jointly reconstructing the secret (and thus form qualified sets). RSS can be defined for any $n \geq 2$ and any $t < n$. To secret-share private x using RSS, we treat x as an element of a finite ring $\mathcal{R}$ and additively split it into shares x_T such that $x = \sum_{T \in \mathcal{T}} x_T$ (in $\mathcal{R}$), where $\mathcal{T}$ consists of all maximal unqualified sets of Γ (i.e., all sets of t parties in our case). Then each party $p \in [1, n]$ stores shares x_T for all $T \in \mathcal{T}$ subject to $p \notin T$. In the general case of (n, t)-threshold RSS, the total number of shares is $\binom{n}{t}$ with $\binom{n-1}{t}$ shares stored by each party, which can become large as n and t grow. In what follows, we use the notation $[x]$ to mean that (private) x is secret shared among the parties using RSS.

Example 1

Considering the $(4, 2)$ setting, $\mathcal{T}$ consists of 6 sets $\mathcal{T} = \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}$ and thus there are 6 corresponding shares for every secret-shared x. Then party 1 stores shares $x_{\{2,3\}}, x_{\{2,4\}}, x_{\{3,4\}}$, party 2 stores $x_{\{1,3\}}, x_{\{1,4\}}, x_{\{3,4\}}$, and so on.

The parties will need to perform computation on secret shared values. The first important

property of RSS is that it is linear. That is, given shares of private values, each party can locally compute the corresponding shares of a linear combination of the values. For example, to add $[a]$ and $[b]$, party p computes $a_T + b_T$ (in $\mathcal{R}$) for each $T \in \mathcal{T}$ that p stores. A number of other operations, such as multiplications, reconstructing a value from its shares, etc., are interactive. We consequently describe in Section 4.1 the way we realize these operations. An important optimization that we rely upon is non-interactive evaluation of a pseudo-random function (PRF) using RSS in the computational (as opposed to information-theoretic) setting as proposed in [59]; see Section 4.1 for detail.

In what follows, we use the notation "$\leftarrow$" to denote the output of randomized algorithms, while the notation "$=$" refers to deterministic assignment.

Chapter 4

Integer Protocols

This chapter contains our comprehensive RSS framework for integers. We first establish several fundamental building block protocols. These enable us to construct more complex operations, which we experimentally evaluate against their Shamir SS equivalents, as well as the current state-of-the-art.

4.1 Building Blocks

Recall that RSS is linear. In addition to adding secret-shared values, we use the ability to add/-subtract known integers to a secret-shared value $[a]$ and multiply a secret-shared value $[a]$ by a known integer. Addition $[a] + b$ converts b to $[b]$ without using randomness (e.g., we could set one share to b and the remaining shares to 0 to maintain $\sum_{T \in \mathcal{T}} b_T = b$). Multiplication $[c] = [a] \cdot b$ sets $c_T = a_T \cdot b$ (in $\mathcal{R}$) for every $T \in \mathcal{T}$.

For convenience and without loss of generality, we let $n = 2t + 1$. When $n > 2t + 1$, $2t + 1$ parties can carry out the computation on a reduced set of shares in such a way that there is no need to involve the remaining parties in the computation.

4.1.1 Random Number Generation

A fundamental component of RSS is the parties' ability to generate secret-shared random variables non-interactively. Invocation of $([a_1], [a_2], \ldots) \leftarrow \mathsf{PRG}([s])$ is realized by independently executing a PRG algorithm on each share of s without interaction among the computational parties. Because the output of $\mathsf{PRG}([s])$ is private, we expect it to produce a sequence of secret-shared values (represented as ring elements). Furthermore, in our construction, we only call the PRG to obtain random (secret-shared) ring elements. This means that calling $\mathsf{PRG}(s_T)$ to produce pseudo-random a_T will result in $\mathsf{PRG}([s])$ generating $[a]$, where a is pseudo-random as well because $a = \sum_{T \in \mathcal{T}} a_T$ (in $\mathcal{R}$). This is similar to evaluating a PRF on a secret-shared key in the RSS setting without interaction in [59].

$\mathsf{PRG}(s_T)$ can be realized internally using any suitable algorithm, as long as it is consistent among the computational parties. For example, due to the speed of AES encryption on modern processors, one might implement $\mathsf{PRG}(s_T) = \mathsf{PRF}(s_T, 0)||\mathsf{PRF}(s_T, 1)||\ldots$, where $\mathsf{PRF} : \mathcal{R} \times \{0,1\}^\kappa \rightarrow \mathcal{R}$ is a PRF instantiated with AES.

Let $\mathsf{G} = \mathsf{PRG}([s])$. When the output of G is not consumed all at once, we use notation $\mathsf{G}.\mathsf{next}$ to retrieve the next (secret-shared) element from G. Similarly, if $\mathsf{G}_T = \mathsf{PRG}(s_T)$, notation $\mathsf{G}_T.\mathsf{next}$ refers to the next pseudo-random share output by G_T.

4.1.2 Multiplication

Multiplication $[c] \leftarrow \mathsf{Mul}([a], [b])$ (or simply $[a] \cdot [b]$) is realized using the fact that $a = \sum_{T \in \mathcal{T}} a_T$ and $b = \sum_{T \in \mathcal{T}} b_T$ and thus $[a] \cdot [b] = \sum_{T_1, T_2 \in \mathcal{T}} a_{T_1} \cdot b_{T_2}$ (in $\mathcal{R}$). Note that for any (T_1, T_2) pair, there will be a party holding shares T_1 and T_2, and thus performing this operation involves local multiplication

and addition over different choices of T_1, T_2. More formally, let mapping $\rho : \mathcal{T} \times \mathcal{T} \to [1, n]$ denote a function that for each pair $(T_1, T_2) \in \mathcal{T}^2$ dedicates a party $p \in [1, n]$ responsible for computing the product $a_{T_1} \cdot b_{T_2}$ (clearly, p must possess shares T_1 and T_2). For performance reasons, we also desire that ρ distributes the load across the parties as fairly as possible.

As a result of this (local) computation, the parties hold additive shares of the product $a \cdot b = c$, which needs to be converted to RSS for consecutive computation. This conversion was realized in early publications [126, 27] by having each party create replicated secret shares of their result and distribute each share to the parties entitled to know it (i.e., party p receives shares from each party for each $T \in \mathcal{T}$ subject to $p \notin T$). This results in each participant creating $\binom{n}{t}$ shares and sending $\binom{n-1}{t}$ of them to each party. Consequently, each participant adds the values received for share T and stores the sum as c_T, for each T in its possession.

More recent work, e.g., [21] and others traded information-theoretic security (in the presence of secure channels) for communication efficiency by having the parties generate shared (pseudo–) random values. We pursue this direction as well. However, if this idea is applied naively, it results in unnecessarily high overhead. In particular, if we instruct each party p to generate all shares for its secret, some shares will be known to more than t participants and thus do not contribute to secrecy. Instead, our solution eliminates shares that p does not possess and thus does not contribute to secrecy. Thus, our construction utilizes key material consistent with the setup of the RSS scheme. In particular, we use the same key setup as in pseudorandom secret sharing, where k_T is known by all $p \notin T$. Then, when a party needs to generate a pseudo-random share associated with its value for share T, the party will draw it from the PRG seeded with k_T.

We, however, note that multiple participants may need to draw from the PRG seeded with k_T to produce shares of their values, and it is generally not safe to use the same secret to protect

Protocol 1: $[c] \leftarrow \mathsf{Mul}([a], [b])$

```
// define G_T = PRG(k_T)
// pre-distributed values are [k] and public maps ρ and χ
for p ∈ [1,n] in parallel do
    let S_p = {T ∈ 𝒯 | p ∉ T}
    v^(p)_χ(p) = Σ_{T1,T2∈𝒯, ρ(T1,T2)=p} a_T1 b_T2
    for T ∈ S_p do c_T = 0
    for p' ∈ [1,n] in order do
        for T ∈ S_p do
            if (p' ≠ p) ∧ (p' ∉ T) ∧ (χ(p') ≠ T) then
                c_T = c_T + G_T.next
            else if (p' = p) ∧ (χ(p) ≠ T) then
                z = G_T.next
                c_T = c_T + z
                v^(p)_χ(p) = v^(p)_χ(p) − z
    send v^(p)_χ(p) to each p' ∉ χ(p) (other than itself)
    for p' ∈ [1,n] subject to (p ∉ χ(p')) ∧ (p' ≠ p) do
        receive v^(p')_χ(p') from p', set c_χ(p') = c_χ(p') + v^(p')_χ(p')
    c_χ(p) = c_χ(p) + v^(p)_χ(p)
return [c]
```

multiple values, which is also the case in our application. Instead, multiple elements might be drawn from the PRG (seeded with k_T) to protect different values, and consistent use of the PRG with each seed can be set up by the participants ahead of time, such that this information is public knowledge.

In addition to the mapping ρ, our multiplication protocol requires another mapping $\chi : [1, n] \to \mathcal{T}$, which specifies for each party p the share T (subject to $p \notin T$) that p communicates (with all other shares of p's value being produced as pseudo-random elements). As before, we desire to choose the values of $\chi(p)$ that evenly distribute the load and communication. The above intuition leads us to the optimized n-party multiplication protocol given as Protocol 1.

After computing its private value $v^{(p)}$ according to ρ, each party p distributes it into $\binom{n-1}{t}$ ad-

ditive shares (one of which is communicated while others are computed using PRGs). Afterward, each party sets its c_T as a sum of $t+1$ shares (computed or received) of values $v^{(p')}$ for each party p' entitled to shares c_T. This matches the fact that each share a_T of secret a is maintained by $t+1$ parties. Correctness is achieved by ensuring that in Protocol 1 two different participants p and p' with access to shares T consistently associate the values that they draw from G_T with shares belonging to different parties by always processing the values in the increasing order of participants' IDs. Preparation of the shares in Protocol 1 is done on lines 4 to 12, where a participant either masks its share with a pseudo-random value because it is used by another party or forms its own shares and the value to be transmitted.

In this protocol, each party on average sends t ring elements and draws $\binom{n-1}{t} - 1 + (n-1)\binom{n-2}{t} - t$ pseudo-random ring elements (which is $(t+1)(\binom{n-1}{t} - 1)$ when $n = 2t+1$).[1] The latter can be explained by using $\binom{n-1}{t} - 1$ pseudo-random shares for its value being re-shared and $\binom{n-2}{t}$ shares that it has in common with any other party except the t values that it receives with a symmetric communication pattern. (Recall that each party maintains $\binom{n-1}{t}$ shares of a secret and has $\binom{n-2}{t}$ shares in common with any other party). When the communication pattern is not symmetric, the overall amount of work and communication remains unchanged, but it may be distributed differently. Thus, we refer to the average work and communication in that case.

Compared to other results, the three-party version of our protocol matches communication of recent multiplication from [21], which is available only for three parties. This also improves on communication of standard multiplication using Shamir SS from [86] (information-theoretically secure in the presence of secure channels) by a factor of 2 and improves on communication of Sharemind's three-party multiplication from [106] by a factor of 2. For multi-party multiplication,

[1] It is possible to distribute the load evenly among the parties by appropriately setting the χ function.

it can be desirable to use a different communication pattern when a designated party reconstructs a protected value and communicates it to others (as in, e.g., [63]) which scales better as n grows. However, our version has lower communication when $n = 3$, uses fewer rounds, and n is typically small with RSS.

Example 2

With three parties, we could have party 1 (in possession of shares $\{2\}$ and $\{3\}$) compute (and add) products $a_{\{2\}}b_{\{2\}}$, $a_{\{2\}}b_{\{3\}}$, and $a_{\{3\}}b_{\{2\}}$, party 2 (in possession of shares $\{1\}$ and $\{3\}$) compute products $a_{\{3\}}b_{\{3\}}$, $a_{\{1\}}b_{\{3\}}$, and $a_{\{3\}}b_{\{1\}}$, and party 3 (in possession of shares $\{1\}$ and $\{2\}$) compute products $a_{\{1\}}b_{\{1\}}$, $a_{\{1\}}b_{\{2\}}$, and $a_{\{2\}}b_{\{1\}}$. This defines mapping ρ. Also let $\chi(1) = \{2\}$, $\chi(2) = \{3\}$, and $\chi(3) = \{1\}$. This, for example, means that when party 1 divides its computed value $v^{(1)}$ into shares $v^{(1)}_{\{2\}}$ and $v^{(1)}_{\{3\}}$, the latter is computed using a PRG, while the former is being sent to party 3 (i.e., the other party entitled to have that share). An illustration of the multiplication protocol with these mappings in the three-party setting is given in Figure 4.1.

We state the security of multiplication as follows:

Theorem 1

Multiplication $[c] \leftarrow [a] \cdot [b]$ is secure according to Definition 1 in the (n, t) setting with $n = 2t + 1$ in the presence of secure communication channels and assuming PRG is a pseudo-random generator.

Proof. Let I denote the set of corrupt parties. We consider the maximal amount of corruption with $|I| = t$. Since the computation proceeds on secret shares and the parties do not learn the result, no information should be revealed to the computational parties as a result of protocol execution.

We build a simulator S_I that interacts with the parties in I as follows: when a party $p \in I$ expects to receive a value from another party $p' \notin I$ in step 15 of the computation according to

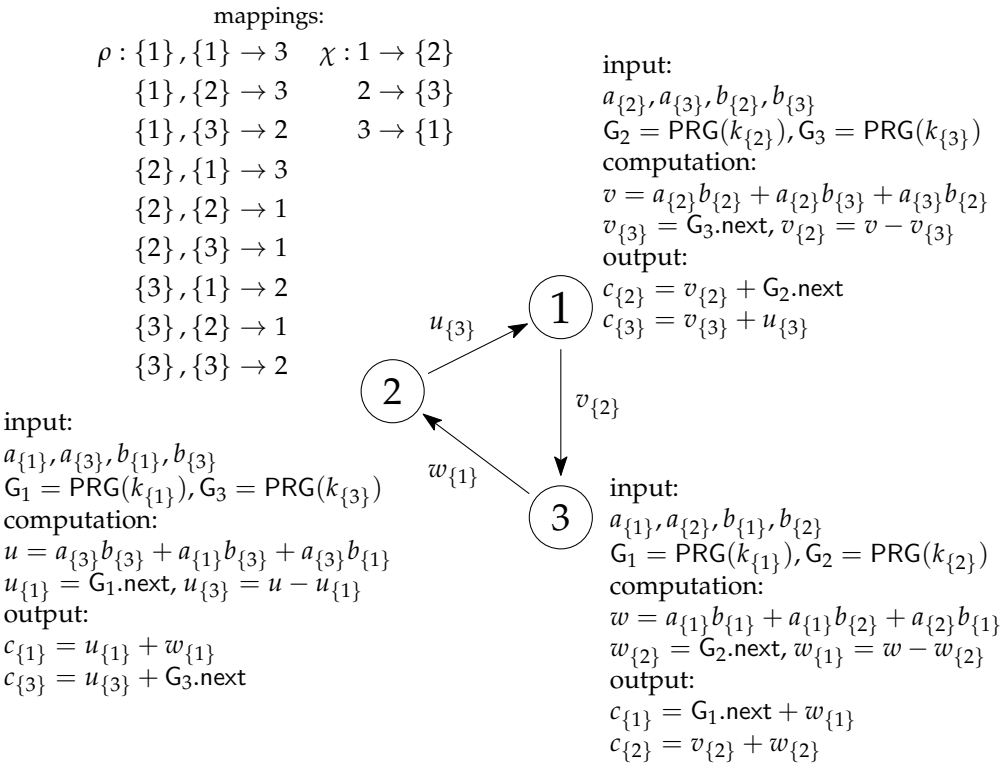


Figure 4.1: Sample three-party multiplication $[a]\cdot[b]$. Arithmetic is in $\mathcal{R}$.

function χ, S_I chooses a random element of $\mathcal{R}$ and sends it to p. S_I preserves the consistency of the view and ensures that when the same value is to be sent by p' to multiple parties in I, all of them receive the same random value. This is the only portion of the protocol where corrupt parties can receive values (that the simulator produces), and the only portion of the protocol when a corrupt party p may send a value to an honest party p' is step 13, which S_I receives on behalf of p'. All other computation is performed locally, in which S_I does not participate.

We next argue that the simulated view is computationally indistinguishable from the real view. First, note that the corrupt parties in I collectively hold shares a_T, b_T and keys k_T (and thus can compute values $\mathsf{G}_T.\mathsf{next}$) for each $T \in \mathcal{T}$ such that $\exists p \in I$ and $p \notin T$. This entitles the corrupt parties to compute the corresponding shares c_T, but the rest of the shares must remain unknown, such that they are unable to compute c. Next, notice that when $|I| = t$, there is only one share

$T^* = I$ such that all parties $p \in I$ have no access to k_{T*} and c_{T^*}, while all parties $p' \notin I$ store those values. Then there are two cases to consider: (1) If one or more parties $p \in I$ receive $\chi(p')$'s share of $v^{p'}$ from another party $p' \notin I$ (it must be the case that $\chi(p') \neq T^*$), the received share has been masked by a fresh pseudo-random element from G_{T^*}, is, therefore, pseudo-random and indistinguishable from random by any $p \in I$. (2) If no party $p \in I$ receives a value from any given $p' \notin I$, indistinguishability is trivially maintained. □

The multiplication protocol we present shares conceptual similarities with (optimized) multiplication from [103]. In particular, both sample pseudorandom secret shares according to the access structure and communicate a single (properly protected) element to a subset of other participants. Our solution explicitly defines all maps and the computation associated with computing each share of the output, while the latter appears to be under-specified in [103].

The computation associated with multiplication can be generalized to compute the dot-product of two secret-shared vectors $\mathsf{DotProd}(\langle [a^1], \ldots, [a^N] \rangle, \langle [b^1], \ldots, [b^N] \rangle)$, or evaluate any other multivariate polynomial of degree 2, using the same communication and the same number of cryptographic operations as in multiplication. For this purpose, we only need to change the computation in step 3 of the multiplication protocol. For example, to compute the dot product, we modify step 3 to compute $v^{(p)} = \sum_{T_1, T_2 \in \mathcal{T}, \rho(T_1, T_2) = p} \sum_{i=1}^{N} a^i_{T_1} b^i_{T_2}$ (in $\mathcal{R}$), while the remainder of the protocol is unchanged.

Table 4.1 contains the performance of multiplication and the other building blocks presented in this section. Communication is measured as the number of ring elements sent by each party and computation is the number of cryptographic operations (i.e., retrieval of the next pseudo-random element using a PRG) per party.

Operation	Rounds	Communication	Crypto Operations
$\mathsf{PRG}([s]).\mathsf{next}$	0	0	$\binom{n-1}{t}$
$\mathsf{Mul}([a],[b])$	1	t	$(t{+}1)\left(\binom{n-1}{t}-1\right)$
$\mathsf{Open}([a])$	1	t	0
$\mathsf{DotProd}(\langle[a^1],\ldots,[a^N]\rangle,\langle[b^1],\ldots,[b^N]\rangle)$	1	t	$(t+1)\left(\binom{n-1}{t}-1\right)$
$\mathsf{Input}^{p^*}(a_1,\ldots,a_m)^\dagger$	1	mt	$(t+1)\left(2\binom{n-1}{t}-1\right)$

Table 4.1: Performance of basic RSS operations in the (n,t) setting with computation and communication (measured in ring elements) per party. $(\dagger)$ The reported computation and communication for Input^{p^*} is the total across all parties since the protocol is asymmetric.

4.1.3 Share reconstruction (Open)

Reconstruction of a secret shared value $a = \mathsf{Open}([a])$ amounts to communicating missing shares to each party such that the value could be reconstructed locally from all shares. Recall that there are $\binom{n}{t}$ total shares and each party holds $\binom{n-1}{t}$ of them. Thus, each party receives $d = \binom{n}{t} - \binom{n-1}{t}$ missing shares during this operation.

Our next observation is that when n is not small (such as when $n = 7$), the value of d will exceed n and transmitting d messages to each party is not needed. Since the value is reconstructed as the sum of all shares, it is sufficient to communicate sums of shares instead of the individual shares themselves. Recall that $[a]$ can be reconstructed by $t+1$ parties. This means that it is sufficient for a participant to receive one element (i.e., a sum of the necessary shares) from t other parties.

As before, we would like to balance the load between the parties and ideally have each party transmit the same amount of data. This means that we instruct each party to send information to t other parties according to another agreed upon mapping $\nu : [1,n] \to (\mathcal{T},[1,n])^d$. For each party p, this mapping will specify which of p's shares should be communicated to which other party.

The mapping ν will then define computation associated with this operation: each p computes $\sum_{T,\nu(p)=T,p'} a_T$ (in $\mathcal{R}$) for each $p' \neq p$ present in the mapping and sends the result to p'.

Similar to other SS frameworks, simply opening the shares of a maintains the security of the computation (in the sense that no information about private values is revealed beyond the opened value a). This is because we maintain that at the end of each operation, secret-shared values are represented using random shares. In particular, it is clear that the result of $\mathsf{PRG}([s])$.next produces random shares; shares are properly re-randomized during multiplication of $[a]$ and $[b]$, and shares of $[a]+[b]$ and $[a]-[b]$ are random if the shares of $[a]$ and $[b]$ are random themselves.

Example 3

With $n = 3$, we could have $\nu(1) = (\{3\},3)$, $\nu(2) = (\{1\},1)$, and $\nu(3) = (\{2\},2)$, such that $\nu(p) = (\{p-1\},p-1)$ (where $p-1 = 3$ for $p = 1$), This corresponds to the communication pattern displayed in Figure 4.2.

For $n = 5$, we may set $\nu(1) = (\{2,5\},5),(\{4,5\},5),(\{4,5\},4),(\{2,4\},4)$, $\nu(2) = (\{1,3\},1),(\{1,5\},1),(\{1,5\},5),(\{3,5\},5)$, etc., which corresponds to

$$\begin{aligned}\nu(p) =&(\{p-1,p-4\},p-1),(\{p-1,p-2\},p-1),\\ &(\{p-1,p-2\},p-2),(\{p-4,p-2\},p-2)\end{aligned}$$

As before $p-i$ for $p \in [1,n]$ means $p-i+5$ if $p-i<1$. This means that party 1 will send the sum of shares $\{2,5\}$ and $\{4,5\}$ to party 5 (instead of sending the individual shares) and the sum of shares $\{2,4\}$ and $\{4,5\}$ to party 4.

When performing computation over the ring $\mathbb{Z}_{2^k}$, if the bitlength of the secret ℓ is smaller than the ring size k (i.e., $\ell < k$), we must reduce each share modulo 2^ℓ prior to parties transmitting their

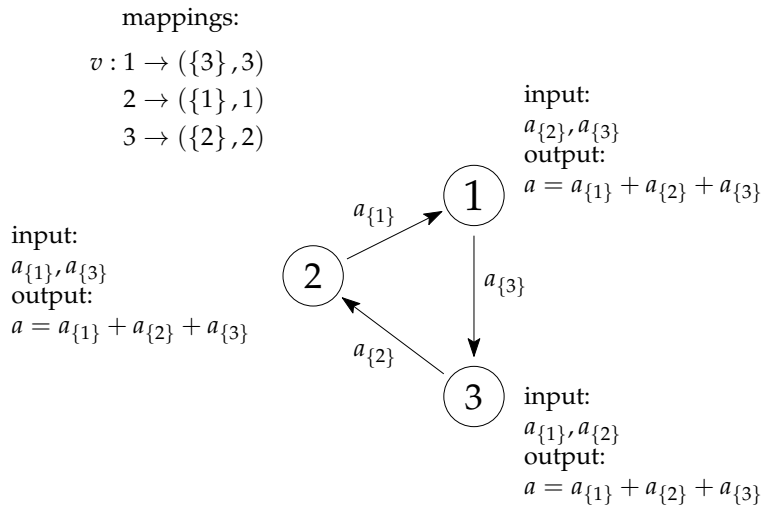


Figure 4.2: Sample three-party $\mathsf{Open}([a])$. Arithmetic is in $\mathcal{R}$.

shares to guarantee no information is revealed beyond the specified ℓ bits.

4.1.4 Inputting Private values

There will be a need to enter private values into the computation, and we discuss the corresponding protocols in this section. We start with a general case when a participant who is not a computational party supplies their input into the computation and consequently discuss an optimized version when the input owner is one of the computational parties.

The input owner holds a private value a which will be represented as an element of ring $\mathcal{R}$. The input owner will need to generate replicated shares that correspond to a and send them to the computational parties. This will be the easiest way to proceed when there is only one element to share. However, when someone is sharing a vector of elements, we can save on communication by using pseudo-random shares. All shares except one for any element can be pseudo-random and computed locally by computational parties after obtaining a PRG seed. This means that among all shares $T \in \mathcal{T}$, one is marked as special and is denoted as T^*. The corresponding share is computed

Protocol 2: $[a_1], \ldots, [a_m] \leftarrow \mathsf{Input}(a_1, \ldots, a_m)$

```
   // Denote IO as the ''input owner''
1  for T ∈ 𝒯 \ {T*} do
2      IO generates random k_T and sends it to each p ∈ T
3  for i ∈ [1, m] do
4      for T ∈ 𝒯 \ {T*} do
5          each p ∉ T sets share a_{i,T} = PRG(k_T).next
6      IO computes a_{i,T*} = a_i − Σ_{T∈𝒯\{T*}} PRG(k_T).next (in ℛ) and sends it to p ∉ T*
7      each p ∉ T* sets share a_{i,T*} to the value received from IO
8  return [a_1], ..., [a_m]
```

by the input owner and is communicated to all parties with access to that share. The construction for Input is given as Protocol 2.

If the input owner is one of the computational parties, we can capitalize on the fact that the parties already have pre-distributed PRG seeds. We denote the input party as p^*, and give a modified version of the construction for Input^{p^*} in Protocol 3. Note that p^* has access to a subset of the PRG seeds corresponding to the shares it is entitled to have access to, but not to all seeds. While we could generate new seeds for each T such that $p^* \in T$ and make it available to all $p \notin T$ and p^*, these seeds will be accessible to more than t parties and do not contribute to security. Therefore, we instead choose to set such shares to 0 and use only shares accessible to p^*. As a result, T^* will be such that $p^* \notin T^*$, the parties will set shares $a_T = \mathsf{PRG}(\mathsf{k}_T).\mathsf{next}$ for each T such that $p^* \notin T$ and $T \neq T^*$, share T^* will be computed as $a_{T^*} = a - \sum_{T \text{ s.t. } p^* \notin T \wedge T \neq T^*} a_T$ (in $\mathcal{R}$) by p^* and communicated to all $p \notin T^*$, and all remaining shares a_T are set to 0.

All variants use a single round. When a single input is shared by an external party, the input owner simply generates all $\binom{n}{t}$ shares and communicates them to the computational parties (each share is stored by $t+1$ participants). This cost (which becomes the sharing of a PRG seed) is amortized among all inputs when sharing multiple inputs. The additional cost per input for the

Protocol 3: $[a_1], \ldots, [a_m] \leftarrow \mathsf{Input}^{p^*}(a_1, \ldots, a_m)$

```
   // p* is a computational party
 1 for p ∈ [1, n] in parallel do
 2     for i ∈ [1, m] do
 3         let S_p = {T ∈ 𝒯 \ {T*} | p ∉ T}
 4         for T ∈ S_p do
 5             if p* ∉ T then
 6                 a_{i,T} = PRG(k_T).next
 7             else
 8                 a_{i,T} = 0
 9         if p = p* then
10             a_{i,T*} = a_i − Σ_{T∈S_p} a_{i,T} (in ℛ)
11             send a_{i,T*} to each p' ∉ T*
12         else if p ∉ T* then
13             receive a_{i,T*} from p*
14 return [a_1], ..., [a_m]
```

input owner becomes generation $\binom{n}{t} - 1$ pseudorandom ring elements and communicating the last, computed share to $t + 1$ computational parties, i.e., the total communication is $t + 1$ ring elements. Each computational party needs to generate $\binom{n-1}{t}$ or $\binom{n-1}{t} - 1$ pseudo-random ring elements. When the input is shared by a computational party, there is no setup cost. The input owner needs to generate $\binom{n-1}{t} - 1$ pseudo-random elements (i.e., similar to the number of shares it stores per shared value) and communicate the computed share to t other parties. Each other party computes $\binom{n-2}{t}$ (i.e., the number of shares it has in common with the data owner) or $\binom{n-2}{t} - 1$ pseudo-random ring elements. As will be relevant later, when a computational party is sharing a ring element in the (3,1) setting, the input owner communicates a single ring element to another party (and only one pseudo-random element is computed by the input owner and the remaining computational party).

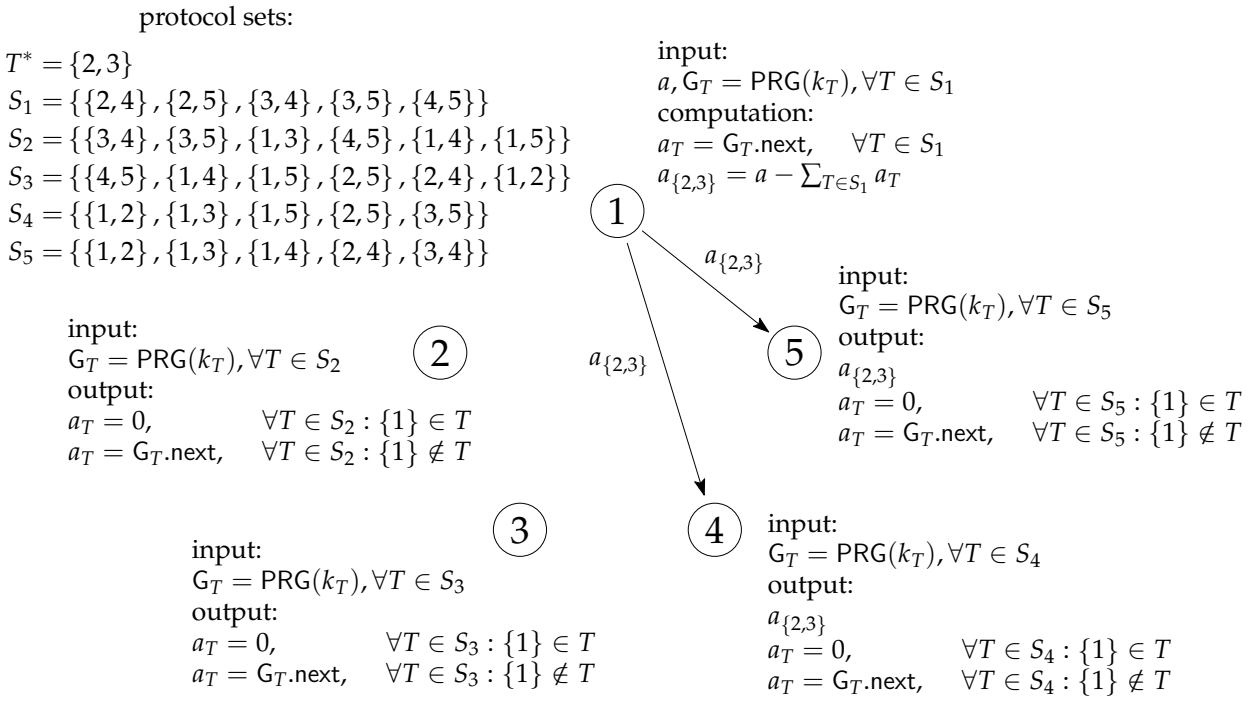


Figure 4.3: Sample five-party $\mathsf{Input}^{p^*}(a)$, where $p^* = 1$. Arithmetic is in $\mathcal{R}$.

Example 4

Consider the setting for $n = 5$ and party 1 is invoking Input ($p^* = 1$). If $T^* = \{2,3\}$, the shares that party 1 is entitled to ($a_{\{2,4\}}, a_{\{2,5\}}, a_{\{3,4\}}, a_{\{3,5\}}$, and $a_{\{4,5\}}$) are generated pseudorandomly using their respective generators $\mathsf{PRG}(\mathsf{k}_T).\mathsf{next}$. Party 1 subsequently calculates $a_{\{2,3\}}$ deterministically and sends the computed share to parties 4 and 5. All remaining shares that party 1 is not entitled to are set to zero by the respective parties who have access, i.e., $a_{\{1,2\}} = a_{\{1,3\}} = a_{\{1,4\}} = a_{\{1,5\}} = 0$. This example is illustrated in Figure 4.3.

Security can be shown as before:

Theorem 2

Input is secure according to Definition 1 in the (n,t) setting with $n = 2t+1$ in the presence of secure communication channels and assuming PRG is a pseudo-random generator.

Proof. It is straightforward to show the security of the full version of Input when the input owner is different from the computational parties. That is, the input owner creates proper shares according to the SS scheme using a PRG. Thus, as long as the security of the PRG holds, the real view is computationally indistinguishable from a simulated view created without the use of any secrets.

However, when the input owner is one of the computational parties, only a reduced set of shares is produced. Thus, we need to evaluate the combined view of each coalition of t corrupt participants. There are two important cases to consider: (i) input owner p^* is a part of the coalition and (ii) it is not.

When p^* is a corrupt participant, building a simulator is trivial: the simulator simply receives shares from the input owner on behalf of honest participants and terminates. Because inputs a_i are available to the corrupt parties, no information needs to be protected and the real and simulated views use identical values.

When there are t corrupt participants who are different from p^*, we simulate the view by choosing a random value for a_{i,T^*} and sending it to each corrupt $p \notin T^*$. What remains to show is that the t corrupt parties do not possess enough shares to reconstruct the secret and, as a result, cannot learn any information about it. In more detail, p^* distributes its secrets using only shares T such that $T \in \mathcal{T} \setminus \{T^*\}$. However, because we use (n, t) threshold SS, there will be a share T possessed by p^* which is not available to any of the t corrupt parties I. Specifically, that share is available to all participants except the corrupt minority I. This means that the corrupt parties will not be able to reconstruct information about the private inputs and the real and simulated views are indistinguishable as long as PRG's security holds.

□

In what follows, references to $\mathsf{Input}_l(a)$ when working over the ring $\mathbb{Z}_{2^l}$ will feature the subscript l to indicate the size of the ring (in bits) in which values are shared.

4.2 Composite Protocols

Having established the necessary building blocks, we direct our attention to constructing composite protocols for more complex integer operations. While the previous operations can be instantiated to work with any finite ring, the techniques we present henceforth work only in a ring $\mathbb{Z}_{2^k}$ for some k. The primary motivation for supporting secure computation over rings is its ability to utilize native CPU instructions for computation. We use the notation $[x]_l$ to denote that secret sharing is over $\mathbb{Z}_{2^l}$, and defer to $[x]$ when there is no ambiguity of the ring size. Lastly, we denote ℓ as the bitlength of the secret, where $\ell \leq k$.

We summarize the theoretical performance of the building blocks discussed in Section 4.1 when instantiated over the ring $\mathbb{Z}_{2^k}$, in conjunction with foundational composite protocols for shared randomness generation and ring conversion in Table 4.3. Communication is measured in the total number of bits sent across all parties. For the sake of completeness, we include the complexities of protocols from other works that are integral to our constructions.

4.2.1 Binary-to-Arithmetic Conversion

Binary-to-arithmetic B2A appears frequently in both integer and floating-point protocols as a means of converting shares of a bit $[x]_1$ shared in $\mathbb{Z}_2$ to $[x]_k$ over $\mathbb{Z}_{2^k}$. The most generic approach follows the procedure from [62], and specifies generating a random bit in $\mathbb{Z}_2$ to protect the input prior to opening. This construction leveraged their RandBit protocol that temporarily switches to a two-bit larger ring $\mathbb{Z}_{2^{k+2}}$ to compute the inverse square root. This is undesirable when computing

Protocol 4: $[x] \leftarrow \mathsf{B2A}([x]_1)$

// one share $\hat{T}$ is marked as ''special''

for $p = 1, \ldots, t$ **in parallel do**
  $v^{(p)} = \bigoplus_{\substack{T \in S_p: \\ \xi(p) = S_p}} x_T$ (in $\mathbb{Z}_2$)
  $[v^{(p)}] \leftarrow \mathsf{Input}_k^p(v^{(p)})$
Parties locally set $v_{\hat{T}}^{(t+1)} = x_{\hat{T}}$ and $v_T^{(t+1)} = 0$ for $T \in \mathcal{T} \setminus \{\hat{T}\}$
$s = t + 1$
for $i = 1, \ldots, \lceil \log(t+1) \rceil$ **do**
  for $j = 1, \ldots, \lfloor s/2 \rfloor$ **in parallel do**
    $[u] \leftarrow \mathsf{Mul}(\left[v^{(2j-1)}\right], \left[v^{(2j)}\right])$ // MulSparse if $\left[v^{(2j)}\right]$ is sparse
    $\left[v^{(j)}\right] \leftarrow \left[v^{(2j-1)}\right] + \left[v^{(2j)}\right] - 2[u]$
  if $s \bmod 2 = 0$ **then**
    $s = s/2$
  else
    $\left[v^{((s+1)/2)}\right] = \left[v^{(s)}\right]$
    $s = (s+1)/2$
$[x] = \left[v^{(1)}\right]$
return $[x]$

in $\mathbb{Z}_{2^{32}}$ or $\mathbb{Z}_{2^{64}}$, since it forces us to size-up to the next larger datatype. Alternative approaches forego the ring switching in favor of more rounds [20] or require oblivious transfer to maintain the cost of the general solution [130]. The work of Blanton et al. [34] designed a novel three-party B2A solution that circumvents the random bit generation entirely while simultaneously lowering the overall communication. We propose an n-party generalization in Protocol 4, which we outline below.

The first phase of the protocol utilizes a mapping to determine which t parties locally compute XOR (in $\mathbb{Z}_2$) of a subset of their accessible shares. We refer to these participants as the "XORing" parties. Formally, let the mapping $\xi : [1, t] \rightarrow \mathcal{T} \setminus \{\hat{T}\}$ denote a function that maps a party $p \in \hat{T}$ to a subset of shares S_p that p is responsible for computing the local XOR $v^{(p)} = \bigoplus_{T \in S_p, \xi(p) = S_p} x_T$

(clearly, p must possess all $T \in S_p$). For performance reasons, we also desire that ξ distributes the load across the parties as fairly as possible. The remaining share that the aforementioned t XORing parties do not have access to (denoted by $x_{\hat{T}}$) can be reshared locally by the remaining $t+1$ participants by setting one share to $v_{\hat{T}}^{(t+1)} = x_{\hat{T}}$ and the rest to zero. Secrets of this form are referred to as "sparse" and parties with access to the nonzero share are referred to as "sparse parties."

Example 5

The mapping for $n = 3$ is simple since only one party p needs to compute the XOR of their local shares, i.e., $\xi(p) = \{\{p+1\}, \{p+2\}\} \pmod{n}$, such that $\hat{T} = \{p\}$.

A mapping for the $n = 5$ configuration may assign 5 shares to p_1 and 4 shares to p_2 as follows:

$$\xi(1) = \{\{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}\}$$

$$\xi(2) = \{\{1,3\}, \{1,4\}, \{1,5\}, \{4,5\}\}$$

and leaving $\hat{T} = \{1,2\}$. For $n = 7$ parties, we may have the following mapping where 11 shares are assigned to p_1 and p_2, and 12 shares to p_3:

$$\begin{aligned}\xi(1) = \{&\{2,3,4\}, \{2,3,5\}, \{2,3,6\}, \{2,3,7\}, \{2,4,6\}, \{2,4,7\},\\ &\{2,5,6\}, \{2,5,7\}, \{2,6,7\}, \{4,6,7\}, \{3,5,7\}\}\\ \xi(2) = \{&\{3,4,5\}, \{3,4,6\}, \{3,4,7\}, \{1,3,4\}, \{3,5,6\}, \{1,3,5\},\\ &\{3,6,7\}, \{1,3,6\}, \{1,3,7\}, \{1,5,7\}, \{1,6,7\}\}\\ \xi(3) = \{&\{4,5,6\}, \{4,5,7\}, \{1,4,5\}, \{2,4,5\}, \{1,4,6\}, \{1,4,7\},\\ &\{1,2,4\}, \{5,6,7\}, \{1,5,6\}, \{1,2,5\}, \{1,2,6\}, \{1,2,7\}\}\end{aligned}$$

which leaves $\hat{T} = \{1,2,3\}$.

Protocol		(Reqs.)	Rounds	Communication
$\mathsf{RandBit}()$	[62]*	(in $\mathbb{Z}_{2^{k+2}}$)	1	$n(k+2)(n-1)$
$\mathsf{B2A}([x]_1)$			2	$n(k+2)(n-1)+nt$
$\mathsf{B2A}([x]_1)$			$\log(t+1)+1$	$ntk\left(t^2-t+1\right)$

Table 4.2: Performance comparison of existing B2A and RandBit protocols with our proposed versions, with communication measured in the total number of bits sent across all parties. Protocols with special requirements are indicated in parentheses. $(*)$ The round and communication complexity of protocols based on [62] are computed under the assumption an optimal multiply-and-open operation is used, such as MulPub from [22].

The next stage involves computing the XOR of the inputted computed XOR value(s) and the sparse secret as a tree over $\mathbb{Z}_{2^k}$, where XOR of is realized by multiplication and local addition, i.e., $[a] \oplus [b] = [a] + [b] - 2[a] \cdot [b]$. We can optimize the protocol further by observing that exactly one product of the tree will involve the sparse secret $[v^{(t+1)}]$, of which the t parties without access to $\hat{T}$ will produce zeros as their local products. These parties can therefore skip the resharing component of multiplication and only receive shares from other participants. We denote this specialized product as $\mathsf{MulSparse}([a], [\hat{b}])$ where shares of $\hat{b}$ are sparse. This functionality reduces the total communication from ntk to $tk(t+1)$, which matches the round and communication complexities of the three-party variant reported in [34] of 2 rounds and $3k$ total bits communicated. The full specification of MulSparse is presented in Protocol 25 in Appendix A.1, alongside its security proof.

We summarize the round and (total) communication performance in Table 4.2. While our protocol requires one extra round in the three-party setting compared to [130]'s single-round OT approach, ours has superior communication complexity ($3k$ versus $6k$ total bits communicated) and can support an arbitrary number of parties. Compared to [62], our technique has the advantage of not needing to switch to a larger ring mid-computation.

We state the security of B2A as follows:

Theorem 3

B2A is secure according to Definition 1 in the (n, t) setting with $n = 2t + 1$ assuming Input, Mul, MulSparse are secure.

Proof. The security of B2A directly follows from the security of the building blocks used. □

A byproduct of our improved B2A construction is that we can use this to generate shared random bits. We discuss this in more detail in Section 4.2.2.

4.2.2 Shared Randomness Generation

Random bit generation is a crucial component of a variety of protocols, including different types of comparisons, bit decomposition, division, etc. Therefore, it is of paramount importance to support this functionality for general-purpose computation. In this work, we examine two variants: (i) generating shares of a single bit as full-size ring elements and (ii) generating shares of k-bit random r as full-size ring elements together with generating shares of individual bits of r in $\mathbb{Z}_2$.

The first variant denoted RandBit generates a single random bit shared in $\mathbb{Z}_{2^k}$. Previous approaches realizing this functionality originate from [47] for field-based SS and then modified in [62] to be compatible with computation over $\mathbb{Z}_{2^k}$. To achieve 50% probability of each outcome of the output bit, the computation temporarily uses a larger ring $\mathbb{Z}_{2^{k+2}}$ for most steps of the protocol, while the remaining computation uses ring $\mathbb{Z}_{2^k}$. This caveat carries the consequence that we are forced to use a slightly smaller ring size to maintain the advantage of using native datatypes, e.g., if $k = 32$, the computation would temporarily be over $\mathbb{Z}_{2^{34}}$ and the smallest usable datatype would be at least 64 bits long. As alluded to in Section 4.2.1, we can leverage our new version of B2A as a new means for generating shares of a single random bit in $\mathbb{Z}_{2^k}$ *without* needing to increase

the ring size beyond k bits.

Protocol 5: $[b] \leftarrow \mathsf{RandBit}()$

// one share $\hat{T}$ is marked as ''special''

for $p = 1, \ldots, t$ **in parallel do**
  Party p samples $v^{(p)} \in \mathbb{Z}_2$
  $\left[v^{(p)}\right] \leftarrow \mathsf{Input}_k^{p^*}\left(v^{(p)}\right)$
for $p = t+1, \ldots, n$ **in parallel do** Parties locally set $v_{\hat{T}}^{(t+1)} = G_{\hat{T}}.\mathsf{next}$
$v_T^{(t+1)} = 0$ for $T \in \mathcal{T} \setminus \{\hat{T}\}$
$s = t+1$
for $i = 1, \ldots, \lceil \log t + 1 \rceil$ **do**
  for $j = 1, \ldots, \lfloor s/2 \rfloor$ **in parallel do**
    $[u] \leftarrow \mathsf{Mul}(\left[v^{(2j-1)}\right], \left[v^{(2j)}\right])$ // MulSparse if $\left[v^{(2j)}\right]$ is sparse
    $\left[v^{(j)}\right] \leftarrow \left[v^{(2j-1)}\right] + \left[v^{(2j)}\right] - 2[u]$
  if $s \bmod 2 = 0$ **then**
    $s = s/2$
  else
    $\left[v^{((s+1)/2)}\right] = \left[v^{(s)}\right]$
    $s = (s+1)/2$
$[b] = \left[v^{(1)}\right]$
return $[b]$

Our approach is presented in Protocol 5. A subset of t parties will leverage their predistributed PRG keys to generate random bits in $\mathbb{Z}_2$ to create shares without requiring communication. The remainder of the protocol follows B2A exactly for computing the XOR of the pseudorandom bits. Assuming participants behave semi-honestly, the output of the protocol is guaranteed to be a uniformly random bit in $\mathbb{Z}_{2^k}$ by the definition of XOR.

The second variant of random bit generation is based on the edaBit construction described in [79] and is denoted as $\mathsf{edaBit}(k)$, where the parameter k specifies the number of generated random bits as well as the bitlength of their representation as integer r. It produces secret-shared k-bit integer r in tandem with shares of the individual bits of r in $\mathbb{Z}_2$. The construction is given as

Protocol 6.

Protocol 6: $([r], [b_0]_1, \ldots, [b_{\ell-1}]_1) \leftarrow \mathsf{edaBit}(k, \ell)$

// steps colored in blue required if $\ell < k$

1 **for** $p = 1, \ldots, t+1$ **in parallel do**

2 Party p samples $r_0^{(p)}, \ldots, r_{\ell-1}^{(p)} \in \mathbb{Z}_2$ and computes $r^{(p)} = \sum_{j=0}^{\ell-1} r_j^{(p)} 2^j$

3 Simultaneously execute $[r^{(p)}] \leftarrow \mathsf{Input}_k(r^{(p)})$ and $[r_i^{(p)}]_1 \leftarrow \mathsf{Input}_1(r_i^{(p)})$ for $i = 1, \ldots, \ell$ with p being the input owner

4 $[r] = \sum_{p=1}^{t+1} [r^{(p)}]$

5 $s = t+1$

6 $\delta^{(j)} = 0$ for $j = 1, \ldots, \lceil \log(t+1) \rceil$

7 **for** $i = 1, \ldots, \lceil \log(t+1) \rceil$ **do**

8 **for** $j = 1, \ldots, \lfloor s/2 \rfloor$ **in parallel do**

9 **if** $\delta^{(2j-1)} \neq \delta^{(2j)}$ **then** $\Delta = \delta^{(j)} + 1$ // add values with the same bitlength

10 **else** $\Delta = \max\left(\delta^{(2j-1)}, \delta^{(2j)}\right)$ // add values with the different bitlengths

11 $\left\langle \left[r_0^{(j)}\right]_1, \ldots, \left[r_{\ell-1+\Delta}^{(j)}\right]_1 \right\rangle \leftarrow \mathsf{BitAdd}\left(\left\langle \left[r_0^{(2j-1)}\right]_1, \ldots, \left[r_{\ell-1+\delta^{(2j-1)}}^{(2j-1)}\right]_1 \right\rangle, \left\langle \left[r_0^{(2j)}\right]_1, \ldots, \left[r_{\ell-1+\delta^{(2j)}}^{(2j)}\right]_1 \right\rangle \right)$

12 $\delta^{(j)} = \Delta$

13 **if** $s \bmod 2 = 0$ **then** $s = s/2$

14 **else**

15 $\left\langle \left[r_0^{((s+1)/2)}\right]_1, \ldots, \left[r_{\ell-1+\delta^{((s+1)/2)}}^{((s+1)/2)}\right]_1 \right\rangle = \left\langle \left[r_0^{(s)}\right]_1, \ldots \left[r_{\ell-1+\delta^{(s)}}^{(s)}\right]_1 \right\rangle$

16 $\delta^{(s)} = \delta^{((s+1)/2)}$

17 $s = (s+1)/2$

18 $[b_0]_1, \ldots, \left[b_{\ell-1+\lfloor \log(t+1) \rfloor}\right]_1 = \left[r_0^{(1)}\right]_1, \ldots, \left[r_{\ell-1+\lfloor \log(t+1) \rfloor}^{(1)}\right]_1$

19 **if** $\ell < k$ **then**

20 $[b_\ell], \ldots, \left[b_{\ell+\lfloor \log(t+1) \rfloor - 1}\right] \leftarrow \mathsf{B2A}\left([b_\ell]_1, \ldots, \left[b_{\ell+\lfloor \log(t+1) \rfloor - 1}\right]_1\right)$

21 $[r] = [r] - 2^\ell \sum_{j=0}^{\lfloor \log(t+1) \rfloor - 1} [b_{j+\ell}] 2^j$

22 **return** $([r], [b_0]_1, \ldots, [b_{\ell-1}]_1)$

The idea consists of $t+1$ parties (without loss of generality, the first $t+1$ parties for this role) each locally generating k random bits and computing the representation of those bits as a k-bit integer (line 2). The bits are inputted into the computation using SS over $\mathbb{Z}_2$, while the integers are entered using shares in $\mathbb{Z}_{2^k}$ (line 3). Since we use Input to generate shares over different rings, we specify the respective subscripts k and 1, which indicate that the shares need to be produced

in their respective rings ($\mathbb{Z}_{2^k}$ and $\mathbb{Z}_2$). If $k = \ell$, the protocol produces an output of the sum of the $t+1$ random integers (without the carry bits) and its bit decomposition, which is computed using bitwise addition BitAdd from [146] of the $t+1$ integers represented as bits in a tree-like manner.

Certain operations (such as bit decomposition) need only a short ℓ-bit edaBit, where $\ell \ll k$, rather than a full-sized edaBit of length k. Our realization of the edaBit functionality in Protocol 6 supports generating short edaBits, which leads to additional steps (colored in blue) of the algorithm that must be performed. Specifically, the consequence of generating a shorter edaBit is that each successive invocation of BitAdd incurs an additional carry bit if inputs of the same bitlength are supplied, up to a total of $\lfloor \log(t+1) \rfloor$ bits. These carry bits must be converted from $\mathbb{Z}_2$ to $\mathbb{Z}_{2^k}$ using B2A and subsequently removed from $[r]$ (lines 20 and 21). If ℓ and k are "close" in value, the performance benefit of generating a shorter edaBit is diminished. Despite performing BitAdd on shorter inputs, the additional cost sustained by the subsequent invocations of B2A outweighs the performance savings, to the point where it is cheaper to simply generate a full-length edaBit.

4.2.3 Comparisons and Equality Testing

Less-than comparisons, $a \overset{?}{<} b$, are traditionally computed using SS by determining the most significant bit of the difference between a and b. Starting from [47], comparison protocols blind the difference by adding a random integer bit decomposition of which is known, open the sum, truncate all but one bit, and compensate for any carry caused by the addition. This logic was adapted to the ring setting in [62] by using building blocks that work over $\mathbb{Z}_{2^k}$. In the solution that we present as Protocol 7, we incorporate the edaBit protocol from [79] for efficient random bit generation into the construction of [62] adopted to the semi-honest setting.

The presence of carry is determined using sub-protocol BitLT which performs a comparison of

Protocol		Rounds	Communication
$\mathsf{Open}([a])$		1	ntk
$\mathsf{Open}_\ell([a])$		1	$nt\ell$
$\mathsf{Mul}([a],[b])$		1	ntk
$\mathsf{MulSparse}([a],[\hat{b}])$		1	$(t+1)tk$
$\mathsf{Input}_k^{p^*}(a)$		1	tk
$\mathsf{kOR}(\langle[a_i]_1\rangle_{i=0}^{\ell-1})$	[146]	$\log(\ell)$	$nt(\ell-1)$
$\mathsf{BitLT}(\langle a_i,[b_i]_1\rangle_{i=0}^{\ell-1})$	[62]	$\log(\ell)$	$2nt(\ell-1)$
$\mathsf{PreOp}(\langle[a_i]_1\rangle_{i=0}^{\ell-1})$	[146]	$\log(\ell)$	$nt(\ell/2)\log(\ell)$
$\mathsf{BitAdd}(\langle a_i,[b_i]_1\rangle_{i=0}^{\ell-1})$	[146]	$\log(\ell)$	$nt\ell\log(\ell)$
$\mathsf{BitAdd}(\langle[a_i]_1,[b_i]_1\rangle_{i=0}^{\ell-1})$	[146]	$\log(\ell)+1$	$nt\ell(\log(\ell)+1)$
$\mathsf{B2A}([x]_1)$		$\log(t+1)+1$	$ntk\left(t^2-t+1\right)$
$\mathsf{RandBit}()$		$\log(t+1)+1$	$ntk\left(t^2-t+1\right)$
$\mathsf{RandBit}()$	(in $\mathbb{Z}_{2^{k+2}}$) [62]	1	$n(k+2)(n-1)$
$\mathsf{edaBit}(k,k)$		$\log(t+1)(\log(k)+1)+1$	$nt^2k(\log(k)+1)$ $+2tk(t+1)$
$\mathsf{edaBit}(k,\ell)$	$(\ell<k)$	$\sum_{i=0}^{\log(t+1)-1}(\log(\ell+i)+1)$ $+\log(t+1)+2$	$nt\sum_{i=0}^{t-1}((\ell+i)(\log(\ell+i)+1))$ $+ntk\log(t+1)(t^2-t+1)$ $+t(t+1)(k+\ell)$
$\mathsf{edaTrunc}(k,m)$		$\log(t+1)(\log(k)+2)+2$	$nt^2k(\log(k)+1)$ $+(2t+1)ntk(t^2-t+1)$ $+3tk(t+1)$

Table 4.3: Performance of various building block protocols, with communication is measured in the total number of bits sent across all parties. Protocols with special requirements are indicated in parentheses.

two bit-decomposed values, one of which is given in the clear, using binary computation over $\mathbb{Z}_2$.

Security of the algorithm follows from prior work and the fact that we use a composition of secure building blocks. In particular, the only values revealed in the protocol (in steps 3 and 9) are information-theoretically protected using freshly generated randomness.

To correctly implement the comparison of two k-bit integers over ring $\mathbb{Z}_{2^k}$, one would need to invoke the MSB protocol 3 times. However, correctness is also guaranteed if we compare two

Protocol 7: $[a_{\ell-1}] \leftarrow \mathsf{MSB}([a])$

// $a = \sum_{i=0}^{\ell-1} a_i 2^i \in \mathbb{Z}_{2^k}$

1 Generate ℓ random bits $[r_0]_1, \ldots, [r_{\ell-1}]_1$ in $\mathbb{Z}_2$ and one random bit $[b]$ over $\mathbb{Z}_{2^k}$

2 Compute $[r] = \sum_{i=0}^{\ell-1} 2^i [r_i]_1$ and $[r'] = \sum_{i=0}^{\ell-2} 2^i [r_i]_1$

3 $c \leftarrow \mathsf{Open}_\ell([a] + [r])$

4 $c' = c \bmod 2^{\ell-1}$

5 $[u]_1 \leftarrow \mathsf{BitLT}(c', [r_0]_1, \ldots, [r_{\ell-2}]_1)$

6 $[u] \leftarrow \mathsf{B2A}([u]_1)$ // Can be skipped if $\ell = k$

7 $[a'] = c' - [r'] + 2^{\ell-1}[u]$

8 $[d] = [a] - [a']$

9 $e \leftarrow \mathsf{Open}_\ell([d] + 2^{\ell-1}[b])$ and let $e_{\ell-1}$ be the most significant bit of e

10 $[a_{\ell-1}] = e_{\ell-1} + [b] - 2e_{\ell-1}[b]$

11 **return** $[a_{\ell-1}]$

$(k-1)$-bit integers over ring $\mathbb{Z}_{2^k}$ using a single call to MSB.

There are noteworthy differences in the design of protocols developed for a ring as opposed to original protocols for a field. Certain operations such as prefix multiplication are not available in a ring, and we resort to logarithmic round building blocks when protocols over a field achieve constant round complexity. In the context of comparison, a typical tool for realizing them was truncation (i.e., right shift), the cost of which was linear in the number of bits truncated, but the modulus had to be increased by a statistical security analysis to support such operations. In a ring, on the other hand, there is no significant increase in the ring size, but the communication cost is linear in the bitlength of the ring and not in the bitlength of the truncated portion. This carries different trade-offs, but the availability of faster arithmetic in a ring will still lead to significant savings.

Similar to less-than comparisons, we can determine if a secret shared value is equal to zero (i.e., $a \stackrel{?}{=} 0$) by performing equality testing protocol EQZ based on [62], and is given in Protocol 8. The algorithm invokes the k-ary protocol with the OR operator kOpL from [146] and has a slightly lower round and communication cost than MSB.

Protocol 8: $[b] \leftarrow \mathsf{EQZ}([a])$, where $b = (a \stackrel{?}{=} 0)$

1 Generate ℓ random bits $[r_0]_1, \ldots, [r_{\ell-1}]_1$ in $\mathbb{Z}_2$
2 Compute $[r] = \sum_{i=0}^{\ell-1} 2^i [r_i]_1$
3 $c \leftarrow \mathsf{Open}_\ell([a] + [r])$
4 $[v_i]_1 \leftarrow c_i \oplus [r_i]_1$ for $i = 0, \ldots, \ell - 1$ (in $\mathbb{Z}_2$)
5 $[b]_1 \leftarrow 1 - \mathsf{kOR}([v_0]_1, \ldots, [v_{\ell-1}]_1)$
6 $[b] \leftarrow \mathsf{B2A}([b]_1)$
7 **return** $[b]$

Protocol 9: $([c_{\mathsf{LT}}], [c_{\mathsf{EQ}}]) \leftarrow \mathsf{LT\&EQ}([a], [b])$

// $c_{\mathsf{LT}} = (a \stackrel{?}{<} b)$ and $c_{\mathsf{EQ}} = (a \stackrel{?}{=} b)$
1 Generate ℓ random bits $[r_0]_1, \ldots, [r_{\ell-1}]_1$ in $\mathbb{Z}_2$ and one random bit $[\hat{b}]$ over $\mathbb{Z}_{2^k}$
2 Compute $[r] = \sum_{i=1}^{\ell-1} 2^i [r_i]_1$ and $[r'] = \sum_{i=1}^{\ell-2} 2^i [r_i]_1$
3 $c \leftarrow \mathsf{Open}_\ell(([a] - [b]) + [r])$
4 $c' = c \bmod 2^{\ell-1}$
5 $[v_i]_1 \leftarrow c_i \oplus [r_i]_1$ for $i = 0, \ldots, \ell - 1$ (in $\mathbb{Z}_2$)
6 Parties execute $[u]_1 \leftarrow \mathsf{BitLT}(c', [r_0]_1, \ldots, [r_{\ell-2}]_1)$ and $[v]_1 \leftarrow 1 - \mathsf{kOR}([v_0]_1, \ldots, [v_{\ell-1}]_1)$ simultaneously and in parallel
7 $[u], [c_{\mathsf{EQ}}] \leftarrow \mathsf{B2A}([u]_1^*, [v]_1)$ // $(*)$ Can be skipped if $\ell = k$
8 $[a'] = c' - [r'] + 2^{\ell-1}[u]$
9 $[d] = ([a] - [b]) - [a']$
10 $e \leftarrow \mathsf{Open}_\ell([d] + 2^{\ell-1}[\hat{b}])$ and let $e_{\ell-1}$ be the most significant bit of e
11 $[c_{\mathsf{LT}}] = e_{\ell-1} + [\hat{b}] - 2e_{\ell-1}[\hat{b}]$
12 **return** $([c_{\mathsf{LT}}], [c_{\mathsf{EQ}}])$

Furthermore, we can simultaneously determine if two secret shared values $[a]$ and $[b]$ are less than or equal to one another by combining Protocols 7 and 8 into LT&EQ (given in Protocol 9). Two bits are produced corresponding to $c_{\mathsf{LT}} = (a \stackrel{?}{<} b)$ and $c_{\mathsf{EQ}} = (a \stackrel{?}{=} b)$. The BitLT and kOR operations can be performed in lock-step in Step 6, such that the cost incurred by computing LT and EQ together is minimal. This construction is useful for certain floating-point operations, such as addition and comparisons.

Protocol 10: $[a_0]_1, \ldots, [a_{\ell-1}]_1 \leftarrow \mathsf{BitDec}([a], \ell)$

Generate ℓ random bits $[r_0]_1, \ldots, [r_{\ell-1}]_1$ in $\mathbb{Z}_2$
Compute $[r] = \sum_{i=1}^{\ell-1} 2^i [r_i]_1$
$c \leftarrow \mathsf{Open}_\ell([a] - [r])$
$[a_0]_1, \ldots, [a_{\ell-1}]_1 \leftarrow \mathsf{BitAdd}(c, [r_0]_1, \ldots, [r_{\ell-1}]_1)$
return $[a_0]_1, \ldots, [a_{\ell-1}]_1$

4.2.4 Bit-Decomposition

It is frequently necessary to obtain shares of individual bits $[a_0]_1, \ldots, [a_{\ell-1}]_1$ of a secret $[a]$ such that $a = \sum_{i=0}^{\ell-1} a_i 2^i$. This is achieved through bit decomposition $\mathsf{BitDec}([a], \ell)$ in Protocol 10, and is derived from [47] and [62]. The protocol returns individual bits $[a_i]_1$ shared over $\mathbb{Z}_2$, which is convenient since it is often optimal to perform computation directly on bitwise shares. Optionally, we can run B2A on the output in parallel to recover shares in $\mathbb{Z}_{2^k}$. Correctness follows exactly from the protocol specification.

4.2.5 Private Left Shift

Arithmetic left shift (multiplication by a power of two), or $[x \cdot 2^a]$ by a private number of bits a is of both independent interest and a necessary component of floating-point addition. This operation is realized by privately raising a into the power of 2 (i.e., computing $[2^a]$) and multiplying the result with the input x. We present this functionality in Protocol 11, and it is based on the construction from [16]. For an ℓ-bit input a, we bit-decompose the lower $m = \log \ell$ of a, since raising 2 into the power of a longer than $\log \ell$-bit value is not representable. After converting the individual bits of a from $\mathbb{Z}_2$ to $\mathbb{Z}_{2^k}$, the final result is obtained by computing the product

$$[2^a] \leftarrow \prod_{i=0}^{m-1} 2^{2^i [a_i]} + 1 - [a_i],$$

Protocol 11: $[2^a] \leftarrow \mathsf{Pow2}([a], \ell)$

$m = \lceil \log \ell \rceil$
$[a_0]_1, \ldots, [a_{m-1}]_1 \leftarrow \mathsf{BitDec}([a], m)$
$[a_0], \ldots, [a_{m-1}] \leftarrow \mathsf{B2A}([a_0]_1, \ldots, [a_{m-1}]_1)$
for $i = 0, \ldots, m-1$ **do**
  $[x_i]_k = 2^{2^i}[a_i]_k + 1 - [a_i]_k$
$[x] \leftarrow \mathsf{KOpMul}([x_0], \ldots, [x_{m-1}])$
return $[x]$

which is computed as a tree.[2] The algorithm is sub-logarithmic in the bitlength ℓ.

4.2.6 Truncation and Division

Division can be approached from several directions, depending on whether the divisor is a power of two or an integer, and is public or private.

4.2.6.1 Truncation

Division by a power of two is accomplished using *truncation*. Truncation is independently an essential building block for computation over fixed-point values, simulating fixed-point computation using integer arithmetic, and floating-point operations. The protocol we present, $\mathsf{Trunc}([a], m)$, is a deterministic (exact) solution that combines the approach from [60] with edaBits from [79] and inherits the requirement from [79] that input a is 1 bit shorter than the ring size, i.e., $\mathsf{MSB}(a) = 0$.

The truncation functionality, given as Protocol 13, uses related random values r and $\hat{r}$ (with known bit decompositions) where $r = \sum_{i=0}^{k-1} 2^i r_i$ is a full-size random value and $\hat{r} = \sum_{i=m}^{k-1} 2^i r_i$ is the portion remaining after truncating m bits. These bits can be generated either through [79]'s edaBit construction or with our RandomBit protocol. If edaBits are used (rand = eda), we invoke

[2] The protocol from [16] utilizes prefix multiplication in place of a k-ary operation since it can be performed in a constant number of rounds over a field.

a modified version of the edaBit generation function. Protocol 12 produces the aforementioned values simultaneously. Each $[r]$ and $[\hat{r}]$ is computed as a sum of $t+1$ integers, so we must compensate for two types of carries: (i) addition of the m least significant bits in r will produce carry bits into the next bits which are not accounted for in $\hat{r}$ and (ii) while the carry bits past the k bits are automatically removed in the ring when computing r, these bits remain in $\hat{r}$ due to its shorter length. Since we compute the bitwise representation of r using bitwise addition protocol BitAdd, we can also extract the carry bit into any desired position that is already computed during the addition. The logic of the truncation protocol necessitates the removal of the $(k-1)$th bit. For this reason, we capture carries into the mth and $(k-1)$th positions and denote those bits from the ith call to BitAdd as $\text{cr}_{i,m}$ and $\text{cr}_{i,k-1}$, respectively (line 8). We subsequently convert the $2\log(t+1)$ carry bits and the most significant bit of r, denoted as b_{k-1}, from shares over $\mathbb{Z}_2$ to $\mathbb{Z}_{2^k}$ using B2A.

Up until the end of step 10, the computed truncation a' is probabilistic in nature with an error of at most 1 biased towards the nearest integer to $a/2^m$. However, the recent work of Li et al. [115] determined that probabilistic truncation protocols can leak information about the secret input $[a]$ since the random protection used to protect $[a]$ is also responsible for sampling the probabilistic 1-bit rounding error of the truncated value. This causes the protocol to be not simulatable and thus not t-private. To rectify this, we convert probabilistic truncation to deterministic by determining if a carry occurred when computing $[a]+[r]$ in the lower m bits and subsequently removing it from the output. This is accomplished through a bitwise comparison of the m lower bits of c and $[r]$ (line 11). The final result of the protocol is an exact truncation that leaks no information, thus preserving security according to Definition 1.

We can leverage our public truncation algorithm to construct a protocol where we can truncate a secret $[a]$ by a *private* number of bits $[m]$. We present this functionality in Protocol 14. The

Protocol 12: $([r], [\hat{r}], [b_{k-1}]) \leftarrow \mathsf{edaTrunc}(k, m)$

for $p = 1, \ldots, t+1$ **in parallel do**
  party p samples $r_0^{(p)}, \ldots, r_{k-1}^{(p)} \in \mathbb{Z}_2$ and computes $r^{(p)} = \sum_{j=0}^{k-1} r_j^{(p)} 2^j$ and $\hat{r}^{(p)} = \sum_{j=m}^{k-1} r_j^{(p)} 2^j$
  simultaneously execute $[r^{(p)}] \leftarrow \mathsf{Input}_k(r^{(p)})$, $[\hat{r}^{(p)}] \leftarrow \mathsf{Input}_k(\hat{r}^{(p)})$, and $[r_i^{(p)}]_1 \leftarrow \mathsf{Input}_1(r_i^{(p)})$ for $i = 0, \ldots, k-1$, with p being the input owner
$[r] = \sum_{p=1}^{t+1} [r^{(p)}]$, $[\hat{r}] = \sum_{p=1}^{t+1} [\hat{r}^{(p)}]$
$s = t + 1$
for $i = 1, \ldots, \lceil \log(t+1) \rceil$ **do**
  for $j = 1, \ldots, \lfloor s/2 \rfloor$ **in parallel do**
    $\delta = j + s \cdot (i-1)$
    $\left\langle \left[r_0^{(j)}\right]_1, \ldots, \left[r_{k-1}^{(j)}\right]_1 \right\rangle, [\mathrm{cr}_{\delta,m}]_1, [\mathrm{cr}_{\delta,k-1}]_1$
      $\leftarrow \mathsf{BitAdd}\left(\left\langle \left[r_0^{(2j-1)}\right]_1, \ldots, \left[r_{k-1}^{(2j-1)}\right]_1 \right\rangle, \left\langle \left[r_0^{(2j)}\right]_1, \ldots, [r_{k-1}^{(2j)}]_1 \right\rangle \right)$
    if $s \bmod 2 = 0$ **then**
      $s = s/2$
    else
      $\left\langle \left[r_0^{(\frac{s+1}{2})}\right]_1, \ldots, \left[r_{k-1}^{(\frac{s+1}{2})}\right]_1 \right\rangle = \left\langle \left[r_0^{(s)}\right]_1, \ldots, \left[r_{k-1}^{(s)}\right]_1 \right\rangle$
      $s = (s+1)/2$
$[b_0]_1, \ldots, [b_{k-1}]_1 = \left[r_0^{(1)}\right]_1, \ldots, \left[r_{k-1}^{(1)}\right]_1$
$[b_{k-1}], ([\mathrm{cr}_{\delta,m}], [\mathrm{cr}_{\delta,k-1}])_{\delta=1}^{t} \leftarrow \mathsf{B2A}([b_{k-1}]_1, ([\mathrm{cr}_{\delta,m}], [\mathrm{cr}_{\delta,k-1}])_{\delta=1}^{t})$
$[\hat{r}] = [\hat{r}] - [b_{k-1}] \cdot 2^{k-m-1} + \sum_{\delta=1}^{t} \left([\mathrm{cr}_{\delta,m}] - [\mathrm{cr}_{\delta,k-1}] 2^{k-m-1} \right)$
return $([r], [\hat{r}], [b_{k-1}], [b_0]_1, \ldots, [b_{k-1}]_1)$

algorithm obliviously determines if $[m]$ exceeds the bitlength of the input by computing the comparison $m < \ell$. If so, this implies that we are truncating by more bits than what are available and the output would be zero. We next privately left shift $[a]$ by $\ell - [m]$ bits via Pow2 (Protocol 11) to generate $2^{\ell-m}$. We subsequently truncate the product by ℓ bits to recover the desired output of $[a/2^m]$ and multiply the result by the comparison bit to account for when $m \geq \ell$.

Protocol 13: $[a/2^m] \leftarrow \mathsf{Trunc}([a], m)$, where $\mathsf{MSB}(a) = 0$

// rand is determined prior to starting computation
if rand $=$ eda **then**
  $([r], [\hat{r}], [b_{k-1}], [b_0]_1, \ldots, [b_{k-1}]_1) \leftarrow \mathsf{edaTrunc}(k, m)$
else
  Generate k random bits $[b_0]_1, \ldots, [b_{k-1}]_1$ in $\mathbb{Z}_2$
  $[b_{k-1}] \leftarrow \mathsf{B2A}([b_{k-1}]_1)$
  Compute $[r] = \sum_{i=0}^{k-1} 2^i [b_i]_1$ and $[\hat{r}] = \sum_{i=m}^{k-2} 2^{i-m} [b_i]_1$
$c \leftarrow \mathsf{Open}_\ell([a] + [r])$
$c' = (c/2^m) \bmod 2^{\ell-m-1}$
$[b] = \left(c/2^{\ell-1}\right) + [b_{k-1}] - 2\left(c/2^{\ell-1}\right)[b_{k-1}]$
$[a'] = c' - [\hat{r}] + [b] \cdot 2^{\ell-m-1}$
$[u]_1 \leftarrow \mathsf{BitLT}\left(c_0, \ldots, c_{m-1}, [b_0]_1, \ldots, [b_{m-1}]_1\right)$
$[u] \leftarrow \mathsf{B2A}([u]_1)$
$[a'] = [a'] - [u]$
return $[a']$

Protocol 14: $[a/2^m] \leftarrow \mathsf{TruncS}([a], [m], \ell)$, where $\mathsf{MSB}(a) = 0$

$[b] \leftarrow \mathsf{LT}\left([m], \ell\right)$
$[2^{\ell-m}] \leftarrow \mathsf{Pow2}\left(\ell - [m], \ell\right)$
$[x] \leftarrow [a] \cdot [2^{\ell-m}]$
$[y] \leftarrow \mathsf{Trunc}([x], \ell)$
$[x] \leftarrow [b] \cdot [y]$
return $[x]$

4.2.6.2 Division

If the divisor is *not* a power of two, we must turn to a general-purpose division algorithm. While many conventional algorithms exist to compute the quotient a/b (e.g., digit recurrence, Newton-Raphson), we restrict our focus to Goldschmidt's iterative method [90] for its inherent compatibility with secure computation compared to other methods.

We summarize the method as follows [123]. Let w_0 be the initial approximation of the reciprocal $1/b$, chosen such that it has a relative error $\varepsilon_0 < 1$ (explained below). Letting $a_0 = a, b_0 = b$

and multiplying the numerator and denominator by w_0 transforms the problem into the form

$$\frac{a}{b} = \frac{aw_0}{bw_0} = \frac{a_1}{b_1}.$$

The primary objective of the algorithm is to determine w_i values to multiply the numerator and denominator of the above equation, such that the denominator converges to 1 and, consequently, the numerator towards the desired quotient. This directly translates to iteratively computing the following terms (for $i \geq 1$):

$$a_i = a_{i-1}w_{i-1}, \qquad b_i = b_{i-1}w_{i-1}, \qquad w_i = 2 - b_{i-1}.$$

Denoting $r_i = \prod_{j=0}^{i} w_j$, if we continue multiplying the numerator and denominator by successive w_i terms, we observe

$$\frac{a}{b} = \frac{aw_0 \cdots w_{i-1}}{bw_0 \cdots w_{i-1}} = \frac{a_i}{b_i} = \frac{a_i w_i}{b_i w_i} = \frac{a_{i+1}}{b_{i+1}} = \frac{ar_i}{br_i}.$$

The initial approximation w_0 carries a relative error of $\varepsilon_0 = 1 - bw_0$, and it can be shown (via induction) that each $b_i = 1 - \varepsilon^{2^{i-1}}$ and $w_i = 1 + \varepsilon^{2^{i-1}}$. By requiring $|\varepsilon_0| < 1$, we ensure that after i iterations a_i converges to a/b (with relative error $\varepsilon_0^{2^i}$), and r_i to $1/b$.

The choice of the initial approximation of $1/b$ directly impacts how quickly the algorithm converges to the desired quotient. The conventional approach involves normalizing b to a value c in the range $[0.5, 1)$, and then computing the reciprocal $1/c$ via linear approximation $w_0 = 2.9142 - 2c$ with relative error $\varepsilon_0 < 0.08578$, which provides 3.5 exact bits (see [48] and [78, Chapter 7] for choice of constants). Therefore, for ℓ-bit inputs the algorithm needs $\theta = \left\lceil \log \frac{\ell}{3.5} \right\rceil$ itera-

tions. Alternate approaches [97] use a table lookup operation to compute the initial approximation of $1/b$ in place of the linear approximation described above. However, the cost associated with array access at a private index outweighs the benefit of not needing to normalize the denominator as part of calculating the linear approximation.

Our division algorithm is presented in Protocol 15. The logic follows closely to the fixed-point protocol designed by [48]. We detail each component of the algorithm below and highlight the necessary modifications to ensure compatibility with integer computation over rings.

Before executing the division algorithm, we compute the absolute values of the inputs a and b (denoted as $\hat{a}$ and $\hat{b}$, respectively) such that we operate only on positive values, as required by our truncation protocol. The products on lines 4–6 are all computed in parallel. This conveniently eliminates the need to take the absolute value of $[b]$ (and subsequent re-application of the sign) in Norm. Naturally, the sign must be re-applied to the computed quotient before returning the result in step 23.

As part of determining the initial approximation $1/b$, we first normalize the divisor b to the range $(0.5, 1)$. This is accomplished in Norm (Protocol 17) We circumvent this restriction by obliviously lifting b by its number of bits of precision, effectively converting it to a fixed-point number. Concretely, assume $2^{m-\ell-1} \leq b < 2^{m-\ell}$, for some $m \leq \ell$. We notice by simple algebra that

$$2^{m-\ell-1} \leq b < 2^{m-\ell} \implies 1/2 \leq b2^{\ell-m} < 1,$$

which is our desired quantity. Therefore, the c is the fixed-point representation of the normalized divisor $b2^{\ell-m}$ with a scale $2^{\ell-m}$. We obtain this scale factor by bit-decomposing b and computing the parallel prefix OR of the individual bits. Throughout the remainder of the algorithm, we

operate on fixed-point values with ℓ bits of precision. This equates to periodically lifting certain values by ℓ bits.

After performing normalization, we proceed with determining the initial approximation $w \approx a/b$ in AppRcr (Protocol 16). This consists of computing the quantity $d = ((2.9142) \cdot 2^{\ell} - 2c) \cdot 2^{\ell - m}$ and truncating the result back down to ℓ bits.

At this point, we can at last proceed with computing the quotient $y \approx a/b$. Each multiplication carries a sequential truncation to guarantee interim values do not overflow beyond the ring $\mathbb{Z}_{2^k}$. After computing the initial terms $y_0 = aw$ and $x_0 = 2^{\ell} - bw$, we truncate y to $\ell + \lambda$ bits. The parameter λ controls how many additional bits of the approximated quotient y are maintained throughout iteration. Temporarily extending the interim bitlength of y from 2ℓ to $2\ell + \lambda$ increases the likelihood that the final result of the computation is closer to the true quotient. This concept is not explicitly stated in the original algorithm from [48], but rather mentioned in the description. We opt to include λ as an argument to IntDiv for completeness.

Then, we iteratively compute

$$y_i = y_{i-1} \cdot 2^{\ell} + y_{i-1} x_{i-1},$$

$$x_i = x_{i-1} x_{i-1},$$

with accompanying ℓ-bit truncations for $1 \leq i < (\theta - 1)$, where $\theta = \left\lceil \log \frac{\ell}{3.5} \right\rceil$. After the last iteration, we let $y = y_{\theta} = y_{\theta-1} 2^{\ell} + y_{\theta-1} x_{\theta-1}$ and truncate by $\ell + \lambda$ bits to obtain the preliminary quotient.

In original specification [48], the algorithm terminates at this point (line 17). However, our protocol includes an error correction and rounding procedure *after* this point that is required for

integer division. As stated, the original protocol was designed for fixed-point arithmetic, which preserves any non-integral precision to the right of the decimal point. However, this leads to undesirable behavior for strictly integer inputs, since it leads to correctness and rounding inconsistencies. For example, calculating a/b where b divides a (i.e., $a \bmod b = 0$) would result in the computed quotient being shifted down by 1 from the true value. To rectify this, we perform a series of rounding corrections on lines 18–23 by determining if the product $y \cdot \hat{b}$ is less than $\hat{a}$, and subsequently adjusting y based on the result. This not only ensures the correct outputs are produced, but also consistency with standard integer division in C-like programming languages where we effectively compute $\lfloor a/b \rfloor$.

Contrary to prior operations (besides truncation), division carries additional requirements for the ring size relative to the bitlengths of the inputs, such that the algorithm produces an accurate result. Assuming a and b are ℓ-bit inputs, the computation generates up to $(2\ell + \lambda)$-bit values (encountered on line 12). Therefore, the ring size k required to accommodate division is $k > 2\ell + \lambda$, where the strict greater-than stems from truncation's requirement that the ring is at least 1 bit larger than the bitlength of the secret.

If we supply a public divisor b, the algorithm substantially simplifies. After taking the absolute values of the inputs as before, the primary division component consists solely of multiplying $[a]$ and $[2^{k+\lambda} \cdot b]$ and truncating the product accordingly by $k + \lambda$ bits. We maintain the integer computation-specific steps of the algorithm, i.e., performing the rounding error correction and re-applying the sign to the output.

Correctness of the protocol follows directly from [48] in conjunction with our modifications outlined above to accommodate for integer arithmetic. Security is guaranteed from the building blocks used.

Protocol 15: $[y] \leftarrow \mathsf{IntDiv}([a], [b], \lambda)$

$\alpha = 2^{\ell}, \theta = \left\lceil \log \frac{\ell}{3.5} \right\rceil$
$[a_{\ell-1}] \leftarrow \mathsf{MSB}([a])$
$[b_{\ell-1}] \leftarrow \mathsf{MSB}([b])$
$[\hat{a}] \leftarrow (1 - 2[a_{\ell-1}]) \cdot [a]$
$[\hat{b}] \leftarrow (1 - 2[b_{\ell-1}]) \cdot [b]$
$[\text{sign}] \leftarrow (1 - 2[a_{\ell-1}]) \cdot (1 - 2[b_{\ell-1}])$
$[w] \leftarrow \mathsf{AppRcr}([\hat{b}], \ell)$
$[x] \leftarrow \alpha - [\hat{b}] \cdot [w]$
$[y] \leftarrow [\hat{a}] \cdot [w]$
$[y] \leftarrow \mathsf{Trunc}([y], \ell - \lambda)$
for $i = 1, \ldots, \theta - 1$ **do**
    $[y] \leftarrow [y] \cdot (\alpha + [x])$
    $[x] \leftarrow [x] \cdot [x]$
    $[y] \leftarrow \mathsf{Trunc}([y], \ell)$
    $[x] \leftarrow \mathsf{Trunc}([x], \ell)$
$[y] \leftarrow [y] \cdot (\alpha + [x])$
$[y] \leftarrow \mathsf{Trunc}([y], \ell + \lambda)$
$[\delta] \leftarrow [\hat{a}] - [y] \cdot [\hat{b}]$
$[\delta_{\ell-1}] \leftarrow \mathsf{MSB}([\delta])$
$[y] = [y] + 1 - 2 \cdot [\delta_{\ell-1}]$
$[\delta] \leftarrow [\hat{a}] - [y] \cdot [\hat{b}]$
$[\delta_{\ell-1}] \leftarrow \mathsf{MSB}([\delta])$
$[y] \leftarrow [\text{sign}] \cdot ([y] - [\delta_{\ell-1}])$
return $[y]$

Protocol 16: $[w] \leftarrow \mathsf{AppRcr}([b], \ell)$

$\alpha = (2.9142) \cdot 2^{\ell}$
$([c], [v]) \leftarrow \mathsf{Norm}([b], \ell)$
$[d] = \alpha - 2[c]$
$[w] \leftarrow [d] \cdot [v]$
$[w] \leftarrow \mathsf{Trunc}([w], \ell)$
return $[w]$

We present a comprehensive summary of the performance of all the protocols defined in this section in Table 4.4.

Protocol 17: $([c],[v]) \leftarrow \mathsf{Norm}([b],\ell)$

$[b_0]_1,\ldots,[b_{\ell-1}]_1 \leftarrow \mathsf{BitDec}([b],\ell)$
$[y_{\ell-1}]_1,\ldots,[y_0]_1 \leftarrow \mathsf{PreOR}([b_{\ell-1}]_1,\ldots,[b_0]_1)$
for $i = 0,\ldots,\ell-2$ **do** $[z_i]_1 = [y_i]_1 - [y_{i+1}]_1$
$[z_{\ell-1}]_1 = [y_{\ell-1}]_1$
$[z_0],\ldots,[z_{\ell-1}] \leftarrow \mathsf{B2A}([z_0]_1,\ldots,[z_{\ell-1}]_1)$
$[v] = \sum_{i=0}^{\ell-1} 2^{\ell-i-1}[z_i]$
$[c] \leftarrow [b]\cdot[v]$
return $([c],[v])$

Protocol	Reqs.	Rand.	Precomputation	Active	
				Rounds	Communication
$\mathsf{MSB}([a])$	$(\ell{=}k)$	rB	$(k{+}1)\cdot\mathsf{rB}$	$\log(k{-}1){+}2$	$4nt(k{-}1)$
		eB	$\mathsf{eB}(k,k){+}\mathsf{rB}$		
	$(\ell{<}k)$	rB	$(k{+}1)\cdot\mathsf{rB}$	$\log((\ell{-}1)(t{+}1)){+}3$	$nt\left(k(t^2{-}t{+}1)\right.$ $\left.{+}4(\ell{-}1)\right)$
		eB	$\mathsf{eB}(k,\ell){+}\mathsf{rB}$		
$\mathsf{EQZ}([a])$		rB	$k\cdot\mathsf{rB}$	$\log\left(\ell(t{+}1)\right){+}2$	$nt\left(k(t^2{-}t{+}1)\right.$ $\left.{+}2\ell{-}1\right)$
		eB	$\mathsf{eB}(k,k)$		
$\mathsf{LT\&EQ}([a])$	$(\ell{=}k)$	rB	$(k{+}1)\cdot\mathsf{rB}$	$\log(k(t{+}1)){+}3$	$nt\left(2k(t^2{-}t{+}1)\right.$ $\left.{+}5(k{-}1)\right)$
		eB	$\mathsf{eB}(k,k){+}\mathsf{rB}$		
	$(\ell{<}k)$	rB	$(\ell{+}1)\cdot\mathsf{rB}$	$\log(\ell(t{+}1)){+}3$	$nt\left(k(t^2{-}t{+}1)\right.$ $\left.{+}5(\ell{-}1)\right)$
		eB	$\mathsf{eB}(k,\ell){+}\mathsf{rB}$		
$\mathsf{BitDec}([a],\ell)$		rB	$\ell\cdot\mathsf{rB}$	$\log(\ell){+}1$	$nt\ell(\log(\ell){+}1)$
		eB	$\mathsf{eB}(k,\ell)$		
$\mathsf{Pow2}([a],\ell)$		rB	$\log\ell\cdot\mathsf{rB}$	$2\log\log\ell$ $+\log(t{+}1){+}2$	$nt\left((\log\ell)(\log\log\ell{+}k)\right.$ $\left.{+}k(\log\ell)(t^2{-}t{+}1){-}k\right)$
		eB	$\mathsf{eB}(k,\log\ell)$		
$\mathsf{Trunc}([a],m)$	$(a_\ell{=}0)$	rB	$k\cdot\mathsf{rB}$	$\log\left(m(t{+}1)\right){+}2$	$nt\left(k(t^2{-}t{+}1)\right.$ $\left.{+}2(m{-}1){+}\ell\right)$
		eB	$\mathsf{eBtr}(k,m)$		
$\mathsf{TruncS}([a],[m],\ell)$	$(a_\ell{=}0)$	rB	$(k{+}\ell{+}\log\ell{+}1)\cdot\mathsf{rB}$	$\log\left(\ell(\ell{-}1)(t{+}1)^3\right)$ $2\log\log\ell{+}9$	$nt\left((\log\ell)\log\log\ell{+}k(\log\ell{+}1)\right.$ $\left.{+}k(\log\ell{+}2)(t^2{-}t{+}1){+}7\ell{+}6\right.$
		eB	$\mathsf{eB}(k,k){+}\mathsf{eB}(k,\log\ell){+}\mathsf{rB}{+}\mathsf{eBtr}(k,\ell)$		
$\mathsf{Norm}([b],\ell)$		rB	$\ell\cdot\mathsf{rB}$	$\log\left(\ell^2(t{+}1)\right){+}3$	$nt(k(t^2{-}t{+}1){+}k$ $+\ell(\log\ell^{3/2}{+}1))$
		eB	$\mathsf{eB}(k,\ell)$		
$\mathsf{AppRcr}([b],\ell)$		rB	$(k{+}\ell)\cdot\mathsf{rB}$	$\log\left(\ell^3(t{+}1)^2\right){+}6$	$nt(2k(t^2{-}t{+}1){+}2k$ $+\ell(\log\ell^{3/2}{+}5){-}2)$
		eB	$\mathsf{eB}(k,\ell){+}\mathsf{eBtr}(k,\ell)$		
$\mathsf{IntDiv}([a],[b],\lambda)$		rB	$(7k{+}2k(\theta{-}1){+}\ell{+}4)\cdot\mathsf{rB}$	$\log\left((\ell^2{-}\lambda^2)\ell^{\theta+2}(t{+}1)^{\theta+3}\right)$ $+\log\left((k{-}1)^3\right){+}21{+}3\theta$	$nt(k(3{+}2\theta)(t^2{-}t{+}1){+}\ell\log\ell^{3/2}$ $+2(\theta{-}1)(4\ell{+}\lambda{-}k{-}2){+}13\ell{+}\lambda{-}22)$
		eB	$4\cdot\mathsf{eB}(k,k){+}\mathsf{eB}(k,\ell){+}4\cdot\mathsf{rB}{+}\mathsf{eBtr}(k,\ell\pm\lambda){+}(2\theta{-}1)\cdot\mathsf{eBtr}(k,\ell)$		

Table 4.4: Composite protocol performance with communication measured in the total number of bits sent across all parties. For convenience, $\mathsf{eB}(k,\ell)=\mathsf{edaBit}(k,\ell)$, $\mathsf{rB}=\mathsf{RandBit}()$, and $\mathsf{eBtr}(k,m)=\mathsf{edaTrunc}(k,m)$. $\theta{=}\lceil\log(\ell/3.5)\rceil$ from IntDiv (Protocol 15).

4.3 Performance Evaluation

We implemented the protocols described in this work and evaluate their performance in this section. We run micro-benchmarks to evaluate the individual operations. Supplementary benchmarks for neural network applications can be found in Appendix B and feature a novel optimization trick specific to quantized neural networks.

The implementation was done in C++ and is available at [7]. We use AES from the OpenSSL cryptographic library [3] to instantiate the PRF and also to implement secure communication channels between each pair of computational parties. We report the average execution time of 1000 executions for the micro-benchmark experiments. The runtimes are also averaged across the computation parties. All experiments use identical 2.4 GHz virtual machines with 26 GB of RAM. They were connected via 10 Gbps Ethernet links, which we throttled to 1 Gbps using the `tc` command. Two-way latency was measured to be 0.106 ms. All experiments are single-threaded.

We report the performance of individual operations such as multiplication, matrix multiplication, random bit generation (RandBit[3] and edaBit), comparisons (MSB), and binary-to-arithmetic conversion (B2A). Unless otherwise noted, the experiments used two bitlengths, $k = 30$ and $k = 60$, which allows us to use the `uint32_t` and `uint64_t` integer types, respectively, to implement ring operations. Tables 4.5 and 4.6 report the performance of multiplication and matrix multiplication, respectively. As we strive to measure performance improvement when we switch computation from a field to a ring, we compare the performance of our protocols to those using Shamir SS in the same setting (i.e., semi-honest security with honest majority) using PICCO implementation [165] with recent improvements to multiplication from [35]. The field size is set

[3]The RandBit results presented in this section use the protocol presented in [22], which is based on the construction from [62].

Protocol			Batch Size							Comm.
			1	10	10^2	10^3	10^4	10^5	10^6	
3PC	$\mathbb{F}$G	(30 bits)	0.081	0.0893	0.245	1.47	12.1	120	1,236	4
		(60 bits)	0.082	0.0912	0.255	1.61	12.9	127	1,289	8
	$\mathcal{R}$	(30 bits)	0.075	0.079	0.097	0.153	0.606	5.87	59.6	4
		(60 bits)	0.075	0.076	0.108	0.320	1.096	9.68	113	8
5PC	$\mathbb{F}$G	(30 bits)	0.124	0.158	0.384	2.37	16.8	159	1,550	8
		(60 bits)	0.129	0.167	0.439	2.45	17.9	173	1,669	16
	$\mathbb{F}$D	(30 bits)	0.224	0.267	0.836	3.74	34.1	235	2,227	6.4*
		(60 bits)	0.229	0.278	0.924	4.01	36.4	254	2,436	12.8*
	$\mathcal{R}$	(30 bits)	0.139	0.141	0.160	0.52	3.96	38.2	377	8
		(60 bits)	0.153	0.155	0.211	0.723	5.40	59.5	579	16
7PC	$\mathbb{F}$G	(30 bits)	0.168	0.198	0.497	3.17	24.5	238	2,353	12
		(60 bits)	0.174	0.224	0.541	3.47	27.7	257	2,520	24
	$\mathbb{F}$D	(30 bits)	0.275	0.327	1.18	7.69	60.4	502	4,829	6.9*
		(60 bits)	0.281	0.354	1.34	8.01	67.8	534	5,186	13.7*
	$\mathcal{R}$	(30 bits)	0.246	0.294	0.469	2.71	23.8	266	2,536	12
		(60 bits)	0.269	0.268	0.555	3.39	33.2	365	3,490	24

Table 4.5: Runtime of multiplication protocols in ms and communication is per party, per operation in bytes (* means average for asymmetric communication patterns). $\mathbb{F}$G and $\mathbb{F}$D refer to the optimized GRR and DN field multiplication from [35], respectively, and $\mathcal{R}$ is our ring realization. 30 and 60 are integer bitlengths.

to accommodate 30- and 60-bit integers. The "batch size" denotes how many operations were executed at the same time in a single batch.

We measure runtime and communication with a number of parties ranging from 3 to 7. For field multiplication, we measure the performance of two variants: GRR-based with higher asymptotic communication and 1 round ($\mathbb{F}$G) and DN-based with lower asymptotic communication and 2 rounds ($\mathbb{F}$D) as described in [35]. The former is strictly better in the three-party setting. The latter, despite its lower communication, does not lead to better performance as the number of parties increases as it internally relies on RSS. However, the difference in performance of the two variants is not substantial enough to play a major role in larger computations, as is demonstrated in Table 4.6. We therefore proceed with $\mathbb{F}$G with 3 parties and $\mathbb{F}$D with 5–7 parties in other experiments

Protocol			Matrix Dimensions			
			10×10	100×100	500×500	1000×1000
3PC	$\mathbb{F}$	(30 bits)	0.318	91.6	1,025	8,289
	$\mathbb{F}$	(60 bits)	0.319	94.2	1,187	8,723
	$\mathcal{R}$	(30 bits)	0.187	2.83	212	1,567
	$\mathcal{R}$	(60 bits)	0.288	3.82	226	1,638
5PC	$\mathbb{F}\mathrm{G}$	(30 bits)	0.457	95.2	1,145	8,927
	$\mathbb{F}\mathrm{G}$	(60 bits)	0.462	97.9	1,321	10,134
	$\mathbb{F}\mathrm{D}$	(30 bits)	1.07	97.4	1,273	9,995
	$\mathbb{F}\mathrm{D}$	(60 bits)	1.09	102	1,493	11,964
	$\mathcal{R}$	(30 bits)	0.219	11.5	720	5,224
	$\mathcal{R}$	(60 bits)	0.202	12.5	813	5,939
7PC	$\mathbb{F}\mathrm{G}$	(30 bits)	0.891	97.7	1,272	9,953
	$\mathbb{F}\mathrm{G}$	(60 bits)	0.904	101	1,478	10,864
	$\mathbb{F}\mathrm{D}$	(30 bits)	1.29	99.8	1,483	11,569
	$\mathbb{F}\mathrm{D}$	(60 bits)	1.35	104	1,536	13,742
	$\mathcal{R}$	(30 bits)	0.514	48.0	5,880	48,793
	$\mathcal{R}$	(60 bits)	0.591	59.0	7,509	71,234

Table 4.6: Runtime of matrix multiplication in ms.

where multiplication is used.

From Table 4.5 we observe that our RSS performance is up to 20 times faster with a sufficiently large batch size in the 3-party setting compared to the field and some performance advantage is maintained even with 7 parties despite the need to compute with a much larger number of shares. Note that the performance gain is due to faster instructions because communication is comparable across different variants. This indicates that using native CPU instructions for secure arithmetic has a remarkable advantage.

Matrix multiplication in Table 4.6 is performed in a single round using the necessary number of dot-products. Because local work is the bottleneck, we see performance improvement by up to a factor of 32.3 after switching to a ring with 3 parties. Performance improvement with 5 parties is by up to a factor of 8.3 and up to a factor of 2 with 7 parties. The ring performance is superior for all configurations evaluated except for the two largest matrices with 7 parties.

Protocol			Batch Size							Comm.
			1	10	10^2	10^3	10^4	10^5	10^6	
3PC	$\mathbb{F}$	(30 bits)	0.104	0.158	0.457	2.87	25.4	259	2,637	20
	$\mathbb{F}$	(60 bits)	0.107	0.164	0.546	3.47	32.8	336	3,480	28
	$\mathcal{R}$	(30 bits)	0.124	0.111	0.156	0.330	2.37	21.8	249	8
	$\mathcal{R}$	(60 bits)	0.112	0.124	0.170	0.555	4.57	43.9	477	16
5PC	$\mathbb{F}$	(30 bits)	0.175	0.281	0.815	5.97	50.8	506	4,985	40
	$\mathbb{F}$	(60 bits)	0.171	0.291	0.869	6.75	65.4	66.1	6,794	56
	$\mathcal{R}$	(30 bits)	0.169	0.178	0.234	0.595	4.50	45.9	468	16
	$\mathcal{R}$	(60 bits)	0.262	0.244	0.356	1.252	8.39	88.1	854	32
7PC	$\mathbb{F}$	(30 bits)	0.249	0.369	1.15	8.14	70.6	684	6,842	60
	$\mathbb{F}$	(60 bits)	0.264	0.412	1.34	9.42	84.9	824	8,251	84
	$\mathcal{R}$	(30 bits)	0.255	0.268	0.472	1.53	10.4	117	1,134	24
	$\mathcal{R}$	(60 bits)	0.237	0.288	0.508	2.15	18.3	217	2,092	48

Table 4.7: Runtime of RandBit protocols in ms and communication is per party, per operation in bytes.

Protocol			Batch Size							Comm.
			1	10	10^2	10^3	10^4	10^5	10^6	
3PC	[9]	(32 bits)	19.7	15.9	16.2	16.7	20.0	138	1,368	–
	[9]	(64 bits)	22.8	25.5	25.2	24.4	30.6	254	2,201	–
	$\mathcal{R}$	(30 bits)	0.564	0.577	0.832	2.96	20.5	207	1,978	32
	$\mathcal{R}$	(60 bits)	0.622	0.737	1.111	5.66	43.2	405	4,175	68

Table 4.8: Runtime of edaBit protocols in ms compared to MP-SPDZ implementation. Communication for our solution is per party, per operation in bytes.

Tables 4.7 and 4.8 provide random bit generation results. To support k-bit integers, ring-based RandBit following [62]'s construction requires ring $\mathbb{Z}_{2^{k+2}}$. Field-based RandBit from [47] does not increase the field size; however, all uses of RandBit we are aware of are for operations such as comparisons that utilize statistical hiding and, as a result, increase the field size by a statistical security parameter κ (typically set to 48 in implementations). For this reason, our field-based RandBit and MSB benchmarks utilize 79- and 109-bit fields. Both versions of RandBit in Table 4.7 communicate the same number of field or ring elements; however, the performance gain of the ring version grows as we increase the batch size, reaching 10 to 12-fold improvement with 3 and

5 parties and indicating that local field-based computation is the bottleneck. This is in large part due to the need to perform modulo exponentiations (see [47]). That is, even though the field-based RandBit also relies on RSS, other non-RSS computation (such as modulo exponentiation) is significant and the overall slowdown with the number of parties is not as large. In the 7-party setting, the improvement of the ring-based variant is by up to a factor of 6.

The concept of edaBit is recent and for that reason in Table 4.8 we compare our implementation to that reported in the original publication [79], available through MP-SPDZ repository [9]. Note that each edaBit corresponds to generating k random bits together with the corresponding k-bit random integer. It is clear from the table that MP-SPDZ's implementation is optimized for large sizes and fast networks. In particular, it gives comparable runtime for batches of size 1 and 1,000. For the same reason, we were unable to accurately report communication cost per operation from the experiments and refer the reader to the original publication [79] for that information. Note that the times we measured for MP-SPDZ are very different from those originally provided in [79], which reported the ability to generate 7.18 million 64-bit edaBits per second. This is over 15 times faster than the fastest time per operation we record and stems from the differences in hardware. In particular, experiments in [79] were run multithreaded on powerful AWS `c5.9xlarge` instances with 36 cores and a 10 Gbps link. This distinction highlights the need to reproduce experiments on similar hardware to draw meaningful comparisons about the performance of different algorithms.

Table 4.9 reports performance of multiple MSB protocols: (i) field-based protocol from [47] using PICCO's implementation with optimizations from [35], our ring implementations (ii) using RandBit and (iii) using edaBit, and ring-based implementations from MP-SPDZ [9] (iv) using edaBit and (v) using ABY3. The last two support only three-party computation.

The gap between the first two shows performance improvement due to switching from field-

	Protocol		Batch Size							Comm.
			1	10	10^2	10^3	10^4	10^5	10^6	
3PC	$\mathbb{F}$	(30 bits)	1.29	3.71	23.7	206	2,051	21.8s	222s	624
	$\mathbb{F}$	(60 bits)	1.97	7.51	54.7	471	4,654	46.7s	487s	864
	$\mathcal{R}$+rB	(30 bits)	0.71	0.74	1.54	9.23	88.7	0.85s	8.25s	265
	$\mathcal{R}$+rB	(60 bits)	0.76	1.01	3.92	29.2	322	3.04s	30.0s	1009
	$\mathcal{R}$+eB	(30 bits)	1.23	1.24	1.51	4.14	30.7	0.27s	2.77s	57
	$\mathcal{R}$+eB	(60 bits)	1.31	1.46	1.88	7.88	60.6	0.56s	5.71s	117
	[9]+eB	(32 bits)	23.3	23.1	22.9	23.5	27.3	0.18s	1.31s	–
	[9]+eB	(64 bits)	34.2	31.6	33.4	32.5	35.9	0.25s	2.15s	–
	[9]+ABY3	(32 bits)	8.51	8.97	9.05	13.6	52.1	0.39s	3.66s	–
	[9]+ABY3	(64 bits)	9.09	9.06	8.88	14.2	58.5	0.41s	3.87s	–
5PC	$\mathbb{F}$	(30 bits)	2.12	6.17	37.5	349	3,219	32.2s	333s	1248
	$\mathbb{F}$	(60 bits)	3.32	11.9	84.0	738	7,021	68.8s	701s	1728
	$\mathcal{R}$+rB	(30 bits)	1.28	1.37	2.98	18.4	197	1.82s	18.6s	530
	$\mathcal{R}$+rB	(60 bits)	1.65	2.07	7.38	63.7	644	6.10s	60.5s	2018
	$\mathcal{R}$+eB	(30 bits)	3.04	3.20	4.84	24.1	203	1.97s	19.1s	162
	$\mathcal{R}$+eB	(60 bits)	3.96	3.93	8.77	48.0	422	4.24s	41.1s	338
7PC	$\mathbb{F}$	(30 bits)	3.08	9.14	48.4	452	4.42s	43.2s	447s	1872
	$\mathbb{F}$	(60 bits)	4.55	13.1	101	943	9.36s	94.2s	959s	2592
	$\mathcal{R}$+rB	(30 bits)	2.05	2.41	7.10	56.1	0.61s	5.95s	65.4s	795
	$\mathcal{R}$+rB	(60 bits)	2.39	3.53	17.5	183	1.75s	17.6s	179s	3027
	$\mathcal{R}$+eB	(30 bits)	5.17	6.35	21.9	173	1.63s	16.8s	165s	316
	$\mathcal{R}$+eB	(60 bits)	5.99	8.87	41.0	371	3.57s	36.3s	356s	663

Table 4.9: Runtime of MSB protocols in ms unless marked otherwise. Communication is per party, per operation in bytes. rB and eB indicate variants using RandBit and edaBit, respectively.

based to ring-based arithmetic. Both of them make a linear in k number of calls to RandBit, but our implementation executes BitLT over $\mathbb{Z}_2$, while field-based uses a fixed field for all operations. As a result, our ring RandBit-based MSB is up to 26.9 times faster than the field version with 3 parties, up to 18.9 times with 5 parties, and up to 8.1 times with 7 parties.

If we compare our RandBit and edaBit MSB implementations, the use of the edaBit version becomes advantageous starting from batch sizes of 100 with 3 parties, 1000–10000 with 5 parties, but is not beneficial with 7 parties. This can be explained by the need to perform a larger number of bitwise additions during edaBit generation as the number of computational parties increases.

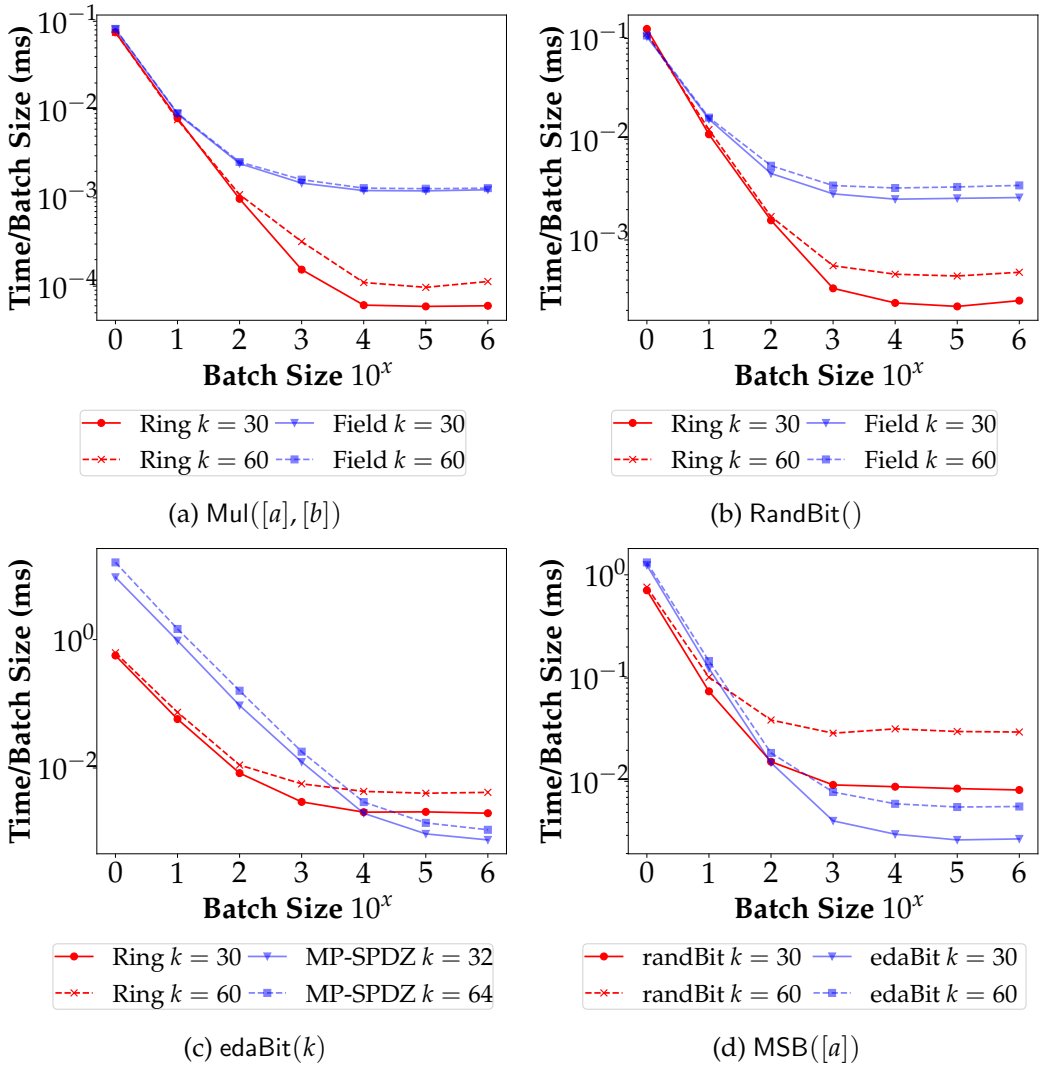


Figure 4.4: Three-party micro-benchmarks results.

MP-SPDZ's edaBit-based implementation in the three-party setting generally took longer to run than our edaBit-based implementation until the batch size became large. As explained earlier, this is due to different performance emphases in the two implementations. ABY3 (three-party) implementation is slower than what we obtain except for the largest batch sizes with the longer bitlengths.

We also visualize time per operation with variable batch sizes in Figure 4.4 using three parties.

Multiplication and RandBit sub-figures compare ring versus field protocols, indicating a substantial gap as expected; edaBit sub-figure compares our and MP-SPDZ implementations in the same setting; and MSB sub-figure compares RandBit and edaBit variants.

It is also informative to compare our field versus ring results with those of SPDZ. While SPDZ [65] and its ring version SPD$\mathbb{Z}_{2^k}$ [58, 62] use a much stronger adversarial model and different type of SS, we would like to know whether similar savings are achievable in different settings. [62] reports that performance improved by a factor of 4.6–4.9 for multiplication and by a factor of 5.2–6.0 for RandBit-based comparison on a 1Gbps LAN. The results are only provided as throughput improvement and do not report different batch sizes. In our experiments, we observed greater improvements, up to 20 times for multiplication and up to 26.9 improvement for MSB. This may be explained by the fact that our techniques are more lightweight and perhaps switching to faster arithmetic makes less of an impact in the SPDZ setting.

Table 4.10 compares the performance of our B2A construction (Protocol 4) to the RandBit-based approach from [62]. Unlike previous experiments in this section, these experiments are conducted with ring sizes of $k = 32$ and $k = 64$. This leads to the RandBit-based versions temporarily operating over the rings $k = 34$ and $k = 66$ (which necessitate switching to the integer types `uint64_t` and `unsigned __int128`, respectively). Our B2A approach for 3 parties consistently outperforms the RandBit-based approach in terms of raw performance and total communication, echoing the results reported for the 3-party-specific construction from [34]. The performance advantage of our approach is maintained throughout all 5-party experiments and even for some of the 7-party batch sizes, all while no longer needing to temporarily operate over a larger ring. We note that the 3- and 5-party results achieve faster runtimes than what is reported for our single-round multiplication in Table 4.5. This is a byproduct of the asymmetric nature of our B2A protocol, coupled with the

	Protocol		Batch Size							Comm.
			1	10	10^2	10^3	10^4	10^5	10^6	
3PC	[62]	(32 bits)	0.174	0.217	0.217	0.564	3.45	33.8	392	14
	[62]	(64 bits)	0.248	0.255	0.293	1.03	7.72	75.4	814	26
	$\mathcal{R}$	(32 bits)	0.069	0.070	0.073	0.120	0.921	7.60	118	8
	$\mathcal{R}$	(64 bits)	0.071	0.074	0.075	0.157	1.52	15.8	240	16
5PC	[62]	(32 bits)	0.427	0.449	0.496	1.30	7.74	84.0	840	28
	[62]	(64 bits)	0.473	0.484	0.513	1.58	12.6	142	1,467	52
	$\mathcal{R}$	(32 bits)	0.229	0.226	0.262	0.715	5.20	55.1	672	24
	$\mathcal{R}$	(64 bits)	0.238	0.239	0.273	1.11	9.19	116	1,268	48
7PC	[62]	(32 bits)	0.693	0.694	0.790	2.13	24.4	252	2,626	42
	[62]	(64 bits)	0.678	0.692	0.877	3.42	48.2	497	4,900	78
	$\mathcal{R}$	(32 bits)	0.368	0.441	0.699	4.35	43.1	451	4,590	48
	$\mathcal{R}$	(64 bits)	0.374	0.448	1.04	7.44	80.2	840	8,491	96

Table 4.10: Runtime of B2A protocols in ms and communication is per party, per operation in bytes.

specialized (cheaper) multiplication MulSparse. If a party or parties are not expecting to receive computed shares from other participants(s) during the Input phase, they can immediately begin the XOR component of the protocol. This phenomenon is most observable for 3 and 5 parties, but less for 7 since Mul and MulSparse are performed in parallel in the first layer of the tree.

Chapter 5

Floating-Point Protocols

In this chapter, we extend our RSS framework to support floating-point operations by leveraging the integer protocols developed in Chapter 4.

5.1 Floating-Point Background

Floating-point is the *de facto* means of representing real numbers on all modern computing devices due to their improved precision over fixed-point or integer representations. The IEEE 754 standard [6] established the representation with p exponent bits and q mantissa (significand) bits. Values are stored in their normalized form, i.e., the leading bit is nonzero. As a result, this leading bit is not explicitly stored since its value is always 1 for a normalized number, effectively supplying $q+1$ bits of precision.

Following the conventions of [16] and [140], we denote a floating-point number $\tilde{a}$, parameterized by $p, q \in \mathbb{Z}_{>0}$, as the tuple $\tilde{a} = (z, s, e, m)$ and corresponds to the value

$$\tilde{a} = (1-z) \cdot (1-2s) \cdot 2^e \cdot m$$

where $m \in \left[2^q, 2^{q+1}\right)$ is the unsigned fixed-point *mantissa* (significand) with scale q, $e \in [-2^{p-1} + 1, 2^{p-1}]$ is the p-bit unbiased signed *exponent*, $z \in \{0, 1\}$, which is the *zero bit* (set if $\tilde{a} = 0$), and $s \in \{0, 1\}$ is the *sign bit* (set if $\tilde{a} \leq 0$). We use the notation $\tilde{a}.z$ to indicate the z component of the $\tilde{a}$ tuple (and equivalently for the s, e, and m components).

The IEEE 754 floating-point specification outlines several exceptions for floating-point numbers [6, 89]: *invalid operation* (mathematically undefined behavior, e.g. square root of a negative number); *division by zero* (e.g. dividing by zero, logarithm of zero) *overflow* and *underflow* (exceeding the maximum and minimum representable values, respectively), and *inexact* (the rounded result of a floating-point operation is not exact). The inexact exception has the lowest priority and is only triggered when the output differs from the infinitely precise result. This is beyond the scope of secure floating-point frameworks [16, 140] and as such can be ignored.

Checking for overflows and underflows can be conducted at the end of a floating-point protocol's execution, and is accomplished by comparing the exponent to the largest and smallest representable values. We omit this checking explicitly from our protocols since they can be enabled on an *ad hoc* basis. Similarly, since we are exclusively considering basic arithmetic operations, the only specific exception we need to check for is division by zero. This is realized by adding the [Error] flag which gets set if the zero bit of the divisor is also set [16]. The approach for checking if the error flag is set is dictated by the security requirements specified by the computation designer. The computational parties may reconstruct the flag after each operation, after several operations, or at the end of the computation alongside the final output of the computation to individuals entitled to learn the result [83].

Protocol 18: $([a/2^m], [a/2^{(m-2)}]) \leftarrow \mathsf{TruncRNTE}([a], m, \mathsf{rand})$, where $\mathsf{MSB}(a) = 0$

```
// rand is determined prior to starting computation
```
if rand = eda **then**
  $([r], [\hat{r}], [\hat{\hat{r}}], [b_{k-1}], [b_0]_1, \ldots, [b_{k-1}]_1) \leftarrow \mathsf{edaTruncRNTE}(k, m)$
else
  Generate k random bits $[b_0]_1, \ldots, [b_{k-1}]_1$ in $\mathbb{Z}_2$
  $[b_{k-1}] \leftarrow \mathsf{B2A}([b_{k-1}]_1)$
  Compute $[r] = \sum_{i=0}^{k-1} 2^i [b_i]_1$, $[\hat{r}] = \sum_{i=m}^{k-2} 2^{i-m} [b_i]_1$, and $[\hat{\hat{r}}] = \sum_{i=(m-2)}^{k-2} 2^{i-(m-2)} [b_i]_1$
$c \leftarrow \mathsf{Open}_\ell([a] + [r])$
$c' = (c/2^m) \bmod 2^{\ell-m-1}$
$d' = \left(c/2^{m-2}\right) \bmod 2^{\ell-m-3}$
$[b] = \left(c/2^{\ell-1}\right) + [b_{k-1}] - 2\left(c/2^{\ell-1}\right)[b_{k-1}]$
$[a'] = c' - [\hat{r}] + [b] \cdot 2^{\ell-m-1}$
$[a''] = d' - [\hat{\hat{r}}] + [b] \cdot 2^{\ell-m-3}$
$([u]_1, [v]_1) \leftarrow \mathsf{BitLT}(c_0, \ldots, c_{m-1}, [b_0]_1, \ldots, [b_{m-1}]_1)$, where $[v]_1$ is the result for the first $m-2$ bits
$[w_i]_1 \leftarrow c_i \oplus [b_i]_1$ for $i = 0, \ldots, m-3$ (in $\mathbb{Z}_2$)
$[w]_1 \leftarrow \mathsf{kOR}([w_0]_1, \ldots, [w_{m-3}]_1)$ `// Checking if any of` $m-2$ `lower bits are set`
Let $[a_0'']_1 = (c_0 \oplus [b_0]_1) \oplus [v]_1$ be the least significant bit of $[a'']$
$([u], [v], [w], [a_0'']) \leftarrow \mathsf{B2A}\left([u]_1, [v]_1, [w]_1, [a_0'']_1\right)$
$[a'] = [a'] - [u]$
$[a_s''] = [a''] - [v] + ([w] - [a_0''])$ `// Removing carry and setting sticky bit`
return $([a'], [a_s''])$

5.2 Rounding and Truncation

Rounding (and consequently, truncation) is an integral component of all floating-point arithmetic as a means for maintaining precision under repeated calculations [142]. The default rounding mode in the IEEE 754 standard is *round nearest, ties to even* (or, equivalently, rounding half to even), which rounds to the nearest value. If the number falls halfway between the representable numbers (a "tie"), then it is rounded to the nearest value with an even least significant bit.

Conforming to this rounding specification requires two additional protocols, the first of which is a modified version of our original truncation algorithm that we denote as $\mathsf{TruncRNTE}([a], m)$

(Protocol 18). It deterministically truncates the input by m bits and $m-2$ bits in parallel, returning $[a']$ and $[a''_s]$, respectively. The protocol carries the additional functionality setting the least significant bit of the shorter truncation to 1 (referred to as the "sticky bit") if any of the lower $m-2$ bits of the input are set. This is determined by a call to kOR on line 15, and we subsequently apply the bit to the interim shorter truncation result (line 19) If edaBit is used as the means for generating shared randomness, we invoke a modified version of our edaTrunc procedure (denoted by edaTruncRNTE) which includes steps for generating the shorter shared random value $[\hat{\hat{r}}]$ (see Appendix A.2 for the full protocol description).

The core rounding functionality $\mathsf{RNTE}([a], m)$ is performed in Protocol 19. After obtaining the two truncations ($[a']$ and $[a''_s]$), we must determine the appropriate rounding of the input. Denote the three least significant bits of the shorter truncation as a_2, a_1, and a_0 as the guard, round, and sticky bits (respectively). As prescribed by the rounding mode, we round up when a_1 and a_0 are set, indicating the round-up result is closer than the round-down result. In the event of a tie where a_1 is set but a_0 is zero, we round up to even if a_2 is set (the number is odd). This translates to the boolean expression $a_1 \wedge (a_2 \vee a_0)$.

From here, we can proceed in one of two directions: we can either perform bit-decomposition on the three least significant bits and directly evaluate the expression in $\mathbb{Z}_2$, or encode the eight possible values of the expression into table entries and perform a lookup operation (such as ArrayRead from [36]). While SecFloat uses the latter approach, the choice is more nuanced in our setting depending on whether precomputational performance is valued over active (online) computation, as well as on the shared randomness generation technique (edaBit versus RandBit). For the sake of simplicity and consistency with the protocols presented in Section 4.2, we proceed with the bit-decomposition version, as private array access is outside the scope of this dissertation. We

Protocol 19: $([a/2^m])_{\text{round}} \leftarrow \mathsf{RNTE}([a], m)$, where the output is correctly rounded

1 $([a'], [a''_s]) \leftarrow \mathsf{TruncRNTE}([a], m, \mathsf{rand})$
2 $[a_0]_1, [a_1]_1, [a_2]_1 \leftarrow \mathsf{BitDec}([a''_s], 3)$
3 $[b]_1 \leftarrow [a_1]_1 \cdot ([a_2]_1 + [a_0]_1 - [a_2]_1 \cdot [a_0]_1)$ // $a_1 \wedge (a_2 \vee a_0)$ in $\mathbb{Z}_2$
4 $[b] \leftarrow \mathsf{B2A}([b]_1)$
5 **return** $[a'] + [b]$

Protocol	Rand.	Precomputation	Active	
			Rounds	Communication
$\mathsf{Trunc}([a], m)$	rB	$k \cdot \mathsf{rB}$	$\log(m(t+1)) + 2$	$nt\left(k(t^2-t+1)\right.$ $\left.+2(m-1)+\ell\right)$
	eB	$\mathsf{eBtr}(k, m)$		
$\mathsf{TruncRNTE}([a], m)$	rB	$k \cdot \mathsf{rB}$	$\log(m(t+1)) + 3$	$nt\left(4k(t^2-t+1)\right.$ $\left.+3(m-1)+\ell+1\right)$
	eB	$\mathsf{eBR}(k, m)$		
$\mathsf{RNTE}([a], m)$	rB	$(3+k) \cdot \mathsf{rB}$	$\log(3m(t+1)^2) + 7$	$nt\left(5k(t^2-t+1)+2k\right.$ $\left.+3m+\ell+6\right)$
	eB	$\mathsf{eBR}(k, m) + \mathsf{eB}(3, 3)$		

Table 5.1: Performance of floating-point-specific truncation and rounding protocols where we require $a_\ell = 0$. For convenience, $\mathsf{eB}(k, \ell) = \mathsf{edaBit}(k, \ell)$, $\mathsf{rB} = \mathsf{RandBit}()$, $\mathsf{eBtr}(k, m) = \mathsf{edaTrunc}(k, m)$, and $\mathsf{eBR}(k, m) = \mathsf{edaTruncRNTE}(k, m)$.

refer the reader to [36] for more information about private array access constructions.

It is worth acknowledging that many works [16, 120, 101, 43, 45, 44] have opted to forego RNTE in favor of the significantly cheaper alternative of performing a simple truncation (i.e., directed rounding towards $-\infty$). This is a valid rounding rule specified by the IEEE 754 standard, but not the default since it does not eliminate bias under repeated addition or subtraction of independent numbers [142]. Other works have considered truncating fewer bit(s) [144] altogether, such that the resultant mantissa is slightly longer than that of the inputs. In the interest of flexibility, we indicate where in the protocol's execution one may choose a truncation method best suited for their specific use case.

We compute the performance of the TruncRNTE and RNTE protocols in Table 5.1 and provide the performance of our original truncation algorithm (Protocol 13) for the sake of comparison.

Protocol 20: $[\tilde{c}] \leftarrow \mathsf{FLMul}([\tilde{a}], [\tilde{b}])$

1 $[m] \leftarrow [\tilde{a}.m] \cdot [\tilde{b}.m]$
2 $[m_q] \leftarrow \mathsf{RNTE}([m], q)$
3 $[m_{q+1}] \leftarrow \mathsf{RNTE}([m], q+1)$
4 $[b] \leftarrow \mathsf{LT}([m_q], 2^{q+1})$
5 $[\tilde{c}.m] \leftarrow [b] \cdot [m_q] + (1 - [b]) \cdot [m_{q+1}]$
6 $[\tilde{c}.z] \leftarrow [\tilde{a}.z] + [\tilde{b}.z] - [\tilde{a}.z] \cdot [\tilde{b}.z]$ // OR
7 $[\tilde{c}.e] \leftarrow (1 - [\tilde{c}.z]) \cdot ([\tilde{a}.e] + [\tilde{b}.e] + q + 1 - [b])$
8 $[\tilde{c}.s] \leftarrow [\tilde{a}.s] + [\tilde{b}.s] - 2[\tilde{a}.s] \cdot [\tilde{b}.s]$ // XOR
9 **return** $[\tilde{c}]$

5.3 Multiplication

Our protocol for multiplying two floating-point numbers $[\tilde{a}] \cdot [\tilde{b}]$ is presented in Protocol 20. We first multiply the mantissas to obtain a $2q+2$-bit fixed-point significand m. This product must be rounded/truncated by either q or $q+1$ bits based on whether the most significant bit of m is set. For RNTE, we perform the roundings in parallel and then obliviously choose the correct mantissa on lines 2 through 5. If truncation is used in place of rounding, we simply substitute calls to RNTE with Trunc. Note, the multiplication protocol from [16] performs two sequential truncations of m by q and one bit(s) since truncation over a field can be performed in a constant number of rounds.

The remainder of the protocol is straightforward. On line 7, we calculate the exponent by summing the input exponents and adjusting to accommodate for the prior truncation by adding $[b]$. In the same step, we normalize the exponent by adding $q+1$ and multiply by the complement of the output's zero bit to account for non-zero values. The resultant sign and zero bits are computed by XORing and ORing, respectively, the corresponding components of $[\tilde{a}]$ and $[\tilde{b}]$.

5.4 Division

The logic behind floating-point division $[\tilde{c}] \leftarrow [\tilde{a}]/[\tilde{b}]$ shares much in common with our integer division protocol outlined in Section 4.2.6.2 (and, by association, the fixed-point division algorithm from [48]) with some key changes. The primary advantage is that for floating-point computation, the mantissas are already normalized to the range $\left[2^q, 2^{q+1}\right)$, thus eliminating the costly normalization and reciprocal approximation. We present our floating-point division algorithm FLDiv in Protocol 21.

The main complexity of floating-point division operation lies within the algorithm used to compute the mantissa quotient $[\tilde{a}.m]/[\tilde{b}.m]$, which we denote by MDiv (Protocol 22). We adjust the divisor by its zero bit $\tilde{b}.z$ to prevent division-by-zero. From here, there are several directions in which the mantissa division algorithm can proceed, each maintaining advantages depending on the end-use application. The protocol from [16] entirely foregoes computing the initial reciprocal w and instead directly sets it to $2^{-(q+1)}$ and subsequently scales all interim values in the iterative portion up by 2^{q+1}. The caveat is that more iterative steps ($\lceil \log(q+1) \rceil$) are required to compensate for a more imprecise starting reciprocal. [44] refines this approach by reintroducing the initial approximation of $w = 2.9142 - 2b$ and brings the number of iterations down to $\left\lceil \log \frac{q+1}{3.5} \right\rceil$. This version also explicitly accounts for extending the fractional part to $q + 1 + \lambda$ to ensure the error introduced through iteration is bounded by $< 2^{-(q+1)}$. SecFloat [140] adopts a different approach that combines a lookup table operation with two Newton-Raphson iterations. This leads to a tighter relative error bound of 2^{-q-2}, but the products and truncations in each iteration cannot be executed in parallel (the output of the first truncation is fed as an input into the second). This more than doubles the number of rounds per iteration at the cost of a lower overall

Protocol 21: $[\tilde{c}] \leftarrow \mathsf{FLDiv}([\tilde{a}], [\tilde{b}])$

1 $[m] \leftarrow \mathsf{MDiv}([\tilde{a}.m], [\tilde{b}.m] + [\tilde{b}.z], 4)$
2 $[b] \leftarrow \mathsf{LT}([m], 2^{q+1})$
3 $[m'] \leftarrow \mathsf{Trunc}([m], 1)$
4 $[\tilde{c}.m] \leftarrow [b] \cdot [m] + (1 - [b]) \cdot [m']$
5 $[\tilde{c}.e] \leftarrow (1 - [\tilde{a}.z]) \cdot ([\tilde{a}.e] + [\tilde{b}.e] - q - [b])$
6 $[\tilde{c}.s] \leftarrow [\tilde{a}.s] + [\tilde{b}.s] - 2[\tilde{a}.s] \cdot [\tilde{b}.s]$ // XOR
7 $[\tilde{c}.z] \leftarrow [\tilde{a}.z]$
8 $[\mathsf{Error}] \leftarrow [\tilde{b}.z]$
9 **return** $([\tilde{c}], [\mathsf{Error}])$

communication complexity.

We base our mantissa division solution MDiv on [44] due to its ability to be parallelized and the similarities it shares with our integer division algorithm (Protocol 15). We highlight the differences between MDiv and IntDiv below. First, the mantissa precision is extended by λ bits (lines 2 and 3) prior to computing the quotient. As stated, the values are already normalized, and thus we compute the initial approximation directly on line 4. The remaining steps of MDiv closely echo our integer division protocol, and we defer to Section 4.2.6.2 for more details.

The computed mantissa quotient m is normalized to be $q + 1$ or $q + 2$ bits, which we resolve by comparing m to 2^{q+1}, truncating by one bit, and selecting the appropriately normalized result (lines 2 to 4). The remainder of FLDiv involves adjusting the exponent by the number of bits truncated (line 5), computing the output's sign bit and zero bits (lines 6 and 7), and setting the error flag according to the zero bit of the divisor ($[\mathsf{Error}] = [\tilde{b}.z]$) (line 8).

5.5 Addition and Subtraction

Addition (and by extension, subtraction) of two floating-point inputs $[\tilde{c}] \leftarrow [\tilde{a}] + [\tilde{b}]$ is considerably more involved than multiplication and division. The difficulty stems from having to align the

Protocol 22: $[m] \leftarrow \mathsf{MDiv}([m_1], [m_2], \lambda)$

1 $\alpha = 2^{q+1+\lambda}, \theta = \left\lceil \log \frac{q+1}{3.5} \right\rceil$
2 $[m_1'] = 2^{\lambda}[m_1]$
3 $[m_2'] = 2^{\lambda}[m_2]$
4 $[w] = (2.9142) \cdot \alpha - 2[m_2]$
5 $[y] \leftarrow [m_1'] \cdot [w]$
6 $[x] \leftarrow [m_2'] \cdot [w]$
7 $[y] \leftarrow \mathsf{Trunc}([y], q + 1 + \lambda)$
8 $[x] \leftarrow \mathsf{Trunc}([x], q + 1 + \lambda)$
9 $[x] = \alpha - [x]$
10 **for** $i = 1, \ldots, \theta - 1$ **do**
11 $\quad [y] \leftarrow [y] \cdot (\alpha + [x])$
12 $\quad [x] \leftarrow [x] \cdot [x]$
13 $\quad [y] \leftarrow \mathsf{Trunc}([y], q + 1 + \lambda)$
14 $\quad [x] \leftarrow \mathsf{Trunc}([x], q + 1 + \lambda)$
15 $[y] \leftarrow [y] \cdot (\alpha 2^{\lambda} + [x])$
16 $[m] \leftarrow \mathsf{Trunc}([y], q + 1 + 2\lambda)$
17 **return** $([m])$

exponents to be the same value, which necessitates left-shifting the mantissa of the larger input. Once everything is aligned, we can proceed with the actual addition depending on the sign of the smaller input and normalize the result.

We provide our FLAdd construction in Protocol 23 and follows the logic of [16] and [140]. Subtraction is realized by flipping the sign of the second argument and performing the addition protocol.

The first step is to obliviously determine the larger and smaller of the two inputs on lines 1 to 6, which we denote by the interim floating-point values $\beta_{\max}$ and $\beta_{\min}$, respectively. We compute the difference between the exponents $\Delta = \beta_{\max}.e - \beta_{\min}.e$, and determine whether we are performing addition ($s = 0$) or subtraction ($s = 1$) (line 7). The difference Δ is guaranteed to be within the range $[0, q + 1]$ and dictates in tandem with s how we proceed throughout the remainder of the algorithm. If $\Delta > q + 1$ and we are performing addition, then the difference between the

two operands is too large, and we return $\beta.max$. If $\Delta > q+1$ and the operating is *subtraction*, this introduces two additional subcases that the protocol must account for, based on whether $\beta_{\max}.m > 2^q$ or $\beta_{\max}.m = 2^q$. For $\beta_{\max}.m > 2^q$, we decrement this value by 1 and set the output's exponent to $\beta_{\max}.e$ In the latter subcase $\beta_{\max}.m = 2^q$, the output's mantissa is exactly q bits long, all of which are set. This is accomplished by left shifting $\beta_{\max}.m$ by one, setting the least significant bit, and decrementing the exponent by one to obtain $\beta_{\max}.e - 1$. Both subcases are captured in a single statement on line 11 by computing $m_1 = 2(\beta_{\max}.m - s) + 1$. From the former case, m_1 will be $q+2$ bits long and need to be truncated back to $q+1$ bits. From the latter case, m_1 will be exactly $q+1$ bits long and the exponent will be correctly decremented on line 21.

Returning to the other parent case of $\Delta \leq q+1$, the final exponent will be $\beta_{\min}.e$ and the mantissa must be shifted by Δ bits. This is accomplished through our Pow2 functionality (Protocol 11) and we obtain the mantissa $m_2 = \beta_{\max}.m \cdot 2^{\Delta} + (1-2s) \cdot \beta_{\min}.m$. This value is $(q + \Delta \pm 1)$-bits long and at most $2q+3$ bits. Both cases ($\Delta > q+1$ and $\Delta \leq q+1$) are captured on line 14. The result is left-shifted by $q+1-\Delta$ bits to obtain a value m that is at most $2q+3$ bits long. We delay truncating the computed mantissa until after normalization, such that the full precision of the result is available for RNTE.

Normalizing m requires left-shifting the mantissa to remove any leading zero bits. This is accomplished by bit decomposing m and computing the prefix-OR of its bits to determine the position e_0 of the most significant nonzero bit, relative to the total bitlength $2q+3$ (lines 15 to 18). The normalization procedure concludes by left shifting the mantissa by e_0 bits to guarantee the mantissa is exactly $2q+3$ bits long and subsequently rounding the result to obtain a normalized $q+1$-bit value (line 20) The mantissa calculation concludes by accounting for when one (or both) of the inputs are zero (line 22).

Protocol 23: $[\tilde{c}] \leftarrow \mathsf{FLAdd}([\tilde{a}], [\tilde{b}])$

1 $([c_{\mathsf{LT}}], [c_{\mathsf{EQ}}]) \leftarrow \mathsf{LT\&EQ}([\tilde{a}.e], [\tilde{b}.e])$
2 $[m_{\mathsf{LT}}] \leftarrow \mathsf{LT}([\tilde{a}.m], [\tilde{b}.m])$
3 $[\beta_{\max}.e] \leftarrow [c_{\mathsf{LT}}] \cdot [\tilde{b}.e] + (1 - [c_{\mathsf{LT}}]) \cdot [\tilde{a}.e]$
4 $[\beta_{\min}.e] \leftarrow [c_{\mathsf{LT}}] \cdot [\tilde{a}.e] + (1 - [c_{\mathsf{LT}}]) \cdot [\tilde{b}.e]$
5 $[\beta_{\max}.m] \leftarrow (1-[c_{\mathsf{EQ}}]) \cdot ([c_{\mathsf{LT}}] \cdot [\tilde{b}.m] + (1-[c_{\mathsf{LT}}]) \cdot [\tilde{a}.m]) + [c_{\mathsf{EQ}}] \cdot ([m_{\mathsf{LT}}] \cdot [\tilde{b}.m] + (1-[m_{\mathsf{LT}}]) \cdot [\tilde{a}.m])$
6 $[\beta_{\min}.m] \leftarrow (1-[c_{\mathsf{EQ}}]) \cdot ([c_{\mathsf{LT}}] \cdot [\tilde{a}.m] + (1-[c_{\mathsf{LT}}]) \cdot [\tilde{b}.m]) + [c_{\mathsf{EQ}}] \cdot ([m_{\mathsf{LT}}] \cdot [\tilde{a}.m] + (1-[m_{\mathsf{LT}}]) \cdot [\tilde{b}.m])$
7 $[s] \leftarrow [\tilde{a}.s] + [\tilde{b}.s] - 2[\tilde{a}.s] \cdot [\tilde{b}.s]$ // XOR
8 $[d] \leftarrow \mathsf{LT}(q+1, [\beta_{\max}.e] - [\beta_{\min}.e])$
9 $[\Delta] = (1 - [d]) \cdot ([\beta_{\max}.e] - [\beta_{\min}.e])$
10 $[2^{\Delta}] \leftarrow \mathsf{Pow2}([\Delta], q+2)$
11 $[m_1] = 2([\beta_{\max}.m] - [s]) + 1$
12 $[m_2] \leftarrow [\beta_{\max}.m] \cdot [2^{\Delta}] + (1 - 2[s]) \cdot [\beta_{\min}.m]$
13 $[2^{q+1-\Delta}] \leftarrow \mathsf{Pow2}(q + 1 - [\Delta], q+2)$
14 $[m] \leftarrow ([d] \cdot [m_1] + (1 - [d]) \cdot [m_2]) \cdot [2^{q+1-\Delta}]$
15 $[m_0]_1, \ldots, [m_{2q+2}]_1 \leftarrow \mathsf{BitDec}([m], 2q+3)$
16 $[h_0]_1, \ldots, [h_{2q+2}]_1 \leftarrow \mathsf{PreOR}([m_{2q+2}]_1, \ldots, [1_0]_1)$
17 $[h_0], \ldots, [h_{2q+2}] \leftarrow \mathsf{B2A}([h_0]_1, \ldots, [h_{2q+2}]_1)$
18 $[e_0] = 2q + 3 - \sum_{i=0}^{2q+2} [h_i]$
19 $[2^{e_0}] = 1 + \sum_{i=0}^{2q+2} 2^i (1 - [h_i])$
20 $[m] \leftarrow \mathsf{RNTE}([2^{e_0}] \cdot [m], q+2)$
21 $[e] = [\beta_{\max}.e] - [e_0] + 1 - [d]$
22 $[\tilde{c}.m] \leftarrow (1 - [\tilde{a}.z]) \cdot (1 - [\tilde{b}.z]) \cdot [m] + [\tilde{a}.z] \cdot [\tilde{b}.m] + [\tilde{b}.z] \cdot [\tilde{a}.m]$
23 $[\tilde{c}.z] \leftarrow \mathsf{EQZ}([\tilde{c}.m])$
24 $[\tilde{c}.e] \leftarrow \left((1 - [\tilde{a}.z]) \cdot (1 - [\tilde{b}.z]) \cdot [e] + [\tilde{a}.z] \cdot [\tilde{b}.e] + [\tilde{b}.z] \cdot [\tilde{a}.e]\right) \cdot (1 - [\tilde{c}.z])$
25 $[s] \leftarrow (1 - [c_{\mathsf{EQ}}]) \cdot ([c_{\mathsf{LT}}] \cdot [\tilde{b}.s] + (1 - [c_{\mathsf{LT}}]) \cdot [\tilde{a}.s]) + [c_{\mathsf{LT}}] \cdot ([m_{\mathsf{LT}}] \cdot [\tilde{b}.s] + (1 - [m_{\mathsf{LT}}]) \cdot [\tilde{a}.s])$
26 $[\tilde{c}.s] \leftarrow ((1 - [\tilde{a}.z]) \cdot (1 - [\tilde{b}.z]) \cdot [s] + (1 - [\tilde{a}.z]) \cdot [\tilde{b}.z] \cdot [\tilde{a}.s] + [\tilde{a}.z] \cdot (1 - [\tilde{b}.z]) \cdot [\tilde{b}.s])$
27 **return** $[\tilde{c}]$

We adjust the exponent on line 21 to account for both the mantissa shifts/truncation and the special cases outlined above. For $\Delta > q+1$, the final exponent is set to $\beta_{\max}.e$ for $m > 2^q$ ($e_0 = 1$), or $\beta_{\max}.e - 1$ for $m = 2^q$ ($e_0 = 1$). The exponent is computed as $\beta_{\max}.e - e_0$. For $\Delta \leq q+1$, the mantissa was left shifted by a total of $\Delta + 1 - e_0$ bits, which is the amount exact $\beta_{\min}.e$ must be adjusted by. Therefore, the final exponent is $\beta_{\min}.e + \Delta + 1 - e_0 = \beta_{\max}.e - e_0 + 1 - d$.

The remainder of the algorithm reconciles all the aforementioned scenarios, specifically when

one (or both) of the inputs are zero. This includes determining zero bit $[\tilde{c}.z]$ by checking if the computed mantissa is zero (line 23) and adjusting the exponent to account for zero values (line 24). The sign bit $[\tilde{c}.s]$ is set based on the previously computed comparison bits (line 25), as well as the zero bits.

5.6 Comparisons

Protocol 24: $[b] \leftarrow \mathsf{FLLT}([\tilde{a}], [\tilde{b}])$

1 $([e_{\mathsf{LT}}], [e_{\mathsf{EQ}}]) \leftarrow \mathsf{LT\&EQ}([\tilde{a}.e], [\tilde{b}.e])$
2 $[m_0] \leftarrow (1 - 2[\tilde{a}.s]) \cdot [\tilde{a}.m]$
3 $[m_1] \leftarrow (1 - 2[\tilde{b}.s]) \cdot [\tilde{b}.m]$
4 $[m_{\mathsf{LT}}] \leftarrow \mathsf{LT}([m_0], [m_1])$
5 $[b^+] \leftarrow [e_{\mathsf{EQ}}] \cdot [m_{\mathsf{LT}}] + (1 - [e_{\mathsf{EQ}}]) \cdot [e_{\mathsf{LT}}]$
6 $[b^-] \leftarrow [e_{\mathsf{EQ}}] \cdot [m_{\mathsf{LT}}] + (1 - [e_{\mathsf{EQ}}]) \cdot (1 - [e_{\mathsf{LT}}])$
7 $[b] \leftarrow [\tilde{a}.z] \cdot (1 - [\tilde{b}.z]) \cdot (1 - [\tilde{b}.s]) + (1 - [\tilde{a}.z]) \cdot [\tilde{b}.z] \cdot [\tilde{a}.s]$
$+(1 - [\tilde{a}.z]) \cdot (1 - [\tilde{b}.z]) \cdot ([\tilde{a}.s] \cdot (1 - [\tilde{b}.s]) + (1 - [\tilde{a}.s]) \cdot (1 - [\tilde{b}.s]) \cdot [b^+] + [\tilde{a}.s] \cdot [\tilde{b}.s] \cdot [b^-])$
8 **return** $[b]$

Floating-point less-than comparison $[b] \leftarrow [\tilde{a}] \overset{?}{<} [\tilde{b}]$ functions as an extension of its integer counterpart with additional steps to accommodate all possible operand configurations. Our construction builds upon [16] and is presented in Protocol 24. The main logic relies upon determining the larger of the two exponents and using the mantissas (with the operands' signs applied) as a fallback if the exponents are equal (lines 2 and 4). If both operands are nonzero and have the same sign, the output is strictly determined by the exponent comparison. This translates into the quantities b^+ and b^- when both operands are positive and negative, respectively. Line 7 computes the resultant bit for all possible operand configurations, including when one or both inputs' zero bits are set and if the inputs have different signs. If both inputs are negative $\tilde{a}.s = \tilde{b}.s = 1$ and their exponents are different $\tilde{a}.e \neq \tilde{b}.s$, the argument with the smaller exponent is larger. The bulk of

the protocol can be executed in parallel, such as lines 1 through 4, as well as many of the products on line 7.

We summarize the performance of the floating-point operations presented in this chapter in Table 5.2.

Protocol	Rand.	Precomputation	Active	
			Rounds	Communication
FLMul	rB	$(3k+7)\mathsf{rB}$	$\log(3(k-1)(q+1))$ $\log(t+1)^2+13$	$nt\,(10k(t^2-t+1)+13k$ $+3(2m+1)+2\ell+10)$
	eB	$\mathsf{eBR}(k,q+1)+\mathsf{eBR}(k,q)$ $+2\cdot\mathsf{eB}(3,3)+\mathsf{eB}(k,k)+\mathsf{rB}$		
FLDiv	rB	$(1+(2\theta+3)k)\cdot\mathsf{rB}$	$\log((q+5)^\theta(q+9)(t+1)^{\theta+1})$ $+\log((k-1)(t+1))$ $+3(\theta+1)+5$	$nt\,((2\theta+2)k(t^2-t+1)$ $+(2\theta+8)k+16(\theta+1)$ $+2(2\theta+1)(2q+5)$ $+q-3)$
	eB	$\mathsf{eB}(k,k)+\mathsf{rB}+\mathsf{eBtr}(k,1)$ $+2\theta\cdot\mathsf{eBtr}(q+5)+\mathsf{eBtr}(q+9)$		
FLAdd	rB	$\binom{5k+2q+9}{+2\log(q+2)}\cdot\mathsf{rB}$	$\log((k-1)k^2(2q+3)^2)$ $+\log((q+2)(t+1)^6)$ $+2\log\log(q+2)+30$	$nt\,(k(\log(q+2)+2q+10)(t^2-t+1)$ $+\log(q+2)^2(\log\log(q+2)+k)$ $+\log(2q+3)(3(q+\frac{3}{2}))$ $+7q+45k+4)$
	eB	$4\cdot\mathsf{eB}(k,k)+2\cdot\mathsf{eB}(k,\log(q+2))$ $+\mathsf{eB}(k,2q+3)+\mathsf{eB}(3,3)$ $+\mathsf{eBR}(k,q+2)+3\cdot\mathsf{rB}$		
FLLT	rB	$2(k+1)\cdot\mathsf{rB}$	$\log(k(t+1))+4$	$nt\,(2k(t^2-t+1)$ $+20k-9)$
	eB	$2\cdot(\mathsf{eB}(k,k)+\mathsf{rB})$		

Table 5.2: Performance of floating-point operations. For convenience, $\mathsf{eB}(k,\ell)=\mathsf{edaBit}(k,\ell)$, $\mathsf{rB}=\mathsf{RandBit}()$, $\mathsf{eBtr}(k,m)=\mathsf{edaTrunc}(k,m)$, and $\mathsf{eBR}(k,m)=\mathsf{edaTruncRNTE}(k,m)$.

Part II

Information Disclosure Analysis for Secure Function Evaluation

Chapter 6

Related Work

In what follows, we review prior literature on information disclosure from function output in the context of computing on private data and related techniques that limit information disclosure.

6.1 Quantitative Information Flow

The field of *quantitative information flow* is closely related to our work. Denning [67] is credited as the first to quantify information flow as a measure of the interference between variables at two stages during a program's execution (typically denoted by "high-" and "low-security" variables, which equates to the target's inputs and output in our setting, respectively). Smith [152] formally established the foundations of quantifying the information leakage under the threat model that an attacker can recover a secret in one attempt (denoted by the notion of *vulnerability*). It has been shown by Massey [125] that the Shannon entropy cannot capture this information under the guessing assumption, and Smith recommends min-entropy in its place. Alvim et al. [19] generalized the min-entropy into the g-leakage to incorporate gain functions to model the *benefit* an adversary gains from making guesses about the secret. Subsequent works encompassed vari-

ations on the g-leakage [18]. Other works in differential privacy feature derivations of leakage bounds [54], leakage analysis in the case of an adaptive adversary [107], and knowledge-based approaches for measuring risk [122, 138].

The fundamental advantage of our Shannon-based approach is the ability to derive closed-form expressions for the information leakage of the average salary computation, while other metrics do not share this characteristic. For example, the chain rule of entropy (a simple, yet critical component of our analysis) is not satisfied if min-entropy is used [98, 151] in place of Shannon entropy. Our reductions would no longer hold, and we would be forced to resort to complete enumeration or approximation methods to compute the entropy. However, in Section 8.2 we provide supplementary analysis that demonstrates similarities between Shannon entropy and min-entropy-based analyses. We also remain open to evaluating other metrics in the future.

An additional distinction between our work and existing literature on (quantitative) information flow is that we do not consider possible leakage from intermediate aspects of a computation's execution. Whereas other works may examine a program's loops [122], side-channel vectors [107], or inter-dependent structures [17], we strictly investigate the relationship between the output and target's input, since the function itself is assumed to be evaluated using secure multi-party protocols.

6.2 Function Information Disclosure

Existing literature on information leakage from the output of a secure function evaluation is limited, relative to the rest of the field of secure computation. Secure multi-party protocols are designed to guarantee no information is disclosed throughout a computation, but do not ensure

input protection after the output is revealed. The work of Deshpande et al. [70, 69, 68] was pioneering in that respect and designed secure multi-party protocols for business applications that ensured that the function being evaluated is *non-invertible*, i.e., no participant can infer other participants' inputs from the output. A trivially invertible example is the average salary calculation between two individuals since either party can recover the other's input exactly. Deshpande et al. [70, 69] first addressed non-invertibility in the context of secure supply chain processes. The proposed protocols offered protection from inference of future inputs to a repeated calculation after a result is disclosed. A later work by Deshpande et al. [68] achieved non-invertibility for a framework designed for secure price masking for outsourcing manufacturing. The authors argued information leakage was minimal by analyzing mutual information between correlated normal random variables, but did not consider other distributions or entropy metrics.

Ah-Fat and Huth [12] provided the first in-depth analysis of information leakage from the outputs of secure multi-party computations. The authors formalized two metrics to measure expected information flow from the attacker's and target's perspectives, namely, the *attacker's weighted average entropy* (awae) and *target's weighted average entropy* (twae), respectively. Participants' inputs are modeled using probability distributions and were specified to be uniform, but this constraint can be relaxed. The inherent difficulty of this entropy-based approach is the requirement to enumerate every possible input combinations from all parties, which scales poorly as the input space and number of participants grow. We utilize their definitions for our analysis and demonstrate their utility to computation designers to determine potential disclosure about participants' inputs

This model was expanded in [13] to encompass the Rényi, min-, and g-entropy. The extension is presented in combination with a technique for distorting secure computation outputs to limit information disclosure from the output and achieve a balance between accuracy and privacy. This

was further developed in [14] with a fuzzing method based on randomized approximations. A closed-form expression for the min-entropy of a two- and three-party auction was derived in [15], alongside a conjecture for the case with an arbitrary number of parties.

Conceptually, the notion of *output privacy* is related to our work. The terminology was introduced in the field of data mining [40, 163, 108, 128, 131], with the goal of designing techniques to protect inputs from inference attacks on the output model. Information about the inputs that can be obtained from the output includes, but is not limited to, properties that can be uniquely attributed to a small number of input participants. Conventional approaches for minimizing disclosure involve applying transformations on the result via monotonic functions [40] or even proactive learning [163]. These techniques have little to no impact on the result of the computation. This direction differs from our work since the type of disclosure they aim to rectify is not quantified.

6.3 Information Disclosure from Machine Learning Models

A topic that received significant attention in recent years is the training of machine learning (ML) models on private data. In that context, the goal is to limit information disclosure about the records on which a model has been trained when the model is released or otherwise is used in privacy-preserving inference without disclosing the model itself. In that respect, it has been shown that an ML model can disclose information about individual records on which it has been trained [150, 153, 157, 53, 94, 154, 133, 91]. A model which consists of various weights and biases may be prone to memorizing information about individual records. One possible attack vector is through *membership inference attacks* (MIA), which refers to the following scenario: given a machine learning model in a sensitive context (e.g., corresponding to individuals with a certain med-

ical condition) and single data point, an attacker seeks to determine whether the data point was part of the training dataset. An adversary may have varying levels of access to the model, where the strictest configuration limits the adversary to black-box queries of the model and observe the outputs on data of their choice.

Shokri et al. [150] were among the first to investigate this class of attacks and offered several mitigation strategies for MIAs. Specifically, the entropy of a neural network's output prediction vector was improved by modifying (or adding) a softmax layer and increasing the normalization temperature. Song and Mittal [153] formalized a suite of attacks based on a modified version of the prediction entropy to benchmark a target model's privacy risks. The authors found that several state-of-the-art defense techniques could not effectively mitigate against their benchmark attacks. Hu et al. [94] showed that the entropy of the target dataset (calculated by averaging the mean entropy of all the features) influences the success of MIAs. The mutual information between model parameters and training records quantified the information extracted by the target model from the set of training records, i.e., captured from the features of the training data. A higher mutual information was shown to indicate a greater likelihood of exposure to MIAs. Nasr et al. [133] used the normalized entropy to measure the effectiveness of their MIA prevention mechanism.

6.4 Differential Privacy

The concept of differential privacy (DP) [73, 75, 76, 129] relies upon the principle of restricting the information learned about a single individual within a sensitive dataset. Computation-specific mechanisms are designed to operate on two datasets with the only difference that a single participant is either absent or present, such that the outputs are statistically indistinguishable up to a

specified security parameter ε. Our analysis of information leakage after a secure function evaluation is tangential to DP, where the loss is related to the security parameters. In the context of this work, the number of participants, frequency of participation, and statistical parameters dictate the amount of information revealed about the target. Furthermore, DP can be combined with secure multi-party computation when the function reveals too much information about private inputs.

Barthe and Köpf [24] investigated the relationship between differential privacy and [152]'s notion of information-theoretic leakage. The authors formulated a common model for leakage and differential privacy and proved properties regarding the security guarantees (in terms of the security parameter), as well as compositionality properties for combining two secure systems. If the input domain is binary, the authors showed how to completely characterize the leakage in the context of differential privacy. Larger input domains were limited to strict upper bounds in terms of the security parameter and domain size.

We discuss differential privacy in the context of the average salary in more detail in Section 8.1.4.

Chapter 7

Background

Our threat model considers protecting the privacy of input owners who contribute private data into a joint computation. We study information disclosure about the private inputs from the output of function evaluation, and it is assumed that the evaluation itself does not disclose any information about the inputs (which can be achieved using a variety of known techniques). In what follows, we refer to the parties contributing their private inputs into the computation as "computation participants," as the mechanism for function evaluation itself is orthogonal to this work.

The adversary is interested in learning information about the private inputs of one or more participants from the function output, and we denote the targeted individuals (one participant or more participants treated as a group) as the target. The adversary can participate in the computation as well and control the inputs of one or more participants. All participants controlled by the adversary are referred to as the "adversary." The remaining input owners are independent of the target and adversary and are called "spectators." Their presence plays an important role of protecting the target's inputs.

7.1 Information Theory

Our analysis relies upon the notion of *entropy* to quantify information disclosure. Entropy is the de facto choice for measuring the uncertainty of a random variable. It serves as the foundation of the field as a whole and offers many desirable properties. The Shannon entropy [149], denoted by $H(X)$, measures the information of a discrete random variable X with mass function $\Pr(X = x)$ supported by $\mathcal{X}$ and is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} \Pr(X = x) \cdot \log \Pr(X = x),$$

where all logarithms are to the base 2. If we are interfacing with continuous distributions, we shift to the differential entropy $h(X)$ with density function $f(x)$ over the support set $\mathcal{X}$, defined as

$$h(X) = -\int_{\mathcal{X}} f(x) \log f(x) dx.$$

Note that while the Shannon and differential entropies are formulated in similar manners for discrete and continuous distributions, there is *no direct relationship* between the two, since the differential entropy of a discrete random variable would be infinite (see [57, Chapter 8.3]).

7.2 Formal Setting

Our information-theoretic analysis begins with establishing our formal setting, the first of which involves how we differentiate inputs into a computation controlled by adversaries, targets, and the remaining participants. Adopting [12]'s notation, let P denote the set of all participants in a

computation and n denote the number of participants, i.e., $|P| = n$. All participants P are divided into three groups: parties controlled by an attacker $A \subset P$, a group of parties being targeted $T \subseteq P \setminus A$, and the remaining participants (spectators) $S = P \setminus (A \cup T)$. These groups are permitted to be empty with the exception of the target group (e.g., we may encounter configurations where $S = \{\emptyset\}$). We use random variables to model values used in computations. In that respect, let random variable X_{P_i} with support $\mathcal{P}_i$ model the input of a single participant P_i and x_{P_i} denotes a value that X_{P_i} takes. In addition, let $\vec{X}_P = (X_{P_1}, \ldots, X_{P_k})$ denote a multidimensional random variable and $\vec{x}_P$ be a vector of the individual values of size k. We also let $X_P = \sum_i X_{P_i}$ define a new random variable representing the sum of the participants' random variables. The same notation applies to the sets A, T, and S.

Lastly, consider an arbitrary n-ary function $f : \mathcal{P}_1 \times \mathcal{P}_2 \times \cdots \times \mathcal{P}_n \to \mathcal{O}$, where $\mathcal{P}_i$ is the domain of the ith input corresponding to the input of party P_i and $\mathcal{O}$ is the codomain of f (set possible outputs). We model the output by the random variable O (supported by $\mathcal{O}$) and denote the output value by o. Our analysis is based upon the assumption that all participants' inputs are independent and identically distributed, but we relax this assumption in our treatment of the average salary in Section 8.3.

As stated in Section 6.2, Ah-Fat and Huth [12] provided multiple information-theoretic measures to quantify information disclosure after a function evaluation, which we use here:

Definition 2: jwae [12]

The *joint weighted average entropy* (jwae) of a target T attacked by parties A is defined over all $\vec{x}_A \in \mathcal{A}$

and $\vec{x}_T \in \mathcal{T}$ as

$$\text{jwae}(\vec{x}_A, \vec{x}_T) = \sum_{o \in \mathcal{O}} \Pr(O = o \mid \vec{X}_A = \vec{x}_A, \vec{X}_T = \vec{x}_T) \cdot H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o).$$

This metric measures the information an attacker would learn (on average) about the target when the input vectors are $\vec{x}_A$ and $\vec{x}_T$. One can subsequently define the average of the jwae over all possible $\vec{x}_T$ or $\vec{x}_A$ vectors weighted by their respective prior probabilities.

Definition 3: twae [12]

The *target's weighted average entropy* (twae) of a target T attacked by parties A is defined for all $\vec{x}_T \in \mathcal{T}$ as

$$\text{twae}(\vec{x}_T) = \sum_{\vec{x}_A \in \mathcal{A}} \Pr(\vec{X}_A = \vec{x}_A) \cdot \text{jwae}(\vec{x}_A, \vec{x}_T).$$

The twae informs a target how much information an attacker can learn about its input when the input is $\vec{x}_T$.

Definition 4: awae [12]

The *attacker's weighted average entropy* (awae) of a target T attacked by parties A is defined for all $\vec{x}_A \in \mathcal{A}$ as

$$\text{awae}(\vec{x}_A) = \sum_{\vec{x}_T \in \mathcal{T}} \Pr(\vec{X}_T = \vec{x}_T) \cdot \text{jwae}(\vec{x}_A, \vec{x}_T).$$

The awae informs an attacker about how much information it can learn about the target's input when the attacker's input vector is $\vec{x}_A$. The attacker can consequently compute the awae on all values in $\mathcal{A}$ to determine which input maximizes the information learned about the target's input

(and thus what should be entered into the computation). Using the Definition 2, it follows that:

$$\begin{aligned}\text{awae}(\vec{x}_A) &= \sum_{\vec{x}_T \in \mathcal{T}} \Pr(\vec{X}_T = \vec{x}_T) \sum_{o \in \mathcal{O}} \Big(\Pr(O = o | \vec{X}_A = \vec{x}_A, \vec{X}_T = \vec{x}_T) \\ &\qquad \cdot H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o) \Big) \\ &= \sum_{\vec{x}_T \in \mathcal{T}} \sum_{o \in \mathcal{O}} \Pr(O = o, \vec{X}_T = \vec{x}_T \mid \vec{X}_A = \vec{x}_A) \cdot H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o).\end{aligned}$$

Since $\vec{X}_T$ is independent of $\vec{X}_A$, we derive that awae equals to conditional entropy:

$$\text{awae}(\vec{x}_A) = \sum_{o \in \mathcal{O}} \Pr(O = o \mid \vec{X}_A = \vec{x}_A) \cdot H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o) = H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O)$$

where the last equality is due to the definition of conditional entropy.

We consider several distributions of practical interest for a variety of applications. For our analysis of the average salary function, we prioritize distribution where the sum of independent individual random variables is well studied and their mass or density functions have closed-forms expressions or can be reasonably approximated. This includes the following distributions:

- *Discrete uniform* $\mathcal{U}(a, b)$, where a and b are integers corresponding to the minimum and maximum of the range of the support set $\{a, a+1, \ldots, b-1, b\}$.
- *Poisson* $\text{Pois}(\lambda)$, where $\lambda \in \mathbb{R}_{>0}$ is the shape parameter that indicates the expected (average) rate of an event occurring over a given interval.
- *Normal (Gaussian)* $\mathcal{N}(\mu, \sigma^2)$, where $\mu \in \mathbb{R}$ and $\sigma^2 \in \mathbb{R}_{>0}$ correspond to the mean and squared standard deviation, respectively.
- *Log-normal* $\log \mathcal{N}(\mu, \sigma^2)$ with parameters $\mu \in \mathbb{R}$ and $\sigma^2 \in \mathbb{R}_{>0}$, which correspond to the

mean and squared standard deviation of the random variable's natural logarithm.

The notation $X \sim \mathsf{Dist}$ indicates that random variable X has distribution Dist.

Chapter 8

Average Salary: Single Evaluation

In this chapter, we study the information disclosure of the computation of the average:

$$o = f(\vec{x}_A, \vec{x}_T, \vec{x}_S) = \frac{1}{n}\left(\sum_i x_{T_i} + \sum_j x_{A_j} + \sum_k x_{S_k}\right),$$

where o denotes the output of the function. Using our notation established in Section 7.2, the output o by the random variable O defined over the domain $\mathcal{O}$, namely

$$O = \frac{1}{n}\left(\sum_i X_{T_i} + \sum_j X_{A_j} + \sum_k X_{S_k}\right).$$

For the average, the $1/n$ factor can be ignored in the final expression since the number of participants is typically known by all parties and can trivially be removed from the output. We omit it throughout the remainder of our analysis of the average function.

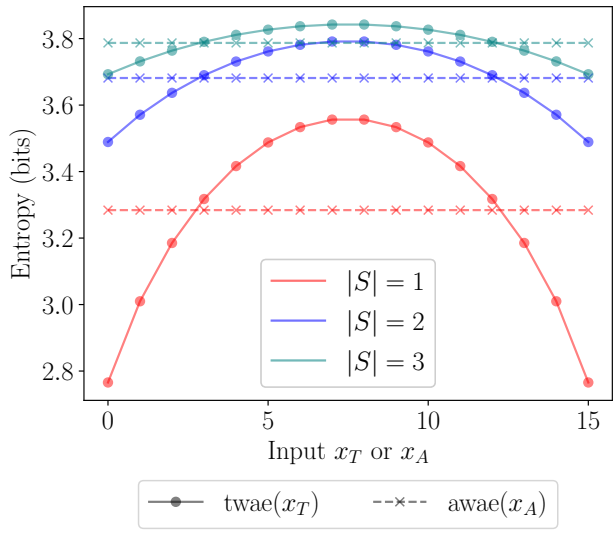


Figure 8.1: The twae($\vec{x}_T$) and awae($\vec{x}_A$) using inputs over $\mathcal{U}(0, 15)$ with a different number of spectators $|S|$.

Recall that the computation is modeled by

$$O = f(\vec{X}_A, \vec{X}_T, \vec{X}_S) = \sum_i X_{T_i} + \sum_j X_{A_j} + \sum_k X_{S_k}, \tag{8.1}$$

and we let $s = |S|$ denote the number of spectators.

As a first step, we plot the values of awae and twae for our function of interest. Figure 8.1 illustrates these values with a single adversarial participant, a single target and a varying number of spectators (1–3). All inputs follow the uniform distribution $\mathcal{U}(0, 15)$. Calculating the twae and awae values using Definitions 3 and 4 requires enumerating all input and output combinations. This quickly becomes computationally inefficient as the input space grows.

Each participant, acting as a target, can utilize the twae prior to the computation to determine how much information an attacker can learn (on average) from the output for a specific input that the participant enters into the computation. As the figure illustrates, the target's remaining

average entropy is maximized when the input is in the middle of the range, indicating that those values have better protection than inputs near the extrema. As the number of spectators increases, the curves shift upwards, i.e., the uncertainty about the target's input increases and the gap in the uncertainty between different input values reduces.

The awae, on the other hand, gives an adversary the ability to determine which input to enter into the computation that leads to the maximum information disclosure about a target's input (without knowing what input the target used). As displayed in the figure, the adversarial knowledge does not change by varying its inputs into the computation. This is consistent with our intuition that, given the output, the adversary can remove their contribution to the computation and possess information about the sum of the inputs of the remaining parties. We formalize this as the following result:

Claim 1

awae($\vec{x_A}$) is independent of attacker's input vector $\vec{x_A}$.

Proof. According to the chain rule of entropy which states that $H(X,Y) = H(X \mid Y) + H(Y)$ [57, Chapter 2.5], we have that:

$$\begin{aligned} H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O) &= H(\vec{X}_T, O \mid \vec{X}_A = \vec{x}_A) - H(O \mid \vec{X}_A = \vec{x}_A) \\ &= H(\vec{X}_T \mid \vec{X}_A = \vec{x}_A) + H(O \mid \vec{X}_T, \vec{X}_A = \vec{x}_A) - H(O \mid \vec{X}_A = \vec{x}_A) \\ &= H(\vec{X}_T) + H\left(\sum_i X_{S_i}\right) - H\left(\sum_i X_{T_i} + \sum_j X_{S_j}\right), \end{aligned} \tag{8.2}$$

which is independent of $\vec{x_A}$. □

Using our notation from Chapter 7, the above expression for awae($\vec{x_A}$) simplifies to

$$H(\vec{X}_T) + H(X_S) - H(X_T + X_S) = H(\vec{X}_T \mid X_T + X_S). \tag{8.3}$$

The next step is to determine which measure (awae or twae) we should use in our analysis of the average salary computation. Ah-Fat and Huth [12] argued that the awae served as a more precise metric for measuring information leakage of a secure function evaluation than twae for their choice of function and used awae in their subsequent work [13]. Our perspective also aligns with that conclusion. In particular, while the twae informs the target of the amount of information leakage for the input they possess, the target may not be technically savvy enough to be able to apply the metric and make an informed decision regarding computation participation (plus, the choice to participate or not participate can leak information about their input). Perhaps more importantly, a function needs to be analyzed by the computation designers in advance and without access to the inputs of future computation participants to determine a safe setup for the participants. Thus, the available mechanism for this purpose is the attacker's perspective or awae, and we focus on this metric in the rest of this work.

Based on the above, in what follows we use $H(\vec{X}_T \mid X_T + X_S)$ to measure the leakage, and the simplified function is

$$f(\vec{X}_T, \vec{X}_S) = \sum_i X_{T_i} + \sum_j X_{S_j} = X_T + X_S.$$

This refines the parameters we can vary in our analysis to (1) the number of participants in the target and spectators groups and (2) the types of distributions and statistical parameters of the

inputs. Furthermore, the computational difficulty associated with directly computing the awae is absent when using $H(\vec{X}_T \mid X_T + X_S)$. Instead, the computation simplifies to calculating the entropy of sums of random variables.

We examine the behavior of the conditional entropy for several characteristic probability distributions next.

8.1 Single Execution Analysis

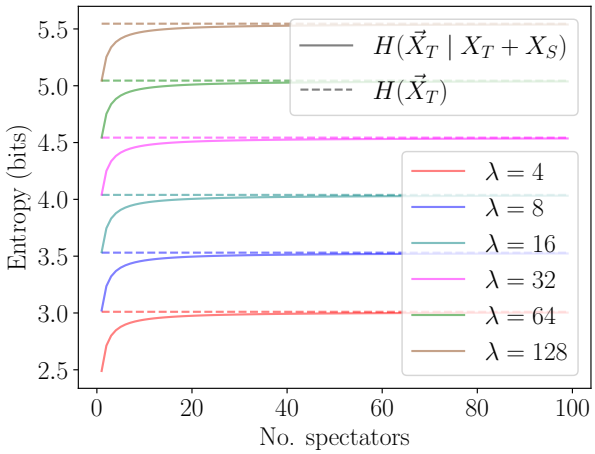
8.1.1 Discrete Distributions

We start with discrete input modeled using the uniform and Poisson distributions. The sum of s identical independent Poisson random variables $X_i \sim \text{Pois}(\lambda)$ is equivalent to a single Poisson random variables $X = \sum_i X_i \sim \text{Pois}(s\lambda)$ with the mass function

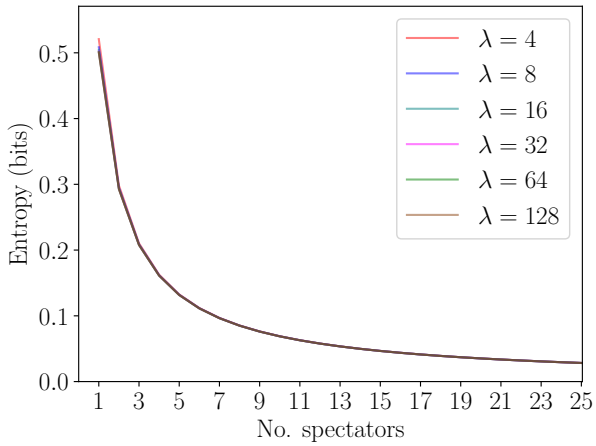
$$\Pr(X = x) = \frac{(s\lambda)^x e^{-s\lambda}}{x!}.$$

Note that the Poisson distribution is defined over all non-negative integers, hence the distribution has infinite support. We choose to halt the calculation of $H(X)$ when $\Pr(X = x) < 10^{-7}$ as the contribution of events beyond this point to the entropy is infinitesimal.

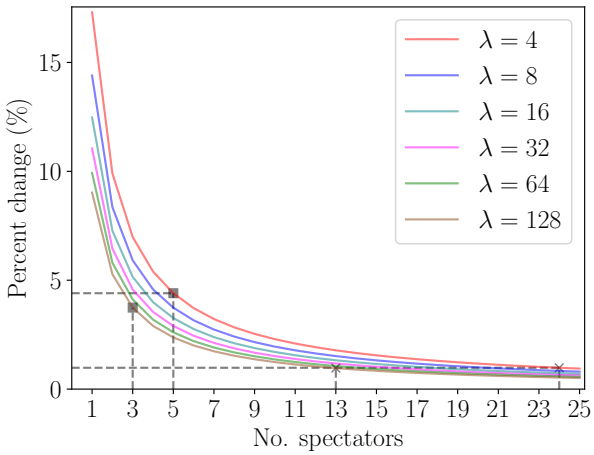
Conversely, the sum of s identical independent uniform random variables $X_i \sim \mathcal{U}(0, N-1)$ is not immediately obvious. Caiado and Rathie [41] derived several equivalent expressions for the mass function of the sum of s uniform random variables, one of which we use in our analysis and



(a) Target's entropy before $H(\vec{X}_T)$ and after $H(\vec{X}_T \mid X_T + X_S)$ the execution.



(b) Target's absolute entropy loss $H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)$.



(c) Target's relative entropy loss $\frac{H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)}{H(\vec{X}_T)}$.

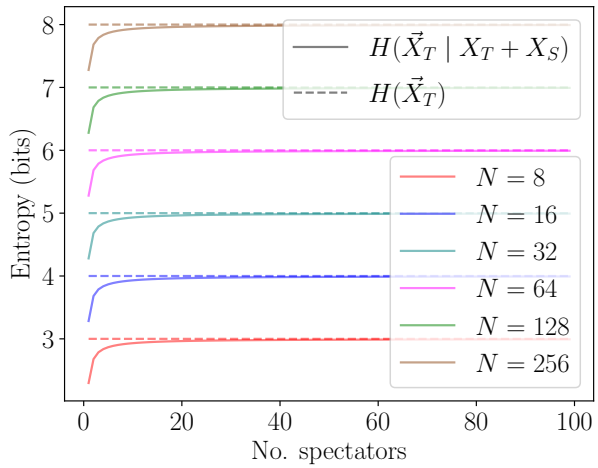
Figure 8.2: Analysis of target's entropy loss using the Poisson distribution with Pois(λ), and varying λ with $|T| = 1$.

is defined as:

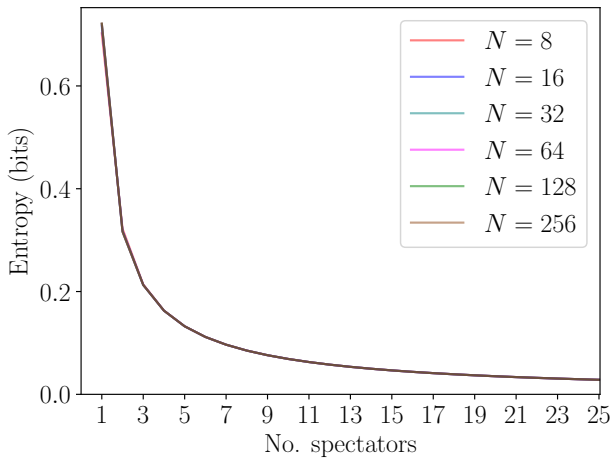
$$\Pr(X = x) = \frac{s}{N^s} \left(\sum_{p=0}^{\lfloor x/N \rfloor} \frac{\Gamma(s + x + pN)(-1)^p}{\Gamma(p+1)\Gamma(s-p+1)\Gamma(x - pN + 1)} \right),$$

where $\Gamma(s) = (s-1)!$ is the Gamma function. The domain of X is $\{0, \ldots, s(N-1)\}$.

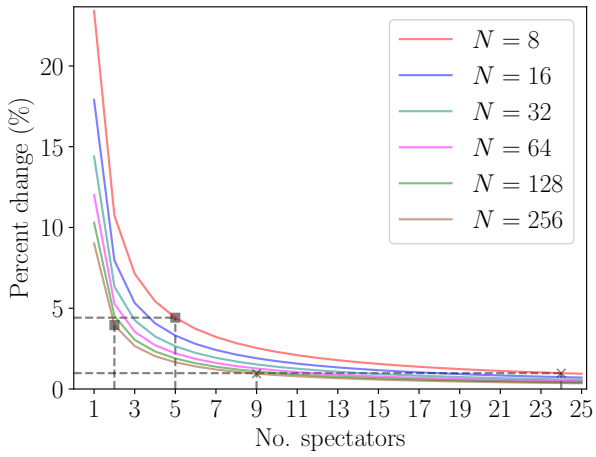
Our analysis of awae for these two distribution is given in Figures 8.2 and 8.3, respectively. We



(a) Target's entropy before $H(\vec{X}_T)$ and after $H(\vec{X}_T \mid X_T + X_S)$ the execution.



(b) Target's absolute entropy loss $H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)$.



(c) Target's relative entropy loss $\frac{H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)}{H(\vec{X}_T)}$.

Figure 8.3: Analysis of target's entropy loss using the uniform distribution with $\mathcal{U}(0, N-1)$, and varying N with $|T| = 1$.

compute and display

- the original entropy of target's inputs prior to the computation $H(\vec{X}_T)$ (subfigure a)
- the awae or target's remaining entropy after the computation $H(\vec{X}_T \mid X_T + X_S)$ (subfigure a)
- their difference of the two that represents the absolute entropy loss $H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)$ (subfigure b) and

- the entropy loss relative to the original entropy prior to the execution $(H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S))/H(\vec{X}_T)$ (subfigure c)

with a single target ($|T| = 1$), a varying number of spectators, and varying distribution parameters. Relative entropy loss is included to demonstrate to potential input contributors, who are likely non-experts, that information disclosure is small. That is, disclosure of, e.g., 5% of input's information is easier to explain to non-experts than 0.1 bits of entropy. The absolute loss is equivalent to the mutual information between the target input and the output:

$$I(\vec{X}_T; O) = H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S).$$

Figure 8.2 presents this information for the Poisson distribution with $\lambda \in \{4, 8, \ldots, 128\}$. In Figure 8.2a, entropy after the execution converges toward the corresponding entropy prior to the execution for all values of λ as the number of spectators increases. Increasing λ by a factor of two repeatedly yields an upward shift of these two curves by a constant amount while preserving their respective shapes. The increase is expected as a result of the inputs having more entropy as λ increases, but the shape of the remaining entropy is notable, as λ does not appear to impact the entropy loss. This is further confirmed when displaying the absolute entropy loss in Figure 8.2b: The resultant curves overlap each other, regardless of λ.

The relative entropy loss in Figure 8.2c, calculated as a percentage of the target's initial entropy, demonstrates how many spectators the computation needs to include to lower the entropy loss to the desired level. The larger the original entropy is (larger λ), the fewer spectators will be needed to stay within the desired percentage. For example, 5 spectators are needed with $\lambda = 4$ to limit relative loss to 5% (marked by ■) and 24 spectators are needed to cap the loss at 1% (marked by

$\times$). When $\lambda = 128$, the number of spectators reduces to 3 and 13 to maintain loss tolerances of 5% and 1%, respectively.

The same trends hold for the uniform distribution in Figure 8.3, where we use $N \in \{8, 16, \ldots, 256\}$, but the values themselves slightly differ. For example, the absolute entropy loss in Figure 8.3b is slightly larger than the loss in Figure 8.2b when the number of spectators is small. When $N = 8$ with 3 bits of original entropy, 5 and 24 spectators are needed to achieve at most 5% and 1% relative loss, respectively. This is the same as what was observed for Poisson distribution with 3-bit inputs ($\lambda = 4$).

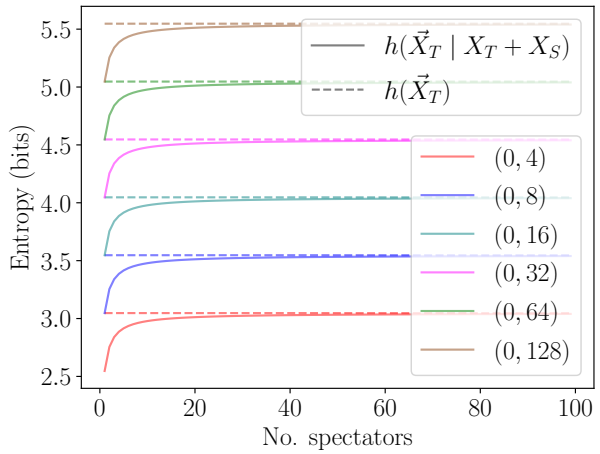
8.1.2 Continuous Distributions

For continuous distributions, we shift to differential entropy and analyze normal and log-normal distributions, the latter of which is typically used to model salaries. Recalling the fact that there is no direct relationship between differential and Shannon entropy (see [57, Chapter 8.3]), we nonetheless demonstrate that they exhibit very similar behavior for the average computation.

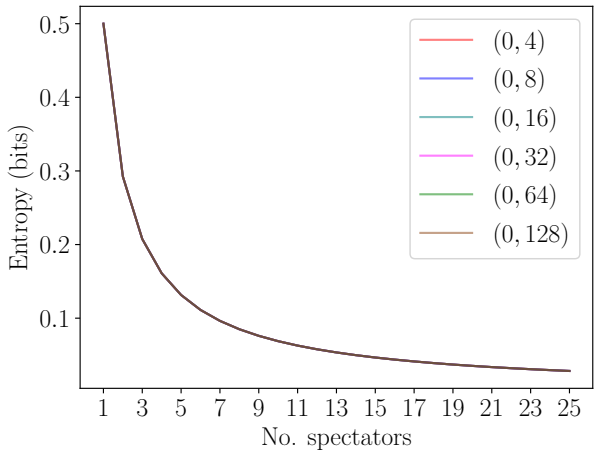
The differential entropy of a normal random variable $X_i \sim \mathcal{N}(\mu, \sigma^2)$ is $h(X_i) = \frac{1}{2}\log\left(2\pi e\sigma^2\right)$ [57, Chapter 8.1]. The sum of s identical normal random variables is also normal, namely $X \sim \mathcal{N}(s\mu, s\sigma^2)$. This enables us to directly apply the differential entropy definition to the sum.

The log-normal distribution is a well-established means of modeling salary data for 99% of the population [55], with the top 1% modeled by the Pareto distribution [155]. The differential entropy of a log-normal random variable $X_i \sim \log\mathcal{N}(\mu, \sigma^2)$ is $h(X_i) = \log\left(e^{\mu+\frac{1}{2}}\sqrt{2\pi\sigma^2}\right)$. However, the sum of s log-normal random variables has no closed form and is an active area of research [23, 56, 82, 25, 26, 145, 147, 164]. We adopt the Fenton-Wilkinson (FW) approximation [1] [82, 56]

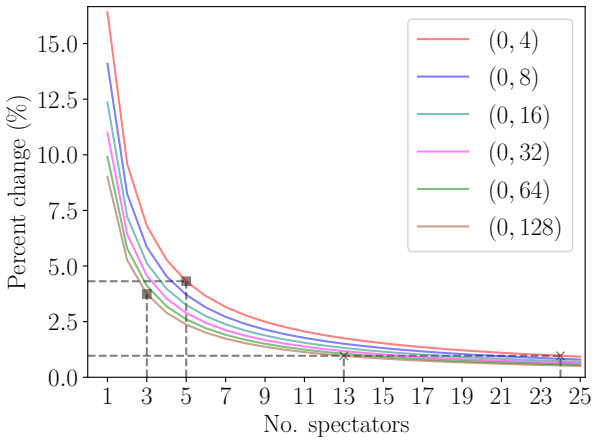
[1] Other approximations for the sum of log-normal random variables are difficult to translate into an expression for the differential entropy and hence we choose the FW approximation. Its disadvantage is that the FW approximation



(a) Target's entropy before $h(\vec{X}_T)$ and after $h(\vec{X}_T \mid X_T + X_S)$ the execution.



(b) Target's absolute entropy loss $h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)$.



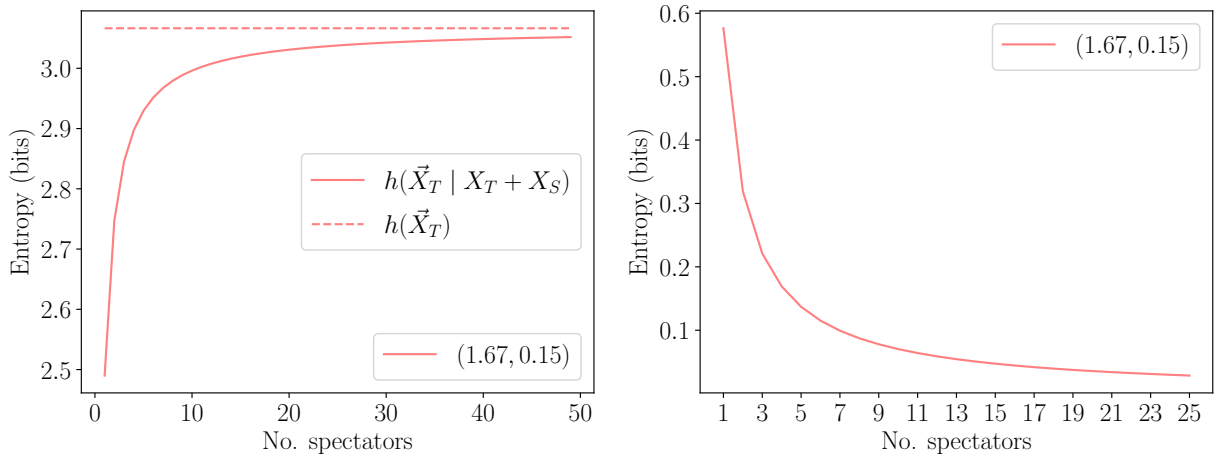
(c) Target's relative entropy loss $\frac{h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)}{h(\vec{X}_T)}$.

Figure 8.4: Analysis of target's entropy loss using the normal distribution with $\mathcal{N}(0, \sigma^2)$, and varying σ^2 with $|T| = 1$.

that specifies a sum of s identical independent log-normal random variables $X_i \sim \log \mathcal{N}(\mu, \sigma^2)$ as another log-normal random variable $X \sim \log \mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ with parameters

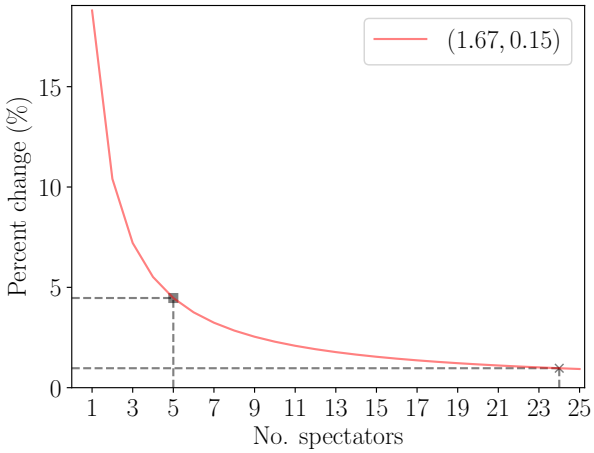
$$\hat{\sigma}^2 = \ln\left(\frac{\exp(\sigma^2) - 1}{s} + 1\right), \; \hat{\mu} = \ln(s \cdot \exp(\mu)) + \frac{1}{2}\left(\sigma^2 - \hat{\sigma}^2\right).$$

deteriorates for $\sigma^2 > 4$ and small values of x in the density function [25, 164]. Fortunately, our σ^2 is sufficiently small, allowing us to use the FW approximation free of consequence.



(a) Target's entropy before $h(\vec{X}_T)$ and after $h(\vec{X}_T \mid X_T + X_S)$ the execution.

(b) Target's absolute entropy loss $h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)$.



(c) Target's relative entropy loss $\frac{h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)}{h(\vec{X}_T)}$.

Figure 8.5: Analysis of target's entropy loss using the log-normal distribution with $\log \mathcal{N}(1.6702, 0.145542)$ and $|T| = 1$.

This enables us to compute differential entropy using a closed-form expression. Unlike prior distributions, we use a single set of μ and σ^2 parameters calculated from real salary data in [42]; namely, $\mu = 1.6702$ and $\sigma^2 = 0.145542$.

Figures 8.4 and 8.5 present experimental evaluation of entropy loss with a single target and a varying number of spectators for normal and log-normal distributions, respectively. As before, we report the target's entropy before and after the execution, the difference of the two as the absolute

entropy loss, and the entropy loss relative to the entropy before the execution.

In Figure 8.4 (normal), we set the mean $\mu = 0$ for all experiments (since differential entropy does not depend on μ) and vary σ^2 from 4 to 128. The results are consistent with the discrete counterparts in terms of the trends, curve shapes, and specific values. The absolute loss in Figure 8.4b is once again constant for any σ^2 and the relative loss is dictated by the amount of input's entropy in Figure 8.4c. When $\sigma^2 = 4$ and inputs have 3 bits of entropy, the number of spectators required to maintain at most 5% and 1% entropy loss (5 and 24 spectators, respectively) is the same as for Poisson and uniform distributions with 3-bit inputs ($\lambda = 4$ and $N = 8$, respectively). With 5.5-bit inputs ($\sigma^2 = 128$), 3 and 13 spectators are needed to achieve at most 5% and 1% loss, respectively, which is the same for the Poisson distribution with 5.5-bit inputs ($\lambda = 128$).

The results in Figure 8.5 (log-normal with real salary parameters) are consistent with both the discrete and continuous distributions. Surprisingly, we observe the same 5 and 24 spectators achieve at most 5% and 1% relative loss, as observed with all other distributions (with input original entropy being slightly over 3 bits).

Before concluding our discussion of continuous distributions, we are able to show one more result. We experimentally demonstrated that the amount of absolute entropy loss is parameter-independent for several distributions, but we can formally prove this for normally distributed inputs:

Claim 2

If the inputs are modeled by independent identically distributed normal random variables, the absolute entropy loss $h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)$ depends only on the number of target $|T| = t$ and spectator $|S| = s$ inputs and is $\frac{1}{2}\log\left(\frac{t}{s} + 1\right)$.

Proof. Let $|T| = t$ and $|S| = s$, such that $X_T \sim \mathcal{N}(0, t\sigma^2)$ and $X_S \sim \mathcal{N}(0, s\sigma^2)$. The absolute entropy loss is therefore

$$\begin{aligned} h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S) &= h(\vec{X}_T) - \Big(h(\vec{X}_T) + h\left(X_S\right) - h\left(X_T + X_S\right)\Big) \\ &= h\left(X_T + X_S\right) - h\left(X_S\right) \\ &= \frac{1}{2}\log 2\pi e(t+s)\sigma^2 - \frac{1}{2}\log 2\pi es\sigma^2 \\ &= \frac{1}{2}\log\left(\frac{t}{s}+1\right) = \Theta\left(\log\left(\frac{t}{s}+1\right)\right), \end{aligned}$$

which depends only on s and t. □

8.1.3 Discrete versus Continuous Distributions

We next compare the information loss across all four (discrete and continuous) distributions. We choose parameters to maintain the initial entropy of an input, $H(X_i)$ or $h(X_i)$, to be approximately 3 bits, as to reasonably correspond to the log-normal distribution. This leads to Pois(4), $\mathcal{U}\,(0, 7)$, and $\mathcal{N}(0, 4)$. We plot this information for a single target and a varying number of spectators in Figure 8.6.

In the figure, all distributions converge with ≥ 4 spectators and are very close even with 3 spectators. This convergence on large values is expected as a consequence of the central limit theorem. From the four distributions, the closest are the Poisson results with $\lambda = 4$ (discrete) and the normal distribution $\mathcal{N}(0, 4)$ (continuous). Unlike the normal, log-normal, and single-variate uniform distributions, an exact expression of the entropy of a Poisson distribution has not been derived. Instead, when computing the necessary values in Section 8.1.1, we directly applied the definition of Shannon entropy. To draw a parallel between discrete and continuous

distributions, and specifically show a similarity between Poisson and normal distributions, we turn to an approximation of Poisson distribution's entropy computation.

It was conjectured that for sufficiently large λ (e.g., $\lambda > 10$), the Poisson distribution's Shannon entropy can be approximated by $H(X_i) = \frac{1}{2}\log(2\pi e\lambda)$, which resembles $h(X_i) = \frac{1}{2}\log(2\pi e\sigma^2)$ used for normal distributions. Evans and Boersma [81] proposed a tighter bound (further formalized by Cheraghchi in [51]), to be

$$H(X_i) = \frac{1}{2}\log(2\pi e\lambda) - \frac{1}{12\lambda} - \frac{1}{24\lambda^2} - \frac{19}{360\lambda^3} + O(\lambda^4)$$

and remains close to that of normal distribution with $\sigma^2 = \lambda$.

One implication of this result for us is that Claim 2, which we demonstrated for normal distributions, would apply to the approximation of Poisson distributions as well. As a result, we obtain independence of the (absolute) entropy loss of distribution parameters for both discrete and continuous distributions and almost identical behavior across the distributions as a function of the number of spectators.

8.1.4 Comparison to Differential Privacy

The purpose of this work is to measure information disclosure from function output, which is the first necessary step to determine whether the function is suitable for evaluation on private data. Once it is determined that it is not, the second question to answer is how the function or the setup is to be modified to reduce information disclosure to a controlled sufficiently small level. This can be achieved by different means, e.g., by enrolling more participants as suggested in this work or by modifying the function to be evaluated (e.g., by injecting noise in the output). In the context of

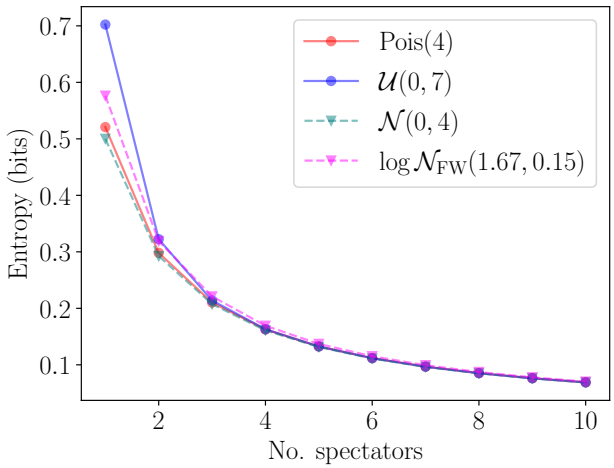


Figure 8.6: Comparing target's absolute entropy loss for discrete $H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S)$ and continuous $h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)$ distributions.

the average computation we consider in this work, the information disclosure can be reduced to any desired level by controlling the number of participants. This eliminates the need for function modification. Nevertheless, we provide a detailed analysis of the suitability of differential privacy (DP) for this application. In short, when the number of participants is large, information disclosure from the output is very low and DP is not needed, but when the number of participants is small, applying a DP mechanism results in distortion that impairs utility.

We start with a brief overview of the fundamentals of DP. The core tenet of DP involves restricting the information learned about a single individual within a sensitive dataset; this is normally accomplished by introducing noise to the result. Formally, DP is defined as follows:

Definition 5: (ε, δ)-differential privacy [74]

Let $\mathcal{M}$ be a mechanism that takes an input database D of size n supported by $\mathcal{D}$ and produces a randomized output in the set S. Then $\mathcal{M}$ is considered to be (ε, δ)-*differentially private* for $\varepsilon, \delta \geq 0$ if

for all *adjacent* databases D and D' (differ by a single entry) and all sets $S \subseteq \text{Range}(\mathcal{M})$ if

$$\Pr\left[\mathcal{M}(D) \in S\right] \leq \exp(\varepsilon) \cdot \Pr\left[\mathcal{M}(D') \in S\right] + \delta,$$

where $\text{Range}(\mathcal{M})$ is the set of all possible outputs of the mechanism $\mathcal{M}$.

The above definition encapsulates both "pure" DP ($\delta = 0$) and "approximate" DP which allows an additive privacy loss of $\delta > 0$, where δ is negligible in the size of the database. We formulate our arguments under the stronger assumption of pure DP.

Though many differentially private mechanisms exist (see, e.g., [118, 127]), we restrict our view to the *Laplace mechanism* [74] due to its well-established nature and reliable performance for low-dimensional queries [134]. We construct the one-dimensional Laplace mechanism since the output of the average function is a single value. We require the notion of *function sensitivity*, i.e., the maximum difference in the output of f when applied to two adjacent datasets:

Definition 6: Function sensitivity [73]

Let $f : \mathcal{D}^n \to \mathcal{R}$. The *sensitivity* of f is $\Delta f = \max_{D,D'} |f(D) - f(D')|$, where D and D' are adjacent.

We can now formally define the Laplace mechanism:

Definition 7: Laplace mechanism [73]

Let $f : \mathcal{D}^n \to \mathcal{R}$. Given $\varepsilon \geq 0$ and sensitivity Δf, the *Laplace mechanism* is defined as $\mathcal{M}(D) = f(D) + L$, where $L \sim \text{Lap}(\Delta f / \varepsilon)$ is a Laplace random variable.

To gauge how DP is not an impactful (and potentially detrimental) solution for the average, we compute how much noise is required to maintain differential privacy for a given ε. The ab-

solute error introduced by a mechanism $\mathcal{M}$ (or, analogously, the "utility" of the mechanism in the literature) is measured via a *loss function* ℓ. The function $\ell(f(D), \mathcal{M}(D))$ refers to the loss of an individual user when the output of the function is $f(D)$ and the mechanism's (perturbed) output is $\mathcal{M}(D)$. The choice of ℓ is arbitrary, but most DP literature [88] uses the mean error $\ell(f(D), \mathcal{M}(D)) = |f(D) - \mathcal{M}(D)|$, and thus we use it for our analysis. Given a statistical significance α, it can be shown (via application of Chernoff bounds) that the error introduced by the Laplace mechanism is bounded by

$$\ell(f(D), \mathcal{M}(D)) = |f(D) - \mathcal{M}(D)| \leq \left(\frac{\Delta f}{\varepsilon}\right) \ln\left(\frac{1}{\alpha}\right),$$

with probability $1 - \alpha$ (i.e., the confidence level). We interpret this bound as the "worst-case" upper bound on the distance between the true and the perturbed outputs. To transform this bound into a measurable quantity, we compare it against the expected value of the function with no distortion applied as a percentage, i.e., the relative error.

For this discussion, we consider participants' inputs are modeled by the uniform distribution $X_i \sim \mathcal{U}(0, N-1)$ since the sensitivity of f can be exactly derived for bounded distributions. For the average with uniformly distributed inputs, the function sensitivity is exactly $\Delta f = N - 1$ since the output of the function o can differ by at most $N - 1$ if one individual does not participate. Given the expected value for the computation sum of n uniform random variables of $\mathbb{E}\left[\sum_i X_i\right] = n\frac{N-1}{2}$, the maximum possible error (denoted by ∇) introduced by the mechanism is

$$\nabla = \frac{(\Delta f/\varepsilon)\ln(1/\alpha)}{n(N-1)/2} = \frac{2\ln(1/\alpha)}{\varepsilon n}.$$

This quantity is monotonically decreasing in n such that as n increases, the bound on the error ∇ shrinks to zero. This echoes the sentiment in DP literature that databases are typically assumed to be large.

Consider the example of 6 participants (5 spectators, 1 target) with $X_i \sim \mathcal{U}(0,7)$, $\varepsilon = 1$, and a standard [93, 134] 95% confidence level ($\alpha = 0.05$). From our analysis in Section 8.1.1, this ensures the relative entropy loss for the target is at most 5%. However, the relative error introduced by the Laplace mechanism is at most $\nabla = 99.8\%$, implying that the output of the computation can vary so drastically as to render the output unusable. Placing this in the context of the Boston gender pay gap study, where the goal was to determine the difference between male and female average salaries, the error is too large for reliable decision-making. Imposing a stricter bound on the relative entropy loss of 1% by increasing the number of participants to 25 reduces the upper bound on the error to $\nabla = 24.0\%$. We find disclosure of 1% of the input's entropy (e.g., about 0.03 bits for this application) to be acceptable, at which point there is no longer a need to use DP, and we can output precise results.

To summarize, increasing the number of participants is a natural mechanism for lowering information disclosure for the average computation. DP is of limited utility in this context, as applying it to the setting where the number of participants cannot be increased results in utility loss.

8.2 Min-Entropy Analysis

We treat min-entropy as an alternative to Shannon entropy, which was studied in the context of information flow by Smith [152]. While we are unable to go as far in our analysis as in the case

of (Shannon) entropy, we certainly observe similar trends. We begin by defining the concept of *vulnerability*:

Definition 8: Vulnerability, [152]

Given a discrete random variable X with support $\mathcal{X}$, the *vulnerability* of X, denoted by $V_\infty(X)$ over the unit interval $[0, 1]$ is given by $V_\infty(X) = \max_{x \in \mathcal{X}} \Pr(X = x)$.

The vulnerability $V_\infty(X)$ is interpreted as the worst-case probability that an adversary could guess the value of X in one attempt. If m guesses are allowed, the adversary's success probability is at most $mV_\infty(X)$. The implication is that if the vulnerability with a practical number of m guesses is significant, then $V_\infty(X)$ must also be significant. Since the vulnerability is a probability, we can convert it to an entropy measure (in bits) by taking the logarithm of $V_\infty(X)$. Conveniently, this is exactly the definition of *min-entropy* $H_\infty(X)$:

$$H_\infty(X) = \log \frac{1}{V_\infty(X)}.$$

Smith's [152] motivation for departing from Shannon entropy stems from its ineffectiveness of properly assessing the threat the output Y has on its input X.

Since our analysis studies the relationship between input and output random variables (i.e. X_T and O), a necessary extension is the *conditional vulnerability*, which specifies the expected probability of guessing X in one try, given that Y is observed:

Definition 9

Given two random variables X and Y with supports $\mathcal{X}$ and $\mathcal{Y}$, respectively, the *conditional vulnerability* $V_\infty(X \mid Y)$ is

$$V_\infty(X \mid Y) = \sum_{y \in \mathcal{Y}} \Pr(Y = y) \cdot V_\infty(X \mid Y = y),$$

where $V_\infty(X \mid Y = y) = \max_{x \in \mathcal{X}} \left(\Pr(X = x \mid Y = y)\right)$.

Having established the necessary foundations of min-entropy, we are equipped to extend our single-execution analysis of Section 8.1 from the perspective of min-entropy:

Definition 10

The attacker's weighted average min-entropy (awae_∞) of a target $\vec{X}_T$ attacked by parties A is defined for all $\vec{x}_A \in \mathcal{A}$ as

$$\begin{aligned}\text{awae}_\infty(\vec{x}_A) &= H_\infty(X_T \mid O, \vec{X}_A = \vec{x}_A) \\ &= -\log \sum_{o \in \mathcal{O}} \Pr(O = o \mid \vec{X}_A = \vec{x}_A) \cdot V_\infty(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o),\end{aligned}$$

where $V_\infty(\vec{X}_T \mid \vec{X}_A = \vec{x}_A, O = o)$ is the conditional vulnerability defined above.

The above definition is a concrete min-entropy specification of Ah-Fat and Huth's [13] generalized awae, which is parameterized by α and a gain function g. We can manipulate Definition 10 into terms consistent with Section 8.1 by plugging in the expression for conditional vulnerability:

$$\begin{aligned}\text{awae}_\infty(\vec{x}_A) &= -\log \sum_{o \in \mathcal{O}} \Pr(O = o \mid \vec{X}_A = \vec{x}_A) \cdot \left(\max_{\vec{x}_T \in \mathcal{T}} \Pr(\vec{X}_T = \vec{x}_T \mid \vec{X}_A = \vec{x}_A, O = o)\right) \\ &= -\log \sum_{o \in \mathcal{O}} \Pr(O = o \mid \vec{X}_A = \vec{x}_A)\end{aligned}$$

$$\cdot \left(\max_{\vec{x}_T \in \mathcal{T}} \frac{\Pr(O = o \mid \vec{X}_T = \vec{x}_T, \vec{X}_A = \vec{x}_A) \cdot \Pr(\vec{X}_T = \vec{x}_T)}{\Pr(O = o \mid \vec{X}_A = \vec{x}_A)} \right)$$

$$= -\log \sum_{o \in \mathcal{O}} \left(\max_{\vec{x}_T \in \mathcal{T}} \Pr(O = o \mid \vec{X}_T = \vec{x}_T, \vec{X}_A = \vec{x}_A) \cdot \Pr(\vec{X}_T = \vec{x}_T) \right).$$

In the second line we invoked Bayes' theorem, and in the third line we observed that the denominator is a constant factor in the max expression and could thus be factored out and subsequently cancelled with the leading $\Pr(O = o \mid \vec{X}_A = \vec{x}_A)$.

In Claim 1, we proved $\text{awae}(\vec{x_A})$ was independent of the attacker's input $\vec{x}_A$. Conversely, $\text{awae}_\infty(\vec{x_A})$ cannot be simplified further to prove the claim holds. Hence, we conjecture the following:

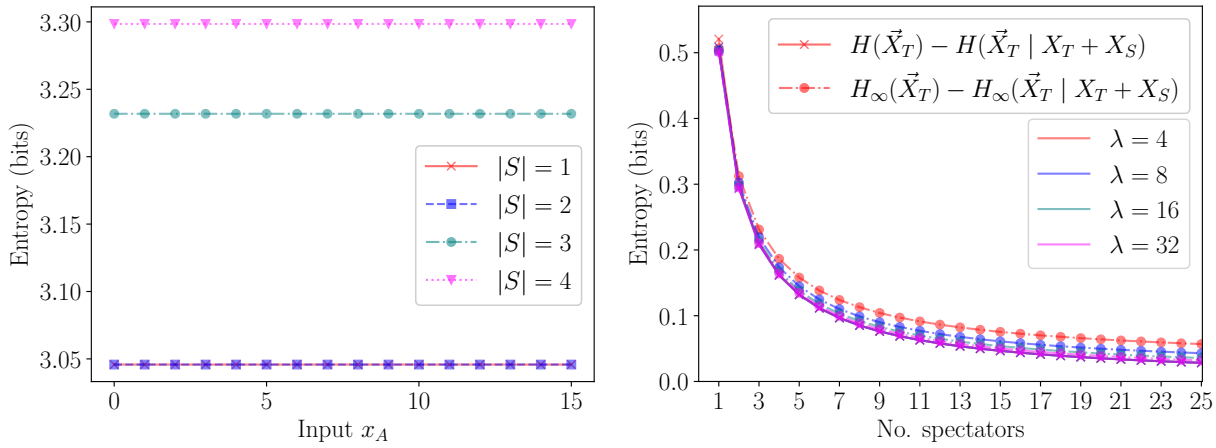
Conjecture 1

$\text{awae}_\infty(\vec{x_A})$ is independent of attacker's input vector $\vec{x_A}$.

We can, however, repeat the calculation of Figure 8.1 using min-entropy. In Figure 8.7a, we once again observe the same behavior that the adversarial knowledge does not change by varying its inputs into the computation.[2] This suggests the conjecture holds for the average salary computation. Hence, we assume such in our subsequent analysis.

The next logical step is to examine how the effect transitioning from Shannon entropy to min-entropy has on the absolute loss. We compute and display both absolute losses in Figure 8.7b, where participants' inputs are modeled by the Poisson distribution (as in Section 8.1.1). As observed previously in Figure 8.2b, the Shannon absolute loss curves all overlap each other. Interestingly, we observe the min-entropy absolute loss curves converge towards their Shannon counterparts as λ grows. This suggests that for a sufficiently large statistical parameter, the choice of

[2] Interestingly, $\text{awae}_\infty(\vec{x_A})$ is the same for $|S| = 1$ and $|S| = 2$ and the curves overlap on the plot.



(a) The $\text{awae}_\infty(\vec{x}_A)$ using uniformly distributed inputs over $\mathcal{U}(0, 15)$ with a different numbers of spectators $|S|$.

(b) Comparing the Shannon and min-entropy absolute losses using Poisson distribution with $\text{Pois}(\lambda)$, varying λ with $|T| = 1$.

Figure 8.7: Min-entropy analysis.

metric used to represent information disclosure is less impactful.

8.3 Mixed Distribution Parameters

Up to this point, we have assumed that all participants' inputs are sampled from identically distributed random variables. However, we can relax this assumption and investigate if/how the information disclosure changes if parties' inputs are non-identically distributed. For example, employee salaries may differ slightly from company to company, while still following the same distribution. We can model this by adjusting the statistical parameters of individual participants.

We begin by formalizing the notion of participant "groups". Define $\mathbb{G}$ as a finite set of statistical distributions, from which participants' inputs can be sourced. For example, if we have two groups B and C of normally distributed inputs parameterized by $\mathcal{N}(0, \sigma_{\mathsf{B}}^2)$ and $\mathcal{N}(0, \sigma_{\mathsf{C}}^2)$ (where $\sigma_{\mathsf{B}}^2 \neq \sigma_{\mathsf{C}}^2$), respectively, then $\mathbb{G} = \{\mathsf{B}, \mathsf{C}\}$. This formulation poses two interesting directions for introducing participant group identities, i.e., correspondence of a participant to one of the distribution groups,

into our analysis:

- **Group identities of individual participants are known.** The first setting we consider is that the identity, i.e., the group, of each individual participant is known. In practice, this is realized by multiple entities with inputs modeled by different statistical distributions contributing to a computation, where the number of inputs submitted by each is publicly available.

- **Group identities of individual participants are unknown.** Conversely, we have the scenario where we have knowledge of the possible distribution groups participants can belong to with anticipated likelihoods, but the group identity of an individual party is not known. This is objectively more general than the first category but requires knowledge in the form of the probabilities of an arbitrary participant belonging to each group.

It is therefore of interest to revisit our prior conclusions under the known *and* unknown group identity generalizations (denoted by Cases 1 and 2, respectively) since both formulations bear operational significance.

Entropy loss as a result of computation participation. The first conclusion we revisit is Claim 1 since it is integral to our analysis as a whole. The claim states that the information disclosure from the average function output is independent of the attackers' inputs. Based on this result, our subsequent analysis enabled us to derive expressions for awae.

In the current generalized setting, Claim 1 remains true for both Cases 1 and 2, since the derivation in the proof of Claim 1 itself remains unchanged. However, we must adjust Equation 8.3 such that participant group membership is captured by our entropy measure. We recall our definitions

of entropy remaining after participation and the absolute entropy loss:

$$H\left(\vec{X}_T \mid X_T + X_S\right) = H\left(\vec{X}_T\right) + H\left(X_S\right) - H\left(X_T + X_S\right) \tag{8.4}$$

$$H(\vec{X}_T) - H(\vec{X}_T \mid X_T + X_S) = H\left(X_T + X_S\right) - H\left(X_S\right) \tag{8.5}$$

Accounting for group identities, we introduce the *participant identity random variable* ID_{P_i} supported by $\mathbb{G}$. This corresponds to the group identity of participant P_i, and we denote $\mathsf{id}_{P_i} \in \mathbb{G}$ as the value ID_{P_i} takes. We similarly denote $\vec{\mathsf{ID}}_P = (\mathsf{ID}_{P_1}, \ldots, \mathsf{ID}_{P_m})$ as a multidimensional random variable, with $\vec{\mathsf{id}}_P$ as the vector of individual values of the same size.

At this point, our analysis splits into two directions based on the knowledge of individual group identities. *Case 1:* If participant group identities are available (i.e., $\vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T$ and $\vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S$), Equation 8.4 becomes:

$$\begin{aligned} H\left(\vec{X}_T \mid X_T + X_S, \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) = & H\left(\vec{X}_T \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T\right) + H\left(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \\ & - H\left(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right). \end{aligned} \tag{8.6}$$

Since we have exact knowledge of each participant's identity, we can explicitly partition the input random variables accordingly. Therefore, all the above quantities are computable with minimal deviation from our original analysis. For instance, if we recall our earlier example with two participant groups B and C, the term $H\left(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right)$ can be computed as:

$$\begin{aligned} H\left(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) &= H\left(\sum_{j \in S} X_{S_j} \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \\ &= H\left(\left(\sum_{j \in S \colon \mathsf{id}_{S_j} = \mathsf{B}} X_{S_j} + \sum_{j \in S \colon \mathsf{id}_{S_j} = \mathsf{C}} X_{S_j}\right) \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right). \end{aligned}$$

The absolute entropy loss directly follows from Equation 8.6, and is computed as:

$$H\left(\vec{X}_T \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T\right) - H\left(\vec{X}_T \mid X_T + X_S, \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \\ = H(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) - H(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S). \tag{8.7}$$

Under this generalization, the group to which a target belongs impacts how much information is disclosed from the computation, i.e., the disclosure can fall within a range based on the values $\vec{\mathsf{id}}_T$ can take. We determine the worst-case information disclosure by iterating over all possible target identities and taking the maximum:

$$\max_{\vec{\mathsf{id}}_T} \left(H(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) - H(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S)\right).$$

We can further refine our earlier notation established in Chapter 7 to encompass participant group identities (applicable to both targets and spectators). Let $P_{\mathsf{G}} \subset P$ be the set of participants belonging to group $\mathsf{G} \in \mathbb{G}$. The sum of random variables modeling participant inputs is given as

$$X_S = \sum_{\mathsf{G} \in \mathbb{G}} \sum_{i \in P_{\mathsf{G}}} X_{P_i} = \sum_{\mathsf{G} \in \mathbb{G}} X_{P_{\mathsf{G}}},$$

where $X_{P_{\mathsf{G}}} = \sum_{i \in P_{\mathsf{G}}} X_{P_i}$.

Case 2: When the group identities of individual inputs are not known and only the probability of belonging to a given group is given, the procedure for evaluating the information disclosure changes. The probability mass and density functions, respectively, for the participant inputs ran-

dom variables are now:

$$\Pr\left(\vec{X}_P = \vec{x}_P\right) = \sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) \Pr\left(\vec{X}_P = \vec{x}_P \mid \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right)$$

$$f\left(\vec{x}_P\right) = \sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) f\left(\vec{x}_P \mid \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right).$$

For a participant set P, there are $|\mathbb{G}|^{|P|}$ possible identity configurations, such that the number of terms in the summation is exponential in the number of participants and/or size of the group identity set. The Shannon and differential entropies are now computed as:

$$\begin{aligned} H(\vec{X}_P) = -\sum_{\vec{x}_P \in \mathcal{P}} &\left(\sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) \Pr\left(\vec{X}_P = \vec{x}_P | \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right)\right) \\ &\cdot \log\left(\sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) \Pr\left(\vec{X}_P = \vec{x}_P | \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right)\right), \end{aligned} \tag{8.8}$$

$$\begin{aligned} h(\vec{X}_P) = -\int_{\mathcal{P}} &\left(\sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) f\left(\vec{x}_P \mid \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right)\right) \\ &\cdot \log\left(\sum_{\vec{\mathsf{id}}_P} \Pr\left(\vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right) f\left(\vec{x}_P \mid \vec{\mathsf{ID}}_P = \vec{\mathsf{id}}_P\right)\right) d\vec{x}_P. \end{aligned} \tag{8.9}$$

The fundamental difference between the entropy calculation under this generalization and the analysis conducted in Sections 8.1.1 and 8.1.2 is that *the entropy of these random variables is no longer exactly modeled by the input distribution itself* (e.g., Poisson, uniform, Gaussian, log-normal). Furthermore, for continuous input distributions, our previous approach of leveraging closed-form expressions is no longer applicable when group identities are unknown – the information disclosure must be computed numerically, rather than exactly.

Parameter independence of the absolute loss for normally distributed inputs. The next con-

clusion we revisit is Claim 2, which previously stated that for normally distributed inputs, the absolute entropy loss depends only on the number of targets and spectators present in the computation. This conclusion changes in the generalized setting when the participants' inputs are no longer identically distributed, as we demonstrate below.

Case 1: Let σ_{G}^2 be the standard deviation of the participants' inputs that belong to group G. Using the definitions from Section 8.1.2 for the entropy sums of identically distributed normal random variables, the entropy of $X_{P_{\mathsf{G}}}$ is $h(X_{P_{\mathsf{G}}}) = \frac{1}{2}\log\left(2\pi e\sigma_{\mathsf{G}}^2 \left|P_{\mathsf{G}}\right|\right)$. We similarly derive the following expressions needed to compute the absolute entropy loss (given in Equation 8.7):

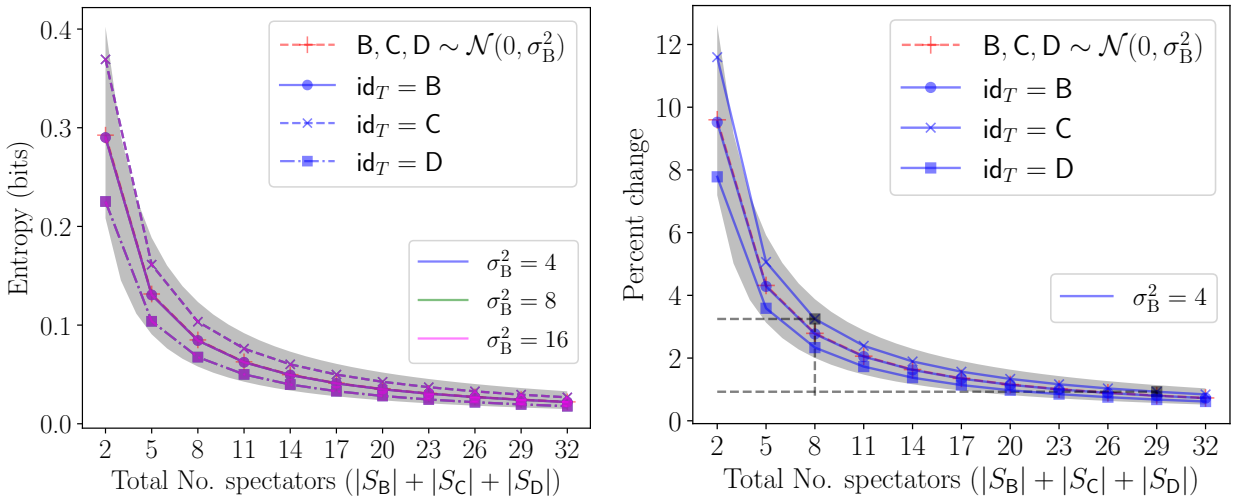
$$h(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) = \frac{1}{2}\log 2\pi e\left(\sum_{\mathsf{G}\in\mathbb{G}}\left(\sigma_{\mathsf{G}}^2 \cdot |T_{\mathsf{G}}| + \sigma_{\mathsf{G}}^2 \cdot |S_{\mathsf{G}}|\right)\right)$$
$$h(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) = \frac{1}{2}\log 2\pi e\left(\sum_{\mathsf{G}\in\mathbb{G}}\left(\sigma_{\mathsf{G}}^2 \cdot |S_{\mathsf{G}}|\right)\right).$$

Plugging these equations into our expression for the absolute entropy loss and simplifying yields:

$$h(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) - h(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) = \frac{1}{2}\log\left(\frac{\sum_{\mathsf{G}\in\mathbb{G}}\left(\sigma_{\mathsf{G}}^2 \cdot |T_{\mathsf{G}}|\right)}{\sum_{\mathsf{G}\in\mathbb{G}}\left(\sigma_{\mathsf{G}}^2 \cdot |S_{\mathsf{G}}|\right)} + 1\right).$$

Unlike the analysis in the proof of Claim 2, the standard deviations do not cancel. However, we can reformulate our interpretation of the sums of standard deviations when accounting for group identities. Let us define $\sigma_{\mathsf{B}}^2 \in \mathbb{R}_{>0}$ as the "base standard deviation" for all input random variables. Then, for all $\mathsf{G} \in \mathbb{G}$, there exists some $\delta_{\mathsf{G}} > 0$ such that $\sigma_{\mathsf{G}}^2 = \delta_{\mathsf{G}} \cdot \sigma_{\mathsf{B}}^2$. Substituting into the above expression yields:

$$h(X_T + X_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) - h(X_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S) = \frac{1}{2}\log\left(\frac{\sum_{\mathsf{G}\in\mathbb{G}}\left(\delta_{\mathsf{G}} \cdot \sigma_{\mathsf{B}}^2 \cdot |T_{\mathsf{G}}|\right)}{\sum_{\mathsf{G}\in\mathbb{G}}\left(\delta_{\mathsf{G}} \cdot \sigma_{\mathsf{B}}^2 \cdot |S_{\mathsf{G}}|\right)} + 1\right)$$



(a) Absolute entropy loss $h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)$

(b) Relative entropy loss $\frac{h(\vec{X}_T) - h(\vec{X}_T \mid X_T + X_S)}{h(\vec{X}_T)}$

Figure 8.8: Mixed distribution analysis under Case 1. The red dashed curves correspond to our baseline where all groups are identically distributed ($\mathsf{B}, \mathsf{C}, \mathsf{D} \sim \mathcal{N}(0, \sigma_B^2)$), while the remaining curves indicate the target belonging to distinct groups distributed by $\mathsf{B} \sim \mathcal{N}(0, \sigma_B^2)$, $\mathsf{C} \sim \mathcal{N}(0, 1.1^2\sigma_B^2)$, and $\mathsf{D} \sim \mathcal{N}(0, 0.9^2\sigma_B^2)$. The shaded regions illustrate the full space for the absolute entropy loss, generated from every possible spectator and group configuration.

$$= \frac{1}{2} \log \left(\frac{\sum_{\mathsf{G} \in \mathbb{G}} (\delta_{\mathsf{G}\cdot} |T_{\mathsf{G}}|)}{\sum_{\mathsf{G} \in \mathbb{G}} (\delta_{\mathsf{G}\cdot} |S_{\mathsf{G}}|)} + 1 \right).$$

The key conclusion from the above equation is that the absolute entropy loss is not directly affected by the statistical parameter σ_{G}^2, but rather the *relationship* each σ_{G}^2 has (via the scaling factor δ_{ID}) to the base standard deviation σ_{B}^2.

To demonstrate this phenomenon, we compute the absolute entropy loss when the target belongs to one of three possible groups differing by $\pm 10\%$ in their average salary. Concretely, we have B, C, or D with deviations σ_B^2, $1.1^2\sigma_B^2$, and $0.9^2\sigma_B^2$, respectively. The target is interpreted to "move" from group to group such that a consistent group size is maintained, e.g., $|S_{\mathsf{B}} \cup T| = |S_{\mathsf{C}}| = |S_{\mathsf{D}}|$ when the target is in group B. This constitutes the curves displayed in Figure 8.8. To further illustrate the best- and worst-case information disclosure, we compute the

absolute entropy loss for all possible spectator-group configurations and compose the shaded region from the maximums and minimums. Figure 8.8a reflects our observation that regardless of the base standard deviation σ_B^2 (4, 8, or 16), the curves fall on top of each other. Moreover, the absolute loss when $\mathsf{id}_T = \mathsf{B}$ is equivalent to our original computation when all groups are identically distributed. We also reproduce our relative loss experiment using the same $\pm 10\%$ salary configuration for $\sigma_B^2 = 4$ in Figure 8.8b. Achieving maximum relative losses of 5% and 1% now requires at least 6 and 27 spectators, respectively.

Case 2: When group identities of individual inputs are unknown, we refer to the definition of absolute entropy loss (Equation 8.5) alongside the expressions for the differential entropy we derived in Equation 8.9 for the required quantities and obtain:

$$
\begin{aligned}
&h(X_T + X_S) \\
&\quad = -\int_{\mathcal{T}\cup\mathcal{S}} \left(\sum_{\vec{\mathsf{id}}_T, \vec{\mathsf{id}}_S} \Pr\left(\vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) f\left(x_T + x_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \right) \\
&\qquad \cdot \log\Bigg(\sum_{\vec{\mathsf{id}}_T, \vec{\mathsf{id}}_S} \Pr\left(\vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \\
&\qquad\quad \cdot f\left(x_T + x_S \mid \vec{\mathsf{ID}}_T = \vec{\mathsf{id}}_T, \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \Bigg) d(x_T + x_S) \\
&h(X_S) = -\int_{\mathcal{S}} \left(\sum_{\vec{\mathsf{id}}_S} \Pr\left(\vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) f\left(x_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \right) \\
&\qquad \cdot \log\left(\sum_{\vec{\mathsf{id}}_S} \Pr\left(\vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) f\left(x_S \mid \vec{\mathsf{ID}}_S = \vec{\mathsf{id}}_S\right) \right) dx_S
\end{aligned}
$$

These values need to be computed numerically since, as previously stated, the random variables that represent the target and spectators' inputs no longer exactly translate to the input distribution itself. Utilizing the same group configuration as specified above under Case 1 and assuming the

probability for each identity is equally likely (for convenience), we compute the absolute loss in Figure 8.9 alongside our baseline where every participant belongs to a single group. The most interesting observation is that all the curves overlap each other, a trend originally observed in Sections 8.1.1 and 8.1.2. We note that this is likely a consequence of the experimental configuration itself (groups' salaries differ by $\pm 10\%$, identity probabilities are equally likely).

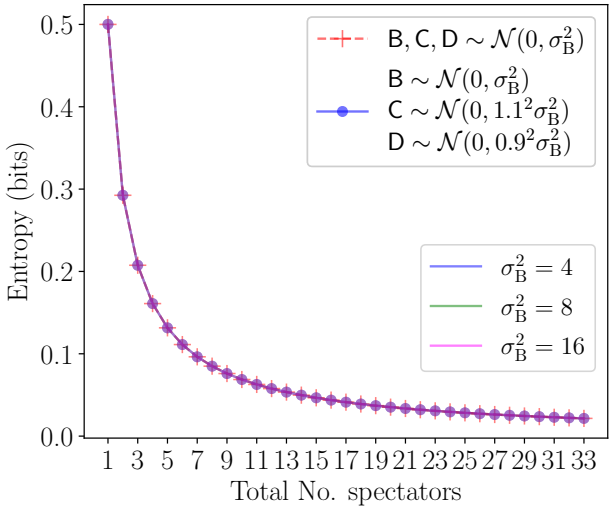


Figure 8.9: Mixed distribution analysis under Case 2, where the probability of an arbitrary participant belonging to any specific group is equally likely, i.e., $\Pr(\mathsf{ID}_P = \mathsf{B}) = \Pr(\mathsf{ID}_P = \mathsf{C}) = \Pr(\mathsf{ID}_P = \mathsf{D}) = 1/3$.

Chapter 9

Average Salary: Multiple Executions

A natural generalization of the results of the prior chapter is to consider executing the average salary computation more than once. For example, after running the Boston gender pay gap study once, the same computation was executed the following year with an extended set of participants. In this case, if the time interval between the executions is small enough such that the inputs do not change between the executions or change minimally, one would expect that repeated participations would lead to additional information disclosure compared to a single execution. Thus, we analyze in this chapter the case of two or more executions and demonstrate their impact on the participants.

9.1 Two Executions

We first consider the case of two evaluations of the average salary computation. We consider both cases when a target contributes its input to both executions and when the target participates only in one of the executions and the other takes place without the target, but on related inputs. Both cases result in additional information disclosure compared to a single execution, which we

quantify in this chapter.

We partition the set of spectators S into the following subsets:

- spectators present only in the first execution $S_1 \subset S$,
- spectators present only in the second execution $S_2 \subseteq S \setminus S_1$,
- and spectators present in both executions $S_{12} = S \setminus (S_1 \cup S_2)$.

A person participating more than once (target or spectator) enters the same input into both executions.

When the target participates in both executions, we have:

$$O_1 = \sum_i X_{T_i} + \sum_{i \in S_{12}} X_i + \sum_{i \in S_1} X_i = X_T + X_{S_{12}} + X_{S_1}$$

$$O_2 = \sum_i X_{T_i} + \sum_{i \in S_{12}} X_i + \sum_{i \in S_2} X_i = X_T + X_{S_{12}} + X_{S_2}.$$

The random variables O_1 and O_2 are *not* independent, as they both are comprised of X_T and $X_{S_{12}}$. We therefore want to compute the conditional entropy (using differential entropy notation):

$$h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T, O_1, O_2) - h(O_1, O_2). \tag{9.1}$$

Claim 3

The above conditional entropy can be expressed as

$$h(\vec{X}_T | O_1, O_2) = h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}) - h(O_1, O_2). \tag{9.2}$$

Proof. Simplifying the first term of Equation 9.1 using the chain rule of entropy $h(X, Y) = h(X \mid$

$Y) + h(Y)$ [57], we obtain:

$$\begin{aligned} h(\vec{X}_T, O_1, O_2) &= h(\vec{X}_T, X_T + X_{S_{12}} + X_{S_1}, X_T + X_{S_{12}} + X_{S_2}) \\ &= h(\vec{X}_T) + h(X_T + X_{S_{12}} + X_{S_1} \mid \vec{X}_T) \\ &\quad + h(X_T + X_{S_{12}} + X_{S_2} \mid X_T + X_{S_{12}} + X_{S_1}, \vec{X}_T). \end{aligned}$$

Using the fact that all participants' inputs are independent, we have:

$$\begin{aligned} h(\vec{X}_T, O_1, O_2) &= h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}) + h(X_T + X_{S_{12}} + X_{S_2}, X_T + X_{S_{12}} + X_{S_1} \mid \vec{X}_T) \\ &\quad - h(X_T + X_{S_{12}} + X_{S_1} \mid \vec{X}_T) \\ &= h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}) + h(X_{S_{12}} + X_{S_2}, X_{S_{12}} + X_{S_1}) - h(X_{S_{12}} + X_{S_1}) \\ &= h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}). \end{aligned}$$

The second term of Equation 9.1 can be rewritten as:

$$\begin{aligned} h(O_1, O_2) &= h(X_T + X_{S_{12}} + X_{S_1}, X_T + X_{S_{12}} + X_{S_2}) \\ &= h(X_T + X_{S_{12}} + X_{S_1}) + h(X_T + X_{S_{12}} + X_{S_2} \mid X_T + X_{S_{12}} + X_{S_1}), \end{aligned}$$

but cannot be simplified further. Therefore, the final expression of the conditional entropy is

$$h(\vec{X}_T | O_1, O_2) = h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}) - h(O_1, O_2). \qquad \square$$

In the special case when no spectators participate in both executions (i.e., $S_{12} = \emptyset$), the middle term simplifies to $h(X_{S_1}) + h(X_{S_2})$.

When the target participates only in one of the experiments, we define executions O_1' and O_2', which are the same as O_1 and O_2, respectively, except that the target's inputs are not included. For instance, $O_1' = X_{S_{12}} + X_{S_1}$. The relevant entropies in that case are $h(\vec{X}_T|O_1', O_2)$ and $h(\vec{X}_T|O_1, O_2')$.

The above requires us to introduce the definition of joint entropy of correlated random variables. Now, the normal distribution stands out among those considered in Section 8.1 as a suitable candidate for our analysis. The generalized multivariate normal distribution is well-studied and has a closed-form differential entropy, which we discuss next.

9.1.1 Bivariate Normal Distributions

Evaluating Equation 9.2 requires defining the differential entropy of a multivariate normal random variable. We then derive the necessary core parameters for our distributions and use them to compute the conditional entropy.

Let $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ be a single normal random variable as defined in Chapter 7. We define $\vec{X} = (X_1, \ldots, X_k)^{\mathrm{T}}$ to be a general multivariate normal distribution of a k-dimensional random vector, with $\vec{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Here, $\boldsymbol{\mu} \in \mathbb{R}^k$ is the mean vector specified as

$$\boldsymbol{\mu} = \mathbb{E}[\vec{X}] = (\mathbb{E}[X_1], \mathbb{E}[X_2], \ldots, \mathbb{E}[X_k])^{\mathrm{T}} = (\mu_1, \mu_2, \ldots, \mu_k)^{\mathrm{T}},$$

and $\boldsymbol{\Sigma} \in \mathcal{R}^{k \times k}$ is the $k \times k$ covariance matrix with each element defined as $\Sigma_{i,j} = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right] = \mathrm{Cov}\left[X_i, X_j\right]$. The differential entropy of the multivariate normal distribution $\vec{X}$ is given by

$$h(\vec{X}) = \frac{1}{2} \log\left((2\pi e)^k \det \boldsymbol{\Sigma}\right),$$

[57, Chapter 8.4] where $\det \boldsymbol{\Sigma}$ is the determinant of the covariance matrix. The next step is to char-

acterize our multivariate distributions and determine their covariance matrices. We also derive their mean vectors which are used for intermediate results.

To compute the second and third terms of Equation 9.2, we formalize the bivariate distributions $\vec{S} = (X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2})^{\mathrm{T}}$ and $\vec{O} = (O_1, O_2)^{\mathrm{T}}$. We denote $\mu_P = \sum_i \mu_{P_i}$ and $\sigma_P^2 = \sum_i \sigma_{P_i}^2$ as the sum of the means and standard deviations, respectively, of all participants within a group P. Note that the mean is absent from the formula for the differential entropy, and therefore we can safely assume all $\mu_i = 0$. Starting with $\vec{O}$, we invoke the linearity of the expectation for the mean vector:

$$\boldsymbol{\mu}_{\vec{O}} = \begin{pmatrix} \mathbb{E}[O_1] \\ \mathbb{E}[O_2] \end{pmatrix} = \begin{pmatrix} \mathbb{E}[X_T + X_{S_{12}} + X_{S_1}] \\ \mathbb{E}[X_T + X_{S_{12}} + X_{S_2}] \end{pmatrix} = \begin{pmatrix} \mu_T + \mu_{S_{12}} + \mu_{S_1} \\ \mu_T + \mu_{S_{12}} + \mu_{S_2} \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}.$$

For the covariance matrix, using the properties $\mathrm{Cov}[X, X] = \mathrm{Var}[X] = \sigma_X^2$ and $\mathrm{Cov}[X, Y] = \mathrm{Cov}[Y, X]$ yields

$$\begin{aligned} \boldsymbol{\Sigma}_{\vec{O}} &= \begin{pmatrix} \mathrm{Cov}[O_1, O_1] & \mathrm{Cov}[O_1, O_2] \\ \mathrm{Cov}[O_2, O_1] & \mathrm{Cov}[O_2, O_2] \end{pmatrix} = \begin{pmatrix} \mathrm{Var}[O_1] & \mathrm{Cov}[O_1, O_2] \\ \mathrm{Cov}[O_1, O_2] & \mathrm{Var}[O_2] \end{pmatrix} \\ &= \begin{pmatrix} \sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_1}^2 & \mathrm{Cov}[O_1, O_2] \\ \mathrm{Cov}[O_1, O_2] & \sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_2}^2 \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \mathrm{Cov}[O_1, O_2] \\ \mathrm{Cov}[O_1, O_2] & \sigma_2^2 \end{pmatrix}. \end{aligned}$$

The expression for $\mathrm{Cov}[O_1, O_2]$ can be stated as follows:

Claim 4

$\mathrm{Cov}[O_1, O_2] = \sigma_T^2 + \sigma_{S_{12}}^2$ if S_{12} is non-empty, and $\mathrm{Cov}[O_1, O_2] = \sigma_T^2$ otherwise.

Proof.

$$\begin{aligned}
\text{Cov}\left[O_1, O_2\right] &= \mathbb{E}\left[(O_1 - \mu_1)(O_2 - \mu_2)\right] = \mathbb{E}\left[O_1 O_2 - \mu_2 O_1 - \mu_1 O_2 + \mu_1 \mu_2\right] \\
&= \mathbb{E}\left[\left(X_T + X_{S_{12}} + X_{S_1}\right)\left(X_T + X_{S_{12}} + X_{S_2}\right)\right] \mathbb{E}\left[\mu_2\left(X_T + X_{S_{12}} + X_{S_1}\right)\right] \\
&\quad - \mathbb{E}\left[\mu_1\left(X_T + X_{S_{12}} + X_{S_2}\right)\right] + \mathbb{E}\left[\mu_1 \mu_2\right] \\
&= \mathbb{E}\left[X_T^2\right] + \mathbb{E}\left[X_{S_{12}}^2\right] + 2\mathbb{E}\left[X_T X_{S_{12}}\right] + \mathbb{E}\left[X_T X_{S_1}\right] + \mathbb{E}\left[X_T X_{S_2}\right] \\
&\quad + \mathbb{E}\left[X_{S_{12}} X_{S_1}\right] + \mathbb{E}\left[X_{S_{12}} X_{S_2}\right] + \mathbb{E}\left[X_{S_1} X_{S_2}\right] \\
&\quad - \mu_2(\overbrace{\mathbb{E}\left[X_T\right] + \mathbb{E}\left[X_{S_{12}}\right] + \mathbb{E}\left[X_{S_1}\right]}^{\mu_1}) - \mu_1(\underbrace{\mathbb{E}\left[X_T\right] + \mathbb{E}\left[X_{S_{12}}\right] + \mathbb{E}\left[X_{S_2}\right]}_{\mu_2}) \\
&\quad + \mu_1 \mu_2
\end{aligned}$$

Exploiting the definition of variance $\mathbb{E}\left[X^2\right] = \sigma_X^2 + \mu_X^2$ and fundamental properties of expectation:

$$\begin{aligned}
\text{Cov}\left[O_1, O_2\right] &= \sigma_T^2 + \sigma_{S_{12}}^2 - \mu_1 \mu_2 \\
&\quad + \underbrace{\mu_T^2 + \mu_{S_{12}}^2 + 2\mu_T \mu_{S_{12}} + \mu_T \mu_{S_1} + \mu_T \mu_{S_2} + \mu_{S_{12}} \mu_{S_1} + \mu_{S_{12}} \mu_{S_2} + \mu_{S_1} \mu_{S_2}}_{=\mu_1 \mu_2} \\
&= \sigma_T^2 + \sigma_{S_{12}}^2.
\end{aligned}$$

Clearly, if $S_{12} = \emptyset$, the above result simplifies to $\text{Cov}\left[O_1, O_2\right] = \sigma_T^2$. This result is intuitive since the covariance measures the strength of correlation between two random variables, and O_1 and O_2 are both comprised of X_T and $X_{S_{12}}$. □

The final parameters of the bivariate distribution $\vec{O}$ are

$$\boldsymbol{\mu}_{\vec{O}} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \boldsymbol{\Sigma}_{\vec{O}} = \begin{pmatrix} \sigma_1^2 & \sigma_T^2 + \sigma_{S_{12}}^2 \\ \sigma_T^2 + \sigma_{S_{12}}^2 & \sigma_2^2 \end{pmatrix}.$$

Repeating this procedure for the spectator joint distribution $\vec{S}$ yields a similar set of parameters:

$$\boldsymbol{\mu}_{\vec{S}} = \begin{pmatrix} \mu_{S_{12}} + \mu_{S_1} \\ \mu_{S_{12}} + \mu_{S_2} \end{pmatrix}, \boldsymbol{\Sigma}_{\vec{S}} = \begin{pmatrix} \sigma_{S_{12}}^2 + \sigma_{S_1}^2 & \sigma_{S_{12}}^2 \\ \sigma_{S_{12}}^2 & \sigma_{S_{12}}^2 + \sigma_{S_2}^2 \end{pmatrix}.$$

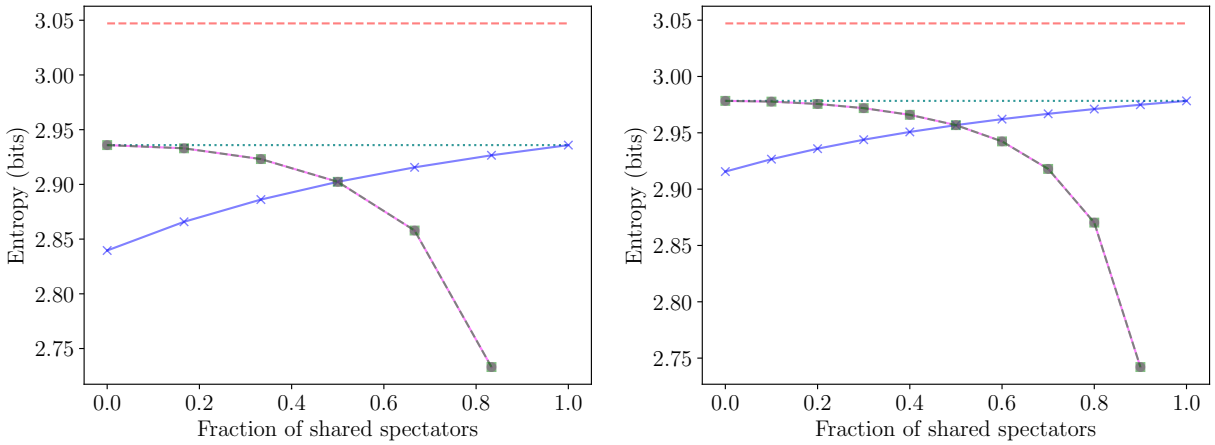
Equipped with expressions for $\boldsymbol{\Sigma}_{\vec{O}}$ and $\boldsymbol{\Sigma}_{\vec{S}}$, we are prepared to begin our experimental analysis of $h(X_T \mid O_1, O_2)$.

9.1.2 Experimental Evaluation

The above specification allows us to experimentally evaluate the target's entropy loss when inputs are normally distributed. We use normal distribution $\mathcal{N}(0, 4)$ to reasonably approximate the log-normal distribution with real data. Once again, $|T| = 1$ for concreteness and we let $|S_1| = |S_2|$ in all experiments, i.e., the number of spectators is the same in both executions.

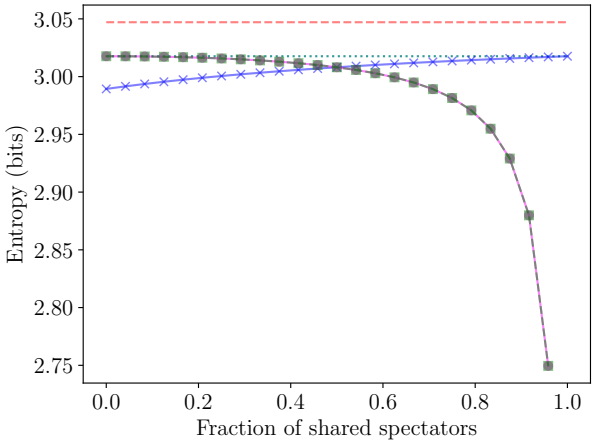
It is informative to analyze information loss as the fraction of shared spectators changes and we do so for three different computation sizes. To be as close to the setup that guarantees 1%–5% entropy loss for the log-normal distribution (5–24 spectators), we choose to execute our experiments with 6, 10, and 24 spectators (where having an even number is beneficial for illustration purposes). This corresponds to the number of non-adversarial participants when the target is absent and the number of non-adversarial participants is one higher when the target is participating.

We display the following information in Figure 9.1:



(a) $s = 6$ spectators per execution.

(b) $s = 10$ spectators per execution.



(c) $s = 24$ spectators per execution.

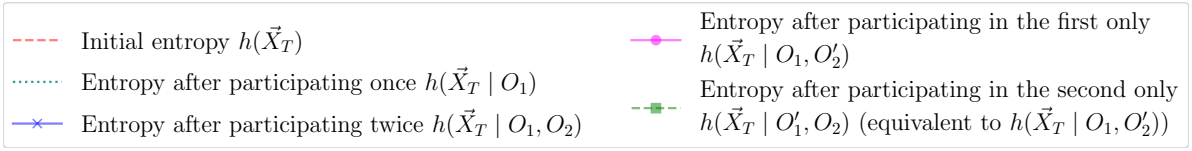


Figure 9.1: Target information loss after participating in one or two computations. Omitted: if the target participates in one experiment and all the shared spectators are reused, then $h(X_T \mid O_1, O'_2) = 0$.

- the target's initial entropy $h(\vec{X}_T)$,
- the target's entropy after a single execution $h(\vec{X}_T \mid O_1)$,
- the target's entropy after participating twice $h(\vec{X}_T | O_1, O_2)$,

- the target's entropy after participating in one of the two executions, i.e., $h(\vec{X}_T \mid O_1, O_2')$ and $h(\vec{X}_T \mid O_1', O_2)$

and plot the values as a function of the fractional overlap between two executions for a given number of spectators.

Naturally, the value of $h(\vec{X}_T \mid O_1)$ remains constant when the number of participants is fixed. We observe that when participating twice, $h(\vec{X}_T \mid O_1, O_2)$ converges to $h(\vec{X}_T \mid O_1)$ as the fraction of shared spectators increases. This is expected because at 100% overlap, we are functionally calculating $h(\vec{X}_T \mid O_1, O_1) = h(\vec{X}_T \mid O_1)$. We formalize this into the claim:

Claim 5

If the target participates in both evaluations and 100% of the spectators are reused, $h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T \mid O_1)$.

Proof. We begin by analyzing the absolute loss between the first and second evaluations when the target participates twice, namely:

$$h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2).$$

Assume all participants' inputs are normally distributed ($X_i \sim \mathcal{N}(0, \sigma^2)$). Denote $p = |P|$ as the size of an arbitrary group P (e.g., $s_{12} = |S_{12}|$), such that $X_P \sim \mathcal{N}(0, p\sigma^2)$. Simplifying the absolute loss between the first and second evaluations, we obtain:

$$\begin{aligned} h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}) - h(X_T + X_{S_{12}} + X_{S_1}) \\ - \Big(h(\vec{X}_T) + h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}) - h(O_1, O_2)\Big) \end{aligned}$$

$$= h(X_{S_{12}} + X_{S_1}) - h(X_T + X_{S_{12}} + X_{S_1}) + h(O_1, O_2)$$

$$- h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}).$$

Using the definitions from Section 9.1.1, we calculate the remaining terms as

$$\begin{aligned}
h(X_{S_{12}} + X_{S_1}) + h(X_T + X_{S_{12}} + X_{S_1}) &= \frac{1}{2} \log \left(\frac{s_{12} + s_1}{t + s_{12} + s_1} \right) \\
h(O_1, O_2) &= \frac{1}{2} \log \left((2\pi e)^2 ((t + s_{12})(s_1 + s_2) + s_1 s_2) \sigma^2 \right) \\
h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}) &= \frac{1}{2} \log \left((2\pi e)^2 (s_{12}(s_1 + s_2) + s_1 s_2) \sigma^2 \right) \\
h(O_1, O_2) - h(X_{S_{12}} + X_{S_1}, X_{S_{12}} + X_{S_2}) &= \frac{1}{2} \log \left(\frac{(t + s_{12})(s_1 + s_2) + s_1 s_2}{s_{12}(s_1 + s_2) + s_1 s_2} \right)
\end{aligned}$$

Therefore, the absolute entropy loss between the first and second evaluations is

$$\begin{aligned}
h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2) &= \left(\frac{1}{2} \log \left(\frac{s_{12} + s_1}{t + s_{12} + s_1} \right) \right) + \frac{1}{2} \log \left(\frac{(t + s_{12})(s_1 + s_2) + s_1 s_2}{s_{12}(s_1 + s_2) + s_1 s_2} \right) \\
&= \frac{1}{2} \log \left(\left(\frac{s_{12} + s_1}{t + s_{12} + s_1} \right) \left(\frac{t(s_1 + s_2) + s_{12}(s_1 + s_2) + s_1 s_2}{s_{12}(s_1 + s_2) + s_1 s_2} \right) \right).
\end{aligned}$$

Since we assume $s_1 = s_2$, the above expression simplifies to

$$h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2) = \frac{1}{2} \log \left(\left(\frac{s_{12} + s_1}{t + s_{12} + s_1} \right) \left(\frac{2t + 2s_0 + s_1}{2s_0 + s_1} \right) \right).$$

This function is monotonically decreasing when the total number of spectators is fixed to $s_{12} + s_1$, and we vary the ratio $\frac{s_{12}}{s_1} \in [0, 1]$, which is consistent with our observation that the absolute loss will converge to $h(\vec{X}_T \mid O_1)$. □

Conversely, increasing the fraction of the overlap has the inverse effect for $h(\vec{X}_T \mid O_1, O_2')$,

causing it to trend downward. At 100% overlap, $h(\vec{X}_T \mid O_1, O_2') = 0$ (point omitted from the plots). This is a consequence of effectively computing $h(\vec{X}_T \mid O_1, X_{S_{12}})$:

$$\begin{aligned} h(\vec{X}_T|O_1, X_{S_{12}}) &= h(\vec{X}_T, O_1, X_{S_{12}}) - h(O_1, X_{S_{12}}) \\ &= h(\vec{X}_T) + h(X_{S_{12}}) - (h(X_T + X_{S_{12}} \mid X_{S_{12}}) + h(X_{S_{12}})) \\ &= h(\vec{X}_T) + h(X_{S_{12}}) - (h(X_T) + h(X_{S_{12}})) = h(\vec{X}_T) - h(X_T). \end{aligned}$$

When $|T| = 1$, then $h(\vec{X}_T) = h(X_T)$, thus reducing the above equation to zero. This informs us that the output of the second computation O_2' without any unique spectators reveals the target's information entirely. We state this observation as follows:

Claim 6

If the target participates in one evaluation and 100% of the spectators are reused, $h(\vec{X}_T \mid O_1, O_2') = 0$.

Proof. We now examine the absolute entropy loss between the first and second evaluations when the target participates in only the first evaluation:

$$h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2')$$

The only difference from the prior calculation arises is replacing $h(O_1, O_2)$ with $h(O_1, O_2')$, which evaluates to

$$h(O_1, O_2) = \frac{1}{2} \log \left((2\pi e)^2 (t(s_{12} + s_2) + s_{12}(s_1 + s_2) + s_1 s_2) \sigma^2 \right),$$

such that our final expression is

$$h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2') = \frac{1}{2}\log\left(\left(\frac{s_{12}+s_1}{t+s_{12}+s_1}\right)\left(\frac{t(s_{12}+s_2)+s_{12}(s_1+s_2)+s_1s_2}{s_{12}(s_1+s_2)+s_1s_2}\right)\right)$$
$$= \frac{1}{2}\log\left(\left(\frac{s_{12}+s_1}{t+s_{12}+s_1}\right)\left(\frac{t(s_{12}+s_1)+s_1(2s_0+s_1)}{s_1(2s_0+s_1)}\right)\right).$$

This function blows up to infinity when $s_1 = s_2 = 0$, which confirms that the output of the second computation O_2' without the presence of any unique spectators reveals the target's information entirely. □

A passive result of both proofs is that all forms of absolute loss are parameter-independent, which is consistent with Claim 2.

Our next observation pertains to the point of intersection where $h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T \mid O_1, O_2')$, which occurs when 50% of the spectators are shared across the computation. This appears for the special case when the total number of spectators in a single evaluation is even. Concretely, we compare

$$\begin{aligned} h(\vec{X}_T|O_1, O_2) &= h(\vec{X}_T, O_1, O_2) - h(O_1, O_2), \\ h(\vec{X}_T|O_1, O_2') &= h(\vec{X}_T, O_1, O_2') - h(O_1, O_2'). \end{aligned} \tag{9.3}$$

It can be shown using the procedure outlined in Section 9.1 that $h(\vec{X}_T, O_1, O_2) = h(\vec{X}_T, O_1, O_2')$. Therefore, we prove the following:

Claim 7

With normally distributed inputs, the terms $h(O_1, O_2)$ and $h(O_1, O_2')$ are equal when $|S_{12}| = |S_1|$.

Proof. Following the steps used to derive the covariance matrix of $\vec{O} = (O_1, O_2)$, the covariance

matrix of $\vec{O}' = (O_1, O_2')$ is

$$\mathbf{\Sigma}_{\vec{O}'} = \begin{pmatrix} \sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_1}^2 & \sigma_{S_{12}}^2 \\ \sigma_{S_{12}}^2 & \sigma_{S_{12}}^2 + \sigma_{S_2}^2 \end{pmatrix}.$$

Recall that the differential entropy of the multivariate normal is $h(\vec{X}) = \frac{1}{2}\log\left((2\pi e)^k \det \mathbf{\Sigma}\right)$. The sole object of interest is the $\det \mathbf{\Sigma}$ term, as the remainder contributes a constant factor. We calculate

$$\begin{aligned} \det \mathbf{\Sigma}_{\vec{O}} &= (\sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_1}^2)(\sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_2}^2) - (\sigma_T^2 + \sigma_{S_{12}}^2)^2 \\ &= \sigma_T^2(\sigma_{S_1}^2 + \sigma_{S_2}^2) + \sigma_{S_{12}}^2(\sigma_{S_1}^2 + \sigma_{S_2}^2) + \sigma_{S_1}^2\sigma_{S_2}^2. \end{aligned}$$

Similarly,

$$\begin{aligned} \det \mathbf{\Sigma}_{\vec{O}'} &= (\sigma_T^2 + \sigma_{S_{12}}^2 + \sigma_{S_1}^2)(\sigma_{S_{12}}^2 + \sigma_{S_2}^2) - \sigma_{S_{12}}^4 \\ &= \sigma_T^2(\sigma_{S_{12}}^2 + \sigma_{S_2}^2) + \sigma_{S_{12}}^2(\sigma_{S_1}^2 + \sigma_{S_2}^2) + \sigma_{S_1}^2\sigma_{S_2}^2. \end{aligned}$$

Therefore, the equality $h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T \mid O_1, O_2')$ is satisfiable if and only if $\sigma_{S_{12}}^2 = \sigma_{S_1}^2$, which occurs when $|S_{12}| = |S_1|$. □

As computation designers, we can minimize information disclosure for all participants by targeting a 50% participant overlap between the first and second executions. For the configurations in Figure 9.1, at 50% overlap, the percentages of information loss from the second evaluation relative to the first evaluation are comparable for the selected number of spectators s (30.18% for $s = 6$, 31.3% for $s = 10$, and 32.45% for $s = 24$). This corresponds to the intersection points in Figure 9.1.

As we may be unable to guarantee that exactly 50% of participants overlap between two execu-

Number of evaluations the target participates in	Spectator overlap		
	40%	50%	60%
One	18.0%	31.3%	52.3%
Two	40.1%	31.3%	23.5%

Table 9.1: Percentage of information loss after two executions relative to a single execution for $s = 10$.

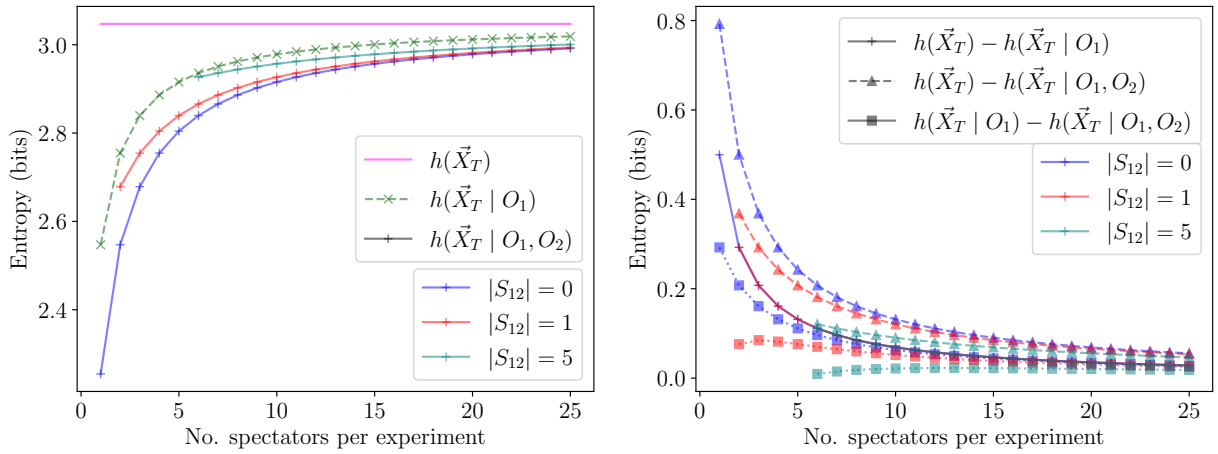
tions, we can increase our tolerance for entropy loss by inviting more participants and building a buffer to accommodate overlaps in a range, e.g., 40–60%. Using data in Figure 9.1, this information is convenient to gather for $s = 10$. That is, if we increase the fraction of overlapping spectators, single-participation targets are most at risk. The converse is true if the overlap decreases – the target suffers less exposure from participating in one evaluation. Table 9.1 summarizes the results. This means that performing two executions in the worst case costs a participant entropy loss 1.5 times higher than if only a single computation is executed. As a result, with the target entropy loss of 5% and 1%, we need to increase the number of spectators from 5 and 24 to 7 and 33, respectively.

We note that our analysis of repeated executions applies only when the inputs of the participants in the overlapping set of participants do not change. If the executions are distant enough in time that the participants' inputs significantly change, they would no longer be treated as repeated dependent executions.

9.1.3 Additional Two Executions Experiments

We examine the impact of shared spectators' presence on the target's information loss. In Figure 9.2a, we plot:

- the target's initial entropy $h(\vec{X}_T)$,
- the target's entropy after a single execution $h(\vec{X}_T \mid O_1)$, and



(a) Computing $h(X_T)$, $h(X_T \mid O_1)$, and $h(X_T \mid O_1, O_2)$ for several $|S_{12}|$ sizes.

(b) Absolute entropy loss.

Figure 9.2: Comparing the relative and absolute entropy losses of participants with normally distributed inputs. The number of spectators per experiment on the x-axis is computed as $|S_{12} \cup S_1| = |S_{12} \cup S_2|$, starting with $|S_1| = |S_2| = 1$.

- the target's entropy after two executions $h(\vec{X}_T \mid O_1, O_2)$ with a different number of spectators participating in both executions.

We vary the total number of spectators per experiment $|S_{12} \cup S_1| = |S_{12} \cup S_2|$ on the x-axis, starting with one unique spectator per experiment $|S_1| = |S_2| = 1$. The $h(\vec{X}_T \mid O_1, O_2)$ curves correspond to awae after two executions and start when the number of participants reaches their respective number of shared spectators $|S_{12}|$ in order to make an accurate comparison. A single curve for $h(\vec{X}_T \mid O_1)$ suffices since it does not use the notion of shared spectators.

We observe in Figure 9.2a that the larger the number of shared spectators for a given $|S_0|$ is, the less information is revealed about the target. These spectators function as "noise" that protects the target. The protection offered by a small number of shared spectators becomes less pronounced as the number of participants grows.

We also compute and present in Figure 9.2b the target's absolute entropy loss for the following

experiments:

- after a single execution $h(\vec{X}_T) - h(\vec{X}_T \mid O_1)$,
- after two executions $h(\vec{X}_T) - h(\vec{X}_T \mid O_1, O_2)$, and
- after the second execution $h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2)$

using a varying number of shared spectators $|S_{12}|$. We see that for each fixed number of shared spectators $|S_{12}|$, the absolute loss as a result of the first participation ($h(\vec{X}_T) - h(\vec{X}_T \mid O_1)$) is greater than the absolute loss of the second participation ($h(\vec{X}_T \mid O_1) - h(\vec{X}_T \mid O_1, O_2)$). With no shared spectators the curves converge at about 15 participants per execution, while increasing the number of shared spectators causes the curves to converge at a slower rate.

9.1.4 Mixed Distribution Parameters for Two Executions

In Section 8.3, we examined how our conclusions changed when generalizing our analysis to non-identically distributed participant inputs. We are similarly interested in how this affects our two-execution analysis. Recalling the formulation of our two-evaluation setting where spectators are present in the first, second, or both executions, we now consider these spectator subsets can be further partitioned into sub-subsets based on their group identities. Combining our two-evaluation notation and that of Section 8.3, we define $P_{k,\mathsf{G}} \subseteq P$ as the set of participants present in execution(s) $k = \{1, 2, 12\}$ belonging to group $\mathsf{G} \in \mathbb{G}$. For example, $k = 1$ means participation in the first execution, while $k = 12$ means participation in both. The sum of these participants' inputs is

$$X_{P_k} = \sum_{\mathsf{G} \in \mathbb{G}} \sum_{i \in P_{k,\mathsf{G}}} X_{P_{k,i}} = \sum_{\mathsf{G} \in \mathbb{G}} X_{P_{k,\mathsf{G}}},$$

where $X_{P_G} = \sum_{i \in P_{k,G}} X_{P_{k,i}}$.

Optimal setup for minimizing information disclosure. In Section 9.1.2, we determined that the point of intersection of the entropies $h(\vec{X}_T \mid O_1, O_2)$ and $h(\vec{X}_T \mid O_1, O_2')$ at 50% participant overlap provides the best level of protection for all types of targets, and moving in either direction (increasing or decreasing the overlap) causes the leakage to increase. We revisit our proof of Claim 7 in our generalized setting to determine whether the equality $h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T \mid O_1, O_2')$ still holds when 50% of the spectators are shared across the computation, i.e., $|S_{12}| = |S_1|$.

Considering the expansions of $h(\vec{X}_T \mid O_1, O_2)$ and $h(\vec{X}_T \mid O_1, O_2')$ in Equation 9.3, it can be shown that the terms $h(\vec{X}_T, O_1, O_2)$ and $h(\vec{X}_T, O_1, O_2')$ are equal, leaving $h(O_1, O_2)$ and $h(O_1, O_2')$ for us to compare. We perform this analysis under Case 1 (participant group identities are known), since the proof of Claim 7 relies on the existence of the closed-form expression for the differential entropy.

Computing $h(O_1, O_2)$ and $h(O_1, O_2')$ in the generalized setting requires re-formalizing the covariance matrices of the joint random variables $\vec{O} = (O_1, O_2)$ and $\vec{O}' = (O_1, O_2')$. Performing the steps outlined in Section 9.1.1 yields the following:

$$\mathbf{\Sigma}_{\vec{O}} = \begin{pmatrix} \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{1,G}|\right) & \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}|\right) \\ \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}|\right) & \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{2,G}|\right) \end{pmatrix}$$

$$\mathbf{\Sigma}_{\vec{O}'} = \begin{pmatrix} \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{1,G}|\right) & \sum_{G \in \mathbb{G}} \sigma_G^2 |S_{12,G}| \\ \sum_{G \in \mathbb{G}} \sigma_G^2 |S_{12,G}| & \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{12,G}| + |S_{2,G}|\right) \end{pmatrix}.$$

Recalling the definition of the differential entropy of a multivariate normal $h(\vec{X}) = \frac{1}{2} \log((2\pi e)^k \det \mathbf{\Sigma})$,

we compute the determinants of the above matrices as:

$$
\begin{aligned}
\det \mathbf{\Sigma}_{\vec{O}} &= \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{1,G}| \right) \right) \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{2,G}| \right) \right) \\
&\quad - \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| \right) \right)^2 \\
&= \sum_{G \in \mathbb{G}} \sigma_G^2 \, |T_G| \left(\sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left(|S_{1,G}| + |S_{2,G}| \right) \right) \right) \\
&\quad + \sum_{G \in \mathbb{G}} \sigma_G^2 \left| S_{(12,G)} \right| \left(\sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left(|S_{1,G}| + |S_{2,G}| \right) \right) \right) \\
&\quad + \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{1,G}| \right) \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{2,G}| \right) \\
\det \mathbf{\Sigma}_{\vec{O}'} &= \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|T_G| + |S_{12,G}| + |S_{1,G}| \right) \right) \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{12,G}| + |S_{1,G}| \right) \right) \\
&\quad - \left(\sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{12,G}| \right) \right)^2 \\
&= \sum_{G \in \mathbb{G}} \sigma_G^2 \, |T_G| \left(\sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left(|S_{12,G}| + |S_{2,G}| \right) \right) \right) \\
&\quad + \sum_{G \in \mathbb{G}} \sigma_G^2 \left| S_{(12,G)} \right| \left(\sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left(|S_{1,G}| + |S_{2,G}| \right) \right) \right) \\
&\quad + \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{1,G}| \right) \sum_{G \in \mathbb{G}} \sigma_G^2 \left(|S_{2,G}| \right)
\end{aligned}
$$

By inspection, we obtain that the equality $h(\vec{X}_T \mid O_1, O_2) = h(\vec{X}_T \mid O_1, O_2')$ is satisfiable if and only if $\sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left| S_{12,G} \right| \right) = \sum_{G \in \mathbb{G}} \left(\sigma_G^2 \left| S_{1,G} \right| \right)$. However, this no longer implies that the optimal configuration is at 50% overlap. For $|\mathbb{G}| > 1$, there can be multiple solutions with respect to the individual group sizes, statistical parameters, and overlap percentages such that the equality can be satisfied.

9.2 Three Executions and Beyond

The next logical step is to further generalize our analysis to three and any number M executions.

9.2.1 Three Executions

For three evaluations, there are additional possibilities for spectators to overlap between experiments. Specifically, we have:

- spectators who participate in one experiment (S_1, S_2, S_3),
- spectators who participate in two experiments, but not a third (S_{12}, S_{13}, S_{32}), and
- spectators who participate in all three experiments (S_{123}).

Let s be the (fixed) total number of spectators per experiment. For each evaluation, let superscript (τ_i) denote a target's participation flag defined as:

$$\tau_i = \begin{cases} 0 & T \text{ does } \textit{not} \text{ participate in evaluation } i \\ \\ 1 & T \text{ participates in evaluation } i \end{cases}.$$

We require $\sum_{i=1}^{3} \tau_i > 0$ to signify that the target participates at least once. Therefore, there are $2^3 - 1 = 7$ possible target configurations. For example, $(\tau_1, \tau_2, \tau_3) = (1, 0, 1)$ means the target participated in the first and third executions. We use this notation to generate expressions for all configurations of the targets' participation in evaluations. The random variables for each evaluation are:

$$O_1^{(\tau_1)} = \tau_1 \cdot X_T + X_{S_1} + X_{S_{12}} + X_{S_{13}} + X_{S_{123}} = \tau_1 \cdot X_T + X_{\hat{S}_1}$$

$$O_2^{(\tau_2)} = \tau_2 \cdot X_T + X_{S_2} + X_{S_{12}} + X_{S_{23}} + X_{S_{123}} = \tau_2 \cdot X_T + X_{\hat{S}_2}$$

$$O_3^{(\tau_3)} = \tau_3 \cdot X_T + X_{S_2} + X_{S_{23}} + X_{S_{13}} + X_{S_{123}} = \tau_3 \cdot X_T + X_{\hat{S}_3},$$

where $X_{\hat{S}_i}$ is the sum of all spectator configurations in evaluation i. If we denote $p = |P|$ as the size of an arbitrary group P such that $X_P \sim \mathcal{N}(0, p\sigma^2)$, then covariance matrix for the random vector $\vec{O}_{1,2,3} = \left(O_1^{(\tau_1)}, O_2^{(\tau_2)}, O_3^{(\tau_3)}\right)^{\mathrm{T}}$ is

$$\begin{aligned}
\mathbf{\Sigma}_{\vec{O}_{1,2,3}} &= \begin{pmatrix} \mathrm{Cov}\left[O_1^{(\tau_1)}, O_1^{(\tau_1)}\right] & \mathrm{Cov}\left[O_1^{(\tau_1)}, O_2^{(\tau_2)}\right] & \mathrm{Cov}\left[O_1^{(\tau_1)}, O_3^{(\tau_3)}\right] \\ \mathrm{Cov}\left[O_2^{(\tau_2)}, O_1^{(\tau_1)}\right] & \mathrm{Cov}\left[O_2^{(\tau_2)}, O_2^{(\tau_2)}\right] & \mathrm{Cov}\left[O_2^{(\tau_2)}, O_3^{(\tau_3)}\right] \\ \mathrm{Cov}\left[O_3^{(\tau_3)}, O_1^{(\tau_1)}\right] & \mathrm{Cov}\left[O_3^{(\tau_3)}, O_2^{(\tau_2)}\right] & \mathrm{Cov}\left[O_3^{(\tau_3)}, O_3^{(\tau_3)}\right] \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} \tau_1 \cdot t + s_1 + s_{12} \\ + s_{13} + s_{123} \end{pmatrix} & \tau_1\tau_2 \cdot t + s_{12} + s_{123} & \tau_1\tau_3 \cdot t + s_{13} + s_{123} \\ \tau_1\tau_2 \cdot t + s_{12} + s_{123} & \begin{pmatrix} \tau_2 \cdot t + s_2 + s_{12} \\ + s_{23} + s_{123} \end{pmatrix} & \tau_2\tau_3 \cdot t + s_{23} + s_{123} \\ \tau_1\tau_3 \cdot t + s_{13} + s_{123} & \tau_2\tau_3 \cdot t + s_{23} + s_{123} & \begin{pmatrix} \tau_3 \cdot t + s_3 + s_{23} \\ + s_{13} + s_{123} \end{pmatrix} \end{pmatrix} \sigma^2 \\
&= \begin{pmatrix} \tau_1 \cdot t + s & \tau_1\tau_2 \cdot t + s_{12} + s_{123} & \tau_1\tau_3 \cdot t + s_{13} + s_{123} \\ \tau_1\tau_2 \cdot t + s_{12} + s_{123} & \tau_2 \cdot t + s & \tau_2\tau_3 \cdot t + s_{23} + s_{123} \\ \tau_1\tau_3 \cdot t + s_{13} + s_{123} & \tau_2\tau_3 \cdot t + s_{23} + s_{123} & \tau_3 \cdot t + s \end{pmatrix} \sigma^2.
\end{aligned}$$

The second covariance matrix required is for the random vector $\vec{S}_{1,2,3} = \left(X_{\hat{S}_1}, X_{\hat{S}_2}, X_{\hat{S}_3}\right)^{\mathrm{T}}$ and is given as

$$\boldsymbol{\Sigma}_{\vec{S}_{1,2,3}} = \begin{pmatrix} s & s_{12}+s_{123} & s_{13}+s_{123} \\ s_{12}+s_{123} & s & s_{23}+s_{123} \\ s_{13}+s_{123} & s_{23}+s_{123} & s \end{pmatrix} \sigma^2.$$

With these matrices, we are capable of computing the conditional entropy $h(\vec{X}_T \mid O_1^{(\tau_1)}, O_2^{(\tau_2)}, O_3^{(\tau_3)})$. It will be important later that the above covariance matrices only depend on pairwise spectator overlaps between the executions $(s_{13}+s_{123})$, $(s_{12}+s_{123})$, and $(s_{23}+s_{123})$, rather than individual sets s_{12}, s_{23}, s_{123}, etc.

9.2.2 M Executions

We can generalize the prior section's analysis to obtain the target's conditional entropy for an arbitrary number of evaluations. Let M be the total number of evaluations where $M \in \mathbb{Z}_{>0}$. Denote A as the set of integers from 1 to M, such that $A = \{1, \ldots, M\}$. We can generate the set of all subsets of spectators that overlap and do not overlap between evaluations using the power set of A (denoted by $\mathcal{P}(A)$). Specifically $\mathcal{S} = \mathcal{P}(A) \setminus \{\emptyset\}$, the empty set is excluded as it corresponds to the target not participating in any computation. The number of spectator subsets and target participation configurations is $|\mathcal{S}| = 2^M - 1$. The output random variable of experiment $i \in \{1, \ldots, M\}$ is therefore

$$O_i^{(\tau_i)} = \tau_i \cdot X_T + \sum_{\substack{R \subseteq \mathcal{S}: \\ i \in R}} X_{S_R} = \tau_i \cdot X_T + X_{\hat{S}_i}.$$

We can generate elements of the $M \times M$ covariance matrix of the random vector $\vec{O}_{1,\ldots,M} = (O_1^{(\tau_1)}, \ldots, O_M^{(\tau_M)})^{\mathrm{T}}$ using the following expression for $i, j \in \{1, \ldots, M\}$:

$$\mathrm{Cov}\left[O_i^{(\tau_i)}, O_j^{(\tau_j)}\right] = \begin{cases} \tau_i \tau_j \cdot \sigma_T^2 + \sum\limits_{\substack{R \subseteq \mathcal{S}: \\ (i,j) \in R}} \sigma_R^2 & \text{if } i \neq j \\ \tau_i \cdot \sigma_T^2 + \sum\limits_{\substack{R \subseteq \mathcal{S}: \\ i \in R}} \sigma_R^2 & \text{if } i = j \end{cases}.$$

Similarly, elements of the covariance matrix of the random vector $\vec{S}_{1,\ldots,M} = \left(X_{\hat{S}_1}, \ldots, X_{\hat{S}_M}\right)^{\mathrm{T}}$ can be generated as follows:

$$\mathrm{Cov}\left[X_{\hat{S}_i}, X_{\hat{S}_j}\right] = \begin{cases} \sum\limits_{\substack{R \subseteq \mathcal{S}: \\ (i,j) \in R}} \sigma_R^2 & \text{if } i \neq j \\ \sum\limits_{\substack{R \subseteq \mathcal{S}: \\ i \in R}} \sigma_R^2 & \text{if } i = j \end{cases}.$$

If the total number of spectators per evaluation is fixed to s, then $\sum_{\substack{R \subseteq \mathcal{S}: \\ i \in R}} \sigma_R^2 = \sigma^2 n$.

9.2.3 Experimental Evaluation

Unlike two executions, we can no longer graphically represent conditional entropy as a function of overlap sizes, as there are several dimensions to consider. Instead, we enumerate all possible spectator configurations for $s = 24$ and for each spectator configuration we compute the minimum of the seven conditional entropies corresponding to valid target configurations τ_1, τ_2, τ_3:

$$\min_{\tau_1, \tau_2, \tau_3} h\left(\vec{X}_T \mid O_1^{(\tau_1)}, O_2^{(\tau_2)}, O_3^{(\tau_3)}\right).$$

We then determine the maximums across all spectator configurations which correspond to the optimal choices that minimize the target's information disclosure.

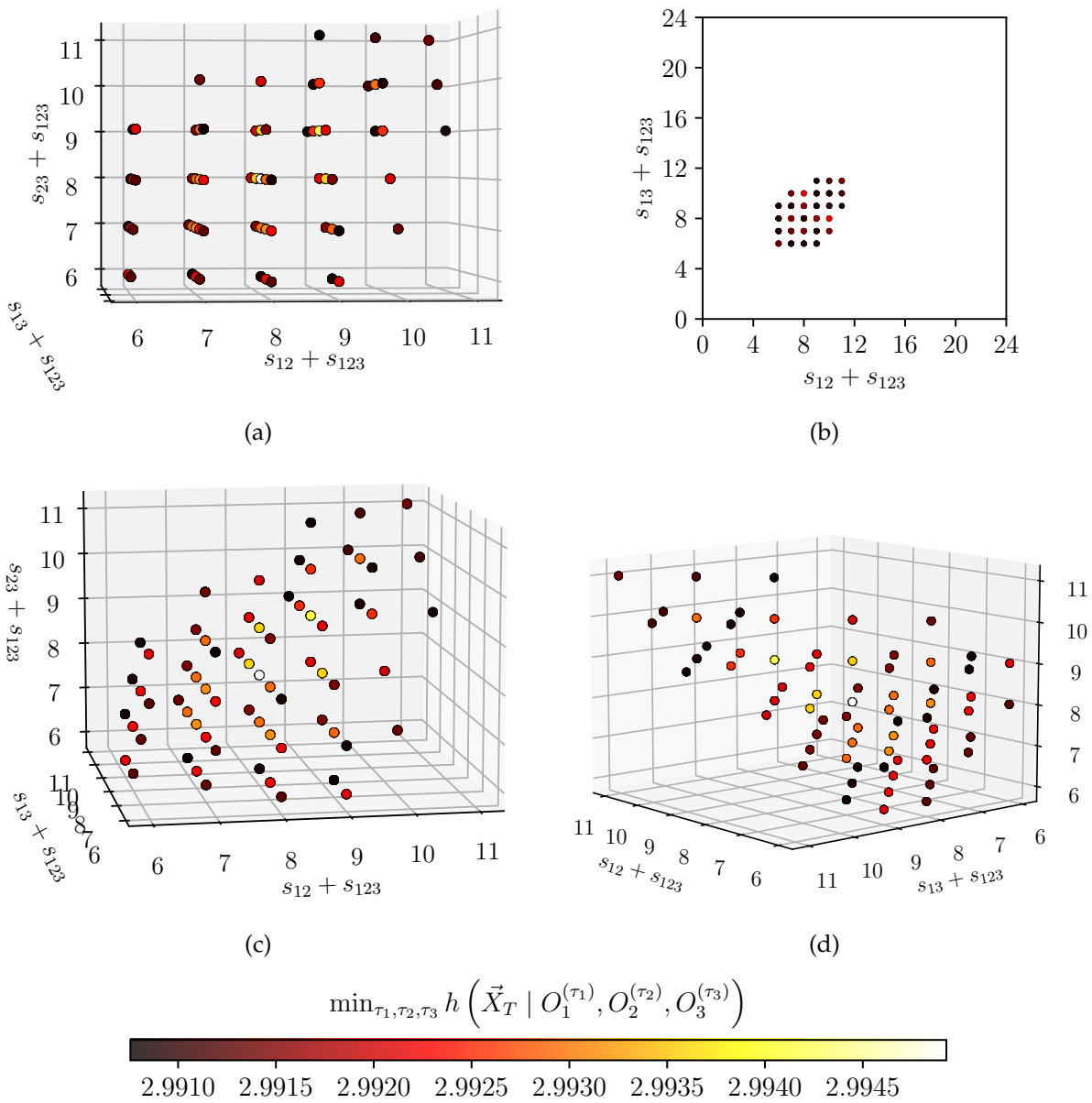


Figure 9.3: Configurations and values of minimal information disclosure as functions of the pairwise spectator overlaps for three evaluations.

We plot the top 500 spectator configurations, which yield 20 unique entropy values (displayed in the color map), in Figure 9.3 from different viewing angles. The axes correspond to pairwise overlap sizes, and a point with a fixed overlap, e.g., $s_{12} + s_{123}$, corresponds to different individual sizes of s_{12} and s_{123} that add to the same values. Recall that only the sum contributes to the entropy

computation.

The maximum conditional entropy (singular white point) occurs when the pairwise overlaps are $1/3$ of s, i.e., when $s_{13} + s_{123} = s_{12} + s_{123} = s_{23} + s_{123} = 8$. Other top configurations are located nearby, but do not deviate from the center evenly. In the projection of two of the three pairwise overlap dimensions ($s_{12} + s_{123}$ versus $s_{13} + s_{123}$, Figure 9.3b), the top-500 configurations are concentrated in the $1/3$ overlap region. The shape is preserved (and thus the figures are identical) in the other two projections. It is important to point out that the difference in entropy between the largest and smallest value plotted is less than $1/100$th of a bit.

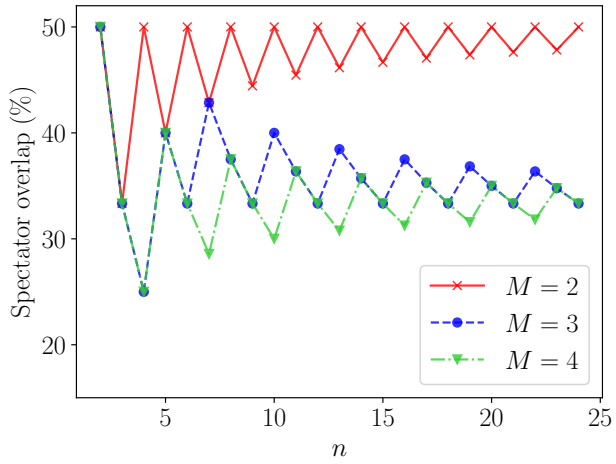


Figure 9.4: The optimal shared spectators overlap configuration relative to the total number of participants s for M evaluations.

Having examined optimal configurations for two and three executions, we want to generalize the findings to any number of experiments and spectators s. In Figure 9.4, we plot the optimal pairwise overlap percentages as a function of s for 2, 3, and 4 executions. Information leakage is always the smallest when all pairwise overlaps are equal (i.e., for $M = 3$, $s_{13} + s_{123} = s_{12} + s_{123} = s_{23} + s_{123}$). The optimal overlap percentage for $M = 2$ is upper bounded by 50% and tends towards 50% as s grows. Interestingly, the optimal overlap for both $M = 3$ and $M = 4$ trend toward

1/3 overlap, while ideal overlaps are generally smaller for $M = 4$. Analysis of large M, while potentially interesting, is of limited practical value.

9.3 Recommendations

Our analysis throughout Chapters 8 and 9 allows us to formalize the following recommendations for computation designers considering the average salary:

- The amount of information disclosure about a target is independent of adversarial inputs. It was also experimentally shown to be independent of distribution parameters for three different distributions and analytically shown for normal distribution. All examined distributions produce nearly identical entropy loss curves.

- One can reduce the amount of entropy loss to a desired level by increasing the number of participants. For example, the computation designer can advertise at most 5% or 1% maximum entropy loss for the average salary application, which will require recruiting 6 or 25, respectively, non-adversarial participants when running only a single evaluation.

- In the presence of repeated computations, information disclosure about inputs continues for both participants who stay and participants who leave. With two executions, protection is the largest with 50% overlap in the participants, while both a small overlap and an overwhelming overlap result in undesirable information disclosure about different types of participants (i.e., those who stay vs. those who leave).

- With more executions, pairwise overlap sizes determine information disclosure. For 3 and 4 executions, optimal configurations have overlap sizes near 1/3 of the number of partici-

pants.

- Information disclosure about participants' inputs can still be kept at a desirable level by enrolling enough participants and restricting the percentage of reused inputs to be in a desired range. For example, with two executions and following the guidelines of keeping the overlap near 50%, the number of non-adversarial input contributors needs to be at least 8 to meet the target of 5% information loss.

Chapter 10

Advanced Statistical Functions

10.1 Candidate Functions

In what follows, we specify the functions we analyze and divide them into two categories: order statistics and variability measures.

Order Statistics The first class of functions we analyze are categorized as the kth *order statistic* of a sample set and corresponds to the kth-smallest value. The first function we consider is the *maximum* (nth order statistic), and similarly the *minimum* (first order statistic). Given a vector $\vec{x}$ of $n > 1$ inputs, the maximum $f_{\max}$ is defined as

$$f_{\max}(\vec{x}) = \max_{i \in [1,n]} x_i.$$

Naturally, the minimum $f_{\min}(\vec{x})$ can be computed by replacing the max operator in the above equation with minimum.

Next, we consider the *median*. Given a sequence of n inputs in non-decreasing order, the me-

dian is computed as follows:

$$f_{\text{med}}(\vec{x}) = \begin{cases} x_{\frac{n+1}{2}} & \text{if } n \text{ is odd} \\ (x_{n/2} + x_{n/2+1})/2 & \text{if } n \text{ is even} \end{cases}.$$

Note, the original functionality for even numbers of inputs is fundamentally different from the odd case, where the function computes the average of the two middle values. To rectify this to ensure the odd and even cases are consistent with one another, we modify the behavior for even numbers of inputs, such that we return the smaller of the two middle values:

$$f_{\text{med}}(\vec{x}) = \begin{cases} x_{\frac{n+1}{2}} & \text{if } n \text{ is odd} \\ \min\left(x_{n/2}, x_{n/2+1}\right) & \text{if } n \text{ is even} \end{cases}.$$

Variability Measures The next class of functions we consider are *variability measures*. This consists of the *standard deviation*, an integral component of data analytics used to measure the dispersion among a set of samples. It is a natural progression from computing the average since the mean of a sample dataset is required in order to compute the standard deviation. The (sample) standard deviation f_σ of a vector of inputs $\vec{x} = (x_1, x_2, \ldots, x_n)$ is defined as

$$f_\sigma(\vec{x}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - f_\mu(\vec{x})\right)^2},$$

where $f_\mu(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the mean (average). Explicitly accounting for the square root operator in the calculation is superfluous for several reasons, since it

(a) can be trivially reversed by squaring the function's output,

(b) may unnecessarily introduce noise into the entropy estimation procedure (discussed next), and

(c) does not affect the computed entropy by virtue of the relationship $H(f(X)) \leq H(X)$, which states that the entropy of a random variable X can only decrease when passed through an arbitrary function.

Instead, we consider the squared standard deviation, i.e., the *variance*:

$$f_{\sigma^2}(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - f_{\mu}(\vec{x})\right)^2.$$

It is not uncommon for both the mean and standard deviation to be revealed simultaneously, as they are fundamental measures the centrality and dispersion. We therefore consider the scenario when both statistics are released concurrently as a tuple of values and is denoted by $f_{(\mu,\sigma^2)}$.

10.2 Entropy Estimators

For a variety of practical applications, it may be required to determine the entropy of a collection of samples sourced from an arbitrary, unknown distribution, the support of which may not be known. While previously in Chapters 8 and 9 we could derive closed-form expressions for the entropy when studying the average salary computation, we are not immediately afforded this luxury for the statistical measures we analyze here. Hence, we turn to alternative approaches for measuring information disclosure. The practice of developing *entropy estimators* to measure the entropy of an unknown distribution given a set of sample data is a rich area of study and is deeply embedded in data-driven fields including neuroscience [156, 159], signal processing [135, 28], RNA sequenc-

ing [132], and various machine learning tasks (i.e., feature selection and classification [84, 29], divergence calculations [137]).

The type of sample data (discrete, continuous, or a mixture) dictates the type of estimator that can be used. Fortunately, the *mixed entropy estimator* proposed by Gao et al. [84] is equipped to handle all possible configurations. Precisely, if a sample set consisting exclusively of discrete data is supplied, then the algorithm reduces to the *plug-in estimator*, a popular yet flexible choice for estimating the entropy of discrete samples over an arbitrary support. The estimator itself is closely related to the maximum likelihood estimator for approximating the statistical parameters of an arbitrary distribution. Conversely, if continuous samples are supplied, then the algorithm reduces to the KSG estimator [109] based on k-nearest neighbor estimates and offers high degrees of precision in the presence of a limited number of samples, provided the random variable is "low-dimensional" (lower than 6 dimensions) [85].

The true power of this estimator emerges when the samples are a *mixture* of both discrete and continuous data, such as when conducting feature selection to determine which features possess the strongest relationships with each other. In our analysis, certain functions will solely produce discrete outputs, regardless of the input distribution. An example would be $f_{\max}$ with a fixed adversary's input x_A. If all inputs are sourced from a continuous distribution and x_A is sufficiently large, then the support of O would consist solely of x_A and thus be a discrete random variable. The estimator enables us to capture the disclosure in this configuration, such that we can perform meaningful analysis.

Note, Gao et al.'s estimator is designed to compute the mutual information $I(X;Y)$ of two random variables X and Y. However, as demonstrated in Chapter 8, the mutual information between the target's input and the output of the function, given a fixed attacker's input, is exactly

equal to the absolute entropy loss, namely

$$I(X_T; O \mid \vec{X_A} = \vec{x_A}) = H(X_T) - H\left(X_T \mid O, \vec{X_A} = \vec{x_A}\right).$$

We can recover the conditional entropy of the target given the output O and a fixed attacker's input $\vec{x}_A$ exactly.

10.3 Experiments

We evaluate our candidate functions with various input distributions below. Our analysis consists of calculating the remaining entropy $H(X_T \mid O, X_A = x_A)$ for the target after the output is revealed as a function of the attacker's input, up to 10 total spectators. We additionally compute the information disclosure from the output if the attacker was not present, namely $H(X_T \mid O)$. This quantity corresponds to an external adversary who wants to extract information about the target, without having the opportunity to participate in the computation. Naturally, this quantity is a constant for a fixed number of spectators for all distributions.

Our analysis of the information disclosure for the uniform, Poisson, normal, and log-normal distributions is presented in Figures 10.1 to 10.4. In each figure, we display the disclosure from:

- the maximum function (subfigure a),
- the median function, with odd and even total participants (subfigures b and c, respectively),
- the variance function (subfigure d), and
- the simultaneous variance and mean release function (subfigure e).

For continuous distributions, we compute the information disclosure for different values of x_A in

intervals of 0.1 and apply the Savitzky-Golay filter to smooth the data.

10.3.1 Maximum

For the maximum, we see in Figures 10.1a, 10.2a, 10.3a and 10.4a, all distributions exhibit consistent behavior. We first observe that the information disclosure is maximized when an attacker supplies an input "opposite" to the function that is being evaluated. Larger inputs in the attacker's domain will lead to lower information disclosure and ultimately converge to the target's initial entropy $H(X_T)$. The intuition behind this is that if an adversary submits a "large" input, then it is highly likely that the output of the computation will be the attacker's own input, i.e., $o = x_A$. In the case of input distributions with positively infinite support, $H\left(X_T \mid O, X_A = x_A\right)$ converge to $H(X_T)$ in the limit of $x_A \to \infty$.

For all distributions, if the adversary does not participate in the computation, the quantity $H\left(X_T \mid O\right)$ forms a strict lower bound on $H\left(X_T \mid O, X_A = x_A\right)$. The implication is that the adversary can learn *at most* the same amount of information that would be learned if they were to not participate at all. The adversary's input(s) cannot be crafted in such a way that they can extract more than a fixed amount of information.

10.3.2 Median

As stated, the median is computed slightly differently based on whether we have an even or odd number of inputs. For an *odd* number of participants (Figures 10.1b, 10.2b, 10.3b and 10.4b), the information disclosure is minimized at the mean of the input distribution, with full symmetry being displayed for distributions with symmetric masses/densities (uniform and normal). Similar to the maximum, $H\left(X_T \mid O\right)$ forms a strict lower bound on $H\left(X_T \mid O, X_A = x_A\right)$ for all x_A's. The

optimal adversarial strategy for the median (odd) is generally to provide the largest possible input into the computation. Specifically, $x_A = 0$ or $x_A = n$ for uniform, $x_A = +\infty$ for Poisson, $x_A = -\infty$ or $x_A = \infty$ for normal, and x_A is near zero or $x_A = \infty$ for log-normal.

For an *even* number of participants (Figures 10.1c, 10.2c, 10.3c and 10.4c), the local maximums of $H(X_T \mid O, X_A = x_A)$ shift slightly towards the right of the mean. More interestingly, the optimal strategy shifts fully to being biased towards the upper end of the input distribution's support, where $H(X_T \mid O, X_A = x_A)$ can be smaller than $H(X_T \mid O)$ for sufficiently large x_A. Under these circumstances, the adversary is incentivized to participate in the computation.

10.3.3 Variance

The variance computation is the first instance where we observe several new phenomena, in conjunction with a break in consistency regarding the behavior of discrete and continuous input distributions.

The first major departure from earlier experiments pertains to the behavior of $H(X_T \mid O)$ for all input distributions. Contrary to our earlier experiments, the quantity $H(X_T \mid O)$ forms an *upper bound* on the information disclosure. This anomaly extends further to the case of $|S| = 1$, which leads to *less disclosure* than experiments with more spectators (in some cases, surpassing $|S| = 3$). In other functions, the information disclosure is directly proportional to the number of spectators (as $|S|$ and $H(X_T \mid O)$, thus less is revealed about the target). The implication here is that the adversary gains a substantial advantage by supplying inputs into the computation, and is thus highly incentivized to participate. This is entirely counter-intuitive relative to our prior results. We leave this as an open problem for future analysis.

Shifting our attention to the distributions themselves, we see for uniform and Poisson (Fig-

ures 10.1d and 10.2d) the information disclosure is minimized around the mean. Moving outward from the mean is more advantageous for the adversary, such that the information disclosure is maximized at the extrema ($x_A = 0$ or $x_A = n$ for uniform, $x_A = +\infty$ for Poisson). The normal and log-normal distributions partially mimic the discrete distributions regarding the local maxima of $H(X_T \mid O, X_A = x_A)$ near the mean, with disclosure increasing when moving outward. However, where we previously observed the largest amount of information disclosure at the extrema, we now observe absolute minimums between $\mu \pm \sigma$ and $\mu \pm 2\sigma$ for normal, and between $\mu - \sigma$ and $\mu - 2\sigma$ for log-normal. Furthermore, inputs near the tails of the distribution lead to higher $H(X_T \mid O, X_A = x_A)$. This behavior is intuitive since extreme values in the variance would negatively impact the result of the computation. The adversary would ultimately obfuscate the "true" result of the computation to the point where they are incapable of learning any information about the target.

The divergence between discrete and continuous continues when the mean and variance are released simultaneously. For uniform and Poisson input distributions (Figures 10.1e and 10.2e), the quantities $H(X_T \mid O, X_A = x_A)$ and $H(X_T \mid O)$ overlap each other. This directly implies that when the mean and standard deviation are revealed simultaneously, then the information disclosure is entirely independent of the attacker's input. This mimics the behavior we observed and analytically proved in Chapter 8 for the average salary. We note that for Poisson, a slight gap begins to form between $H(X_T \mid O, X_A = x_A)$ and $H(X_T \mid O)$ for larger values of x_A and $|S|$. We attribute this to an inherent accuracy limitation within the estimator.

The equivalence between $H(X_T \mid O)$ and $H(X_T \mid O, X_A = x_A)$ does not hold for continuous distributions, where we observe $H(X_T \mid O)$ form a strict lower bound on $H(X_T \mid O, X_A = x_A)$. Moreover, a gap exists between the curves for $|S| = 1$, implying is objectively superior for the

adversary to not participate in the computation. For both normal and log-normal, the optimal strategy for the adversary is to submit inputs centered around the mean.

10.3.4 Relationship between f_μ, f_{σ^2}, and $f_{(\mu,\sigma^2)}$

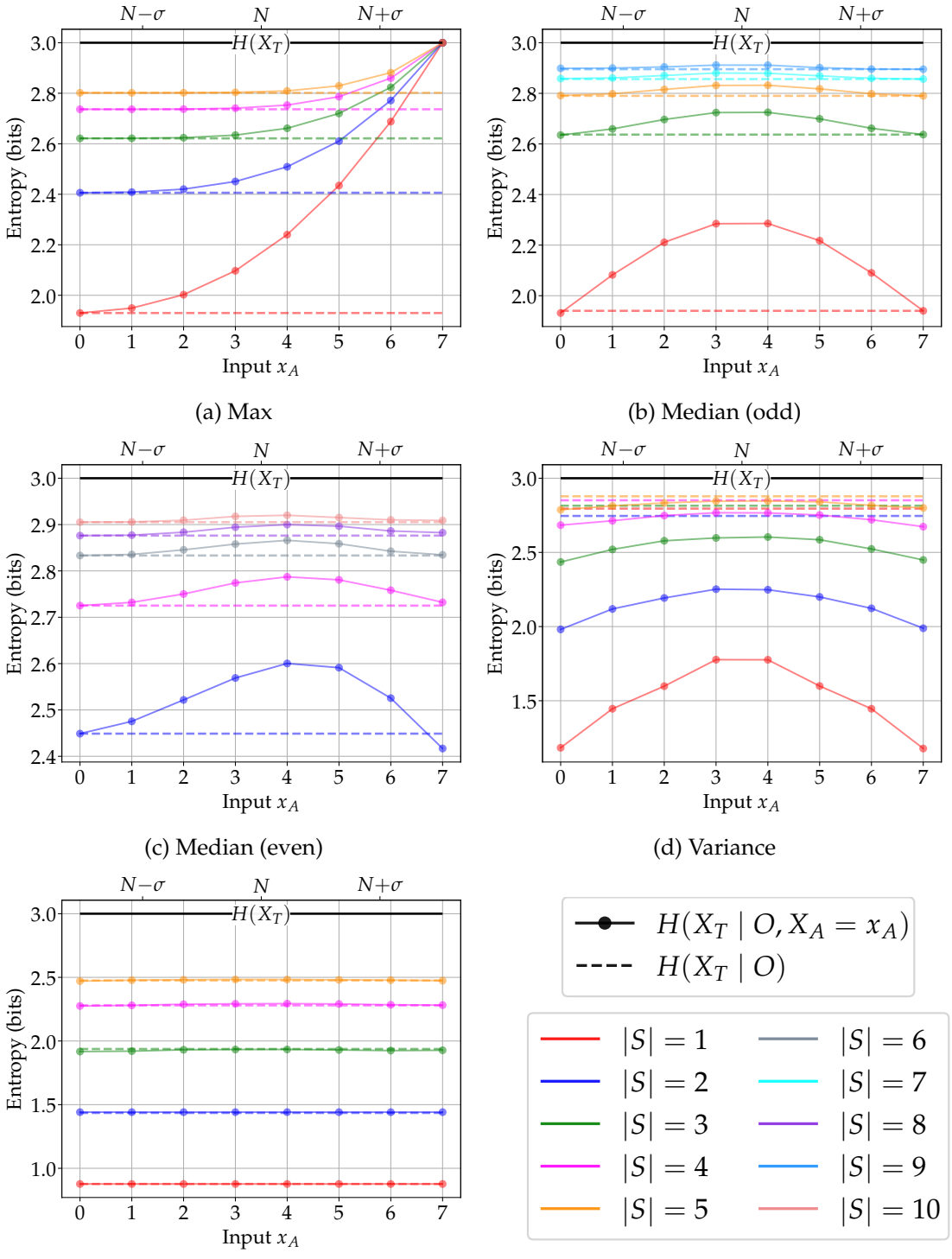
The mean and variance are deeply intertwined statistical measures. As such, we aim to quantify their relationship, relative to how much information each function discloses about an individual targeted participant. Let H_f be the absolute entropy loss for the target after participating in the evaluation of f. Our intuition is that the sum of the information disclosed about the target individually from the outputs of f_μ and f_{σ^2} is at most equivalent to the information disclosure of the simultaneous release $f_{(\mu,\sigma^2)}$. Specifically,

$$H_{f_\mu} + H_{f_{\sigma^2}} \leq H_{f_{(\mu,\sigma^2)}}.$$

Note, the sum of H_{f_μ} and $H_{f_{\sigma^2}}$ is not considered operationally significant and would not be computed directly, since if the functions were evaluated on the same sample data and the outputs were both eventually released (suppose after some period of time), then this would still correspond to the simultaneous release case $H_{f_{(\mu,\sigma^2)}}$. Nonetheless, it is of practical interest to determine whether more information is revealed about the target as a byproduct of revealing both outputs together.

In Figure 10.5, we plot the functions for our four distributions for $|S| = 2$ and $|S| = 5$. Our results demonstrate that the information disclosure of simultaneous mean and variance release is substantially higher than if the computations were performed independently. This is *counterintuitive* to what we anticipated since this implies that the simultaneous release provides are more complete picture of the target's input than the individual functions are capable of. We conjecture

that given a sufficiently accurate estimator, one can construct a function that would reveal the target's input entirely.



(a) Max

(b) Median (odd)

(c) Median (even)

(d) Variance

(e) Variance and mean release

Figure 10.1: Analysis of target's entropy loss using the uniform distribution with $\mathcal{U}(0,7)$, with $|T| = 1$. $N = \frac{a+b}{2}$ corresponds the mean of a uniform random variable.

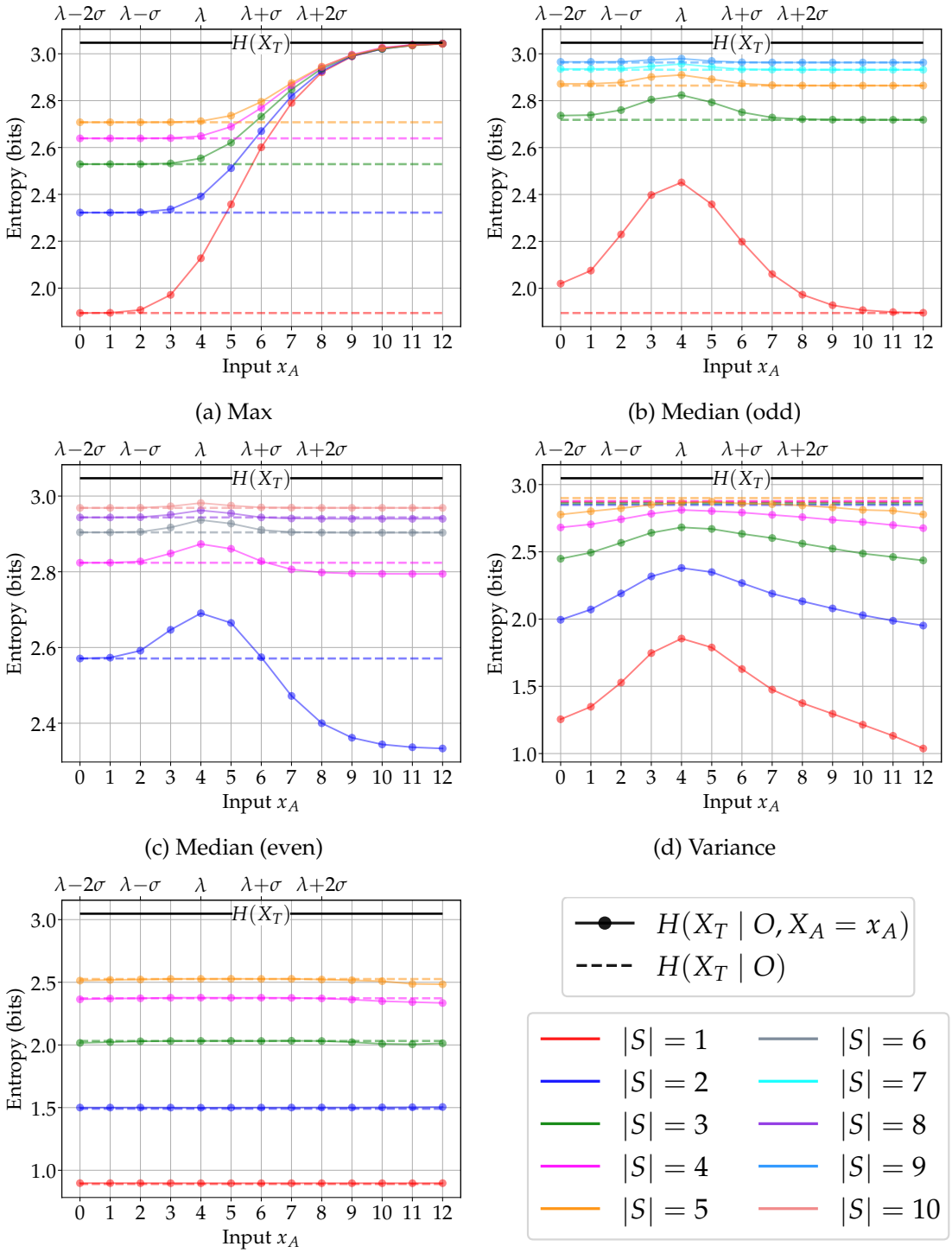
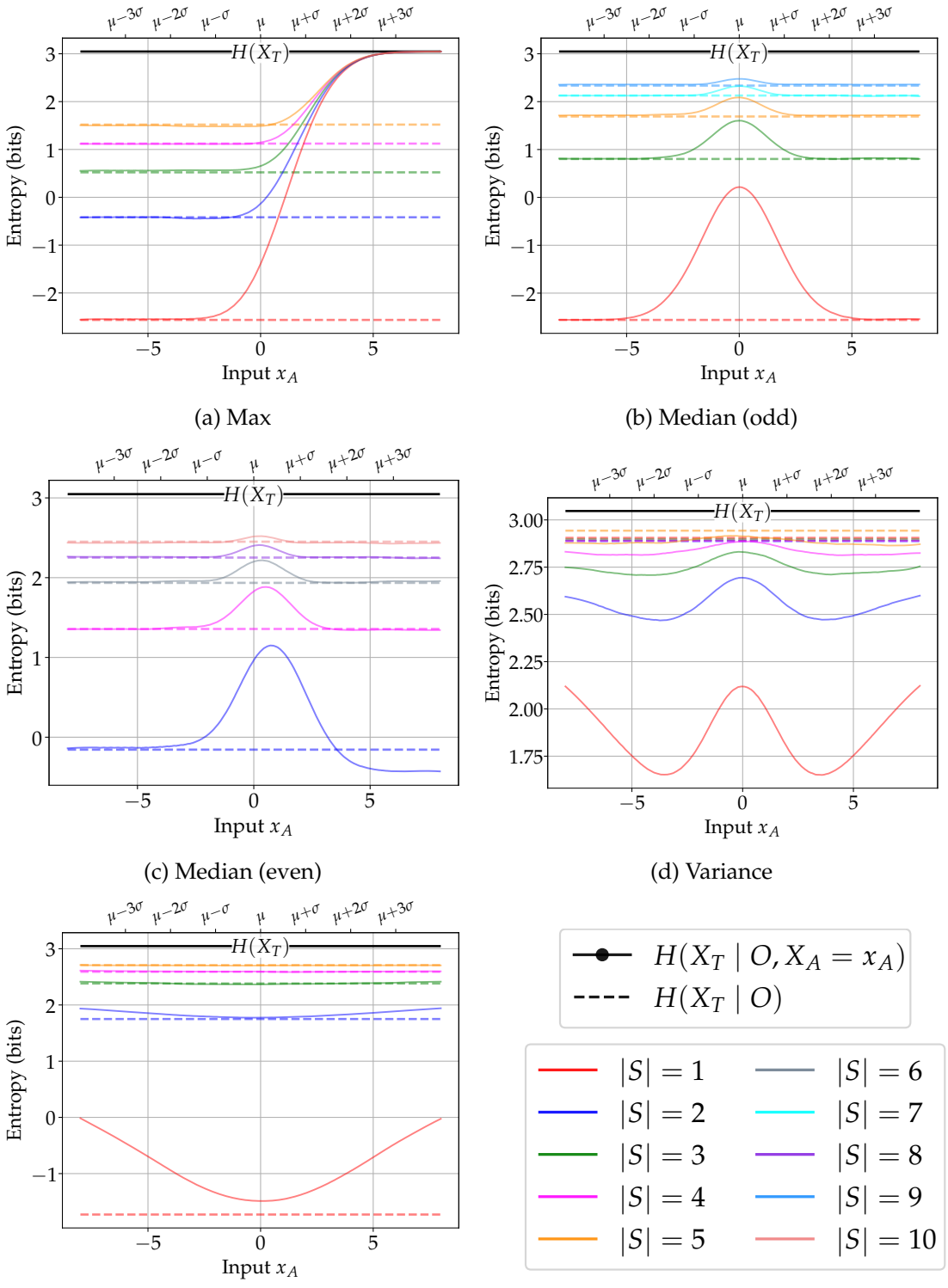


Figure 10.2: Analysis of target's entropy loss using the Poisson distribution with Pois(4), and with $|T| = 1$.



(a) Max

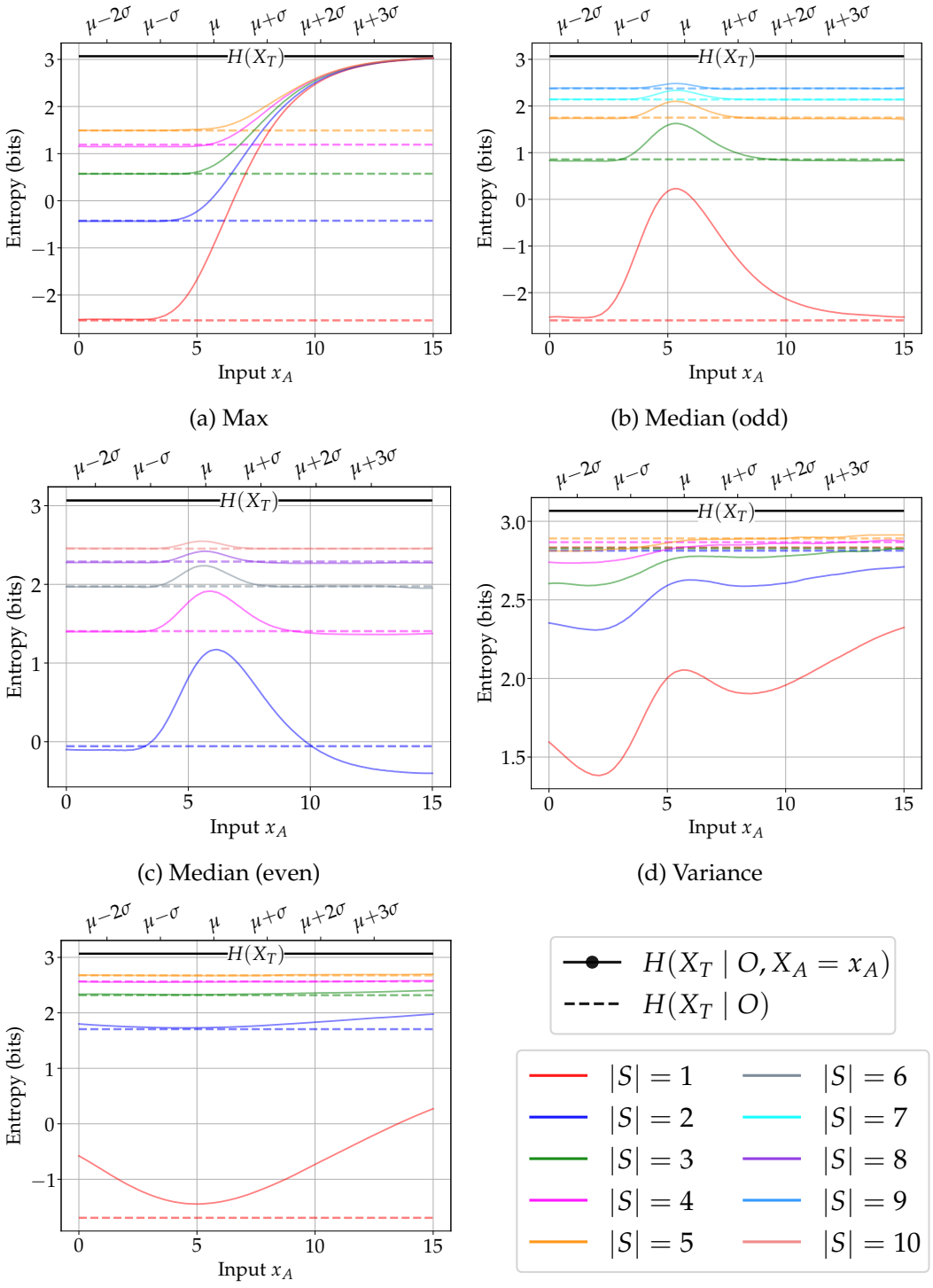
(b) Median (odd)

(c) Median (even)

(d) Variance

(e) Variance and mean release

Figure 10.3: Analysis of target's entropy loss using the Gaussian distribution with $\mathcal{N}(0, 4.0)$, and with $|T| = 1$.



(a) Max

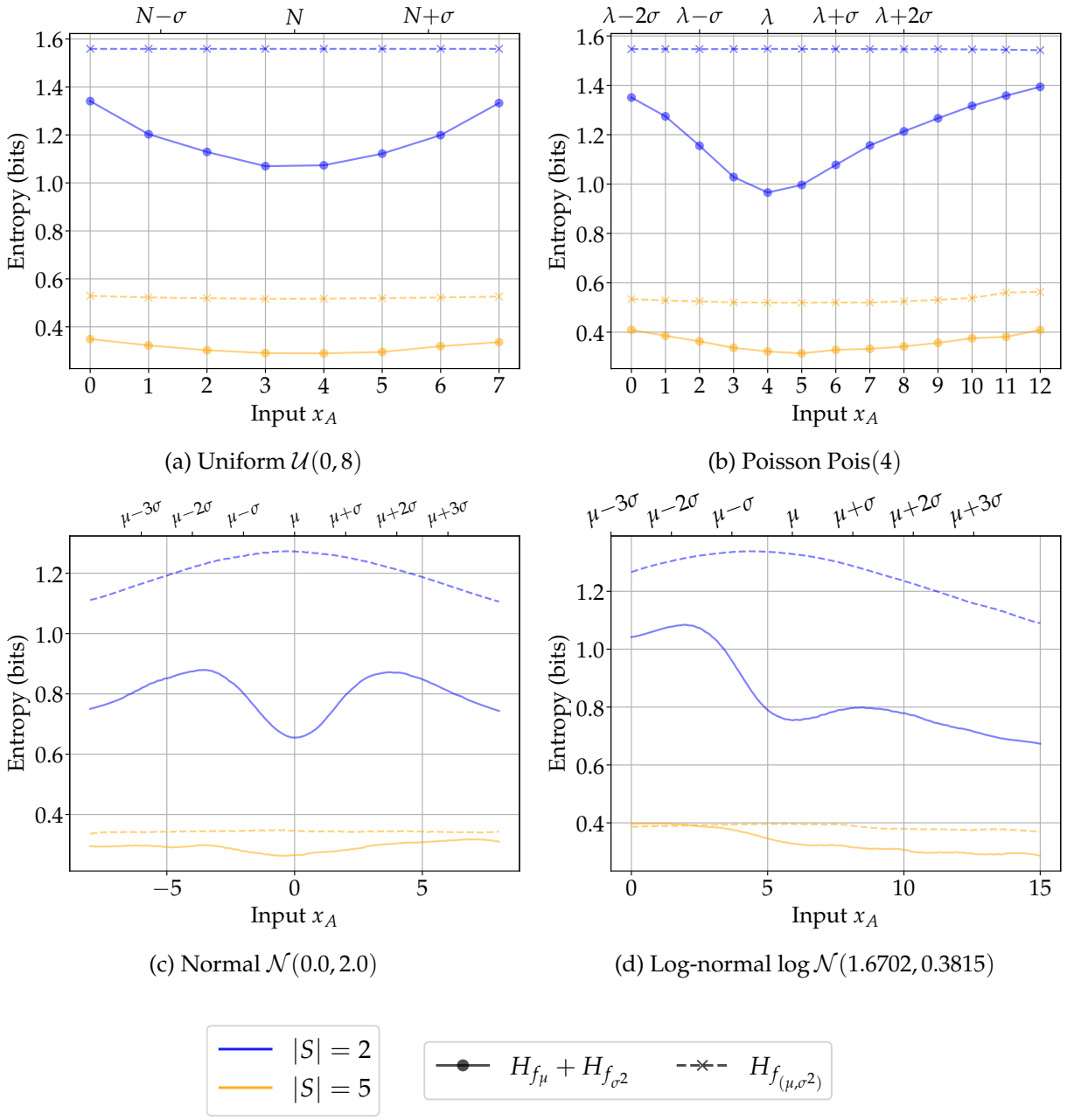
(b) Median (odd)

(c) Median (even)

(d) Variance

(e) Variance and mean release

Figure 10.4: Analysis of target's entropy loss using the log-normal distribution with $\log \mathcal{N}(1.6702, 0.145542)$, and with $|T| = 1$.



(a) Uniform $\mathcal{U}(0, 8)$

(b) Poisson Pois(4)

(c) Normal $\mathcal{N}(0.0, 2.0)$

(d) Log-normal $\log \mathcal{N}(1.6702, 0.3815)$

Figure 10.5: Absolute entropy loss comparison for various distributions of $f_\mu + f_{\sigma^2}$, and $f_{(\mu,\sigma^2)}$ for $|S| = 2$ and $|S| = 5$.

Chapter 11

Conclusions

The primary objective of this dissertation was to approach SMC from two distinct, yet intertwined perspectives. In the traditional sense, we developed a comprehensive suite of protocols that advance the state-of-the-art of the field and offer compelling advantages over prior solutions. Subsequently, we sought to address previously unasked (and thus, unanswered) broader questions about secure computation.

We began by studying multi-party threshold secret sharing over a ring in the semi-honest model with an honest majority with the goal of improving performance compared to field-based computation. We designed building block operations for n-party replicated secret sharing over any ring and consequently built upon them to enable general-purpose integer and floating-point protocols over the ring $\mathbb{Z}_{2^k}$. Our implementation results demonstrated that ring-based implementations of various operations are significantly faster than their field-based equivalents with 3, 5, and even 7 parties.

Future directions for our framework include investigating how switching to lookup tables for certain operations (such as division and floating-point rounding) can impact performance over

the protocols presented in this dissertation. Furthermore, we can extend this framework to accommodate a weaker security assumption where participants can behave maliciously.

In the second part of this dissertation, we investigated information disclosure associated with revealing the output of the average salary computation on private inputs. Using the framework of [12], we extensively analyzed the function and derived several information-theoretic properties associated with the computation. Inputs were modeled using several common discrete and continuous distributions, leading to multiple interesting conclusions about their entropy loss. We expanded the scope to include multiple executions on related inputs and determined the best configurations that minimize information disclosure. These conclusions led us to provide recommendations for future computation designers.

Our comprehensive treatment of the average salary served as the foundation to extend our analysis to encompass more advanced descriptive statistics, including maximum, median, and standard deviation. For these functions, we shifted to data-driven techniques to compute the entropy and uncovered a number of interesting phenomena. In future works, we aim to solidify our observations by deriving closed-form expressions of the leakage. Additionally, we intend to investigate different mitigation strategies when the computational setup cannot be modified.

Appendix A

Additional Protocols

This section contains any additional protocols that are not central to our RSS framework, but useful for the sake of comparison against our proposed solutions.

A.1 Sparse Multiplication

Given inputs $[a]$ and $[\hat{b}]$ where shares of $\hat{b}$ are sparsely distributed (as defined in Section 4.2.1), computing the product $[a] \cdot [\hat{b}]$ can be performed using an optimized multiplication algorithm. The logic follows closely to the original specification in Protocol 1 with the exception of the t parties without access to the sparse share $\hat{T}$ (i.e., parties $p \in \hat{T}$) skipping the resharing component of multiplication since their locally computed product would be zero. These parties only receive shares from other participants, such that we obtain a final communication complexity of $t(t+1)$ ring elements (or $tk(t+1)$ bits in $\mathbb{Z}_{2^k}$) communicated by all parties.

Theorem 4

MulSparse is secure according to Definition 1 in the (n, t) setting with $n = 2t + 1$ in the presence of secure communication channels, and assuming PRG is a pseudo-random generator and that $\hat{b}$ is sparsely shared.

Proof. The security proof of MulSparse is an extension of that of ordinary multiplication (Theorem 1), with the modification that $t + 1$ parties who are entitled access to $\hat{b}_{\hat{T}}$ have exact knowledge of the secret input $\hat{b}$. This is acceptable since this secret is merely one share of the "true" secret, i.e., the input to B2A, and thus does not violate Definition 1. Therefore, we demonstrate that MulSparse does not disclose any information about the private input a and that parties $p \notin \hat{T}$ do not learn any information about $\hat{b}$.

The simulator S_I is constructed equivalently as for Theorem 1: S_I sends a random element of $\mathcal{R}$ to a party (or parties) $p \in I$ in line 14, or S_I receives a computed value from an honest party p on behalf of p' in line 16.

To argue that the simulated view is computationally indistinguishable from the real view, we distinguish between corrupt parties with access to $\hat{b}_{\hat{T}}$ (i.e., $p \in \hat{T}$), and those without access to $\hat{b}_{\hat{T}}$ (i.e., $p \notin \hat{T}$). In the former case, security follows directly from Theorem 1 since the corrupt parties in I will collectively hold a_T, and keys k_T for each $T \in \mathcal{T}$, as well as $\hat{b}_{\hat{T}}$. Together, they can compute the shares c_T, but the remaining share c_{T^*} (where $T^* = I$) is unknown to them. Any share(s) received by another party $p' \notin I$ were masked by a fresh pseudo-random element from G_{T^*} and is, therefore, pseudo-random and indistinguishable from random by any $p \in I$. Similarly, if parties $p \in \hat{T} = I$ are corrupted, indistinguishability is trivially maintained since the values received were protected by pseudo-random element(s) generated PRGs seeded with keys they do

not have access to.

For $t > 1$, we may have a *combination* of the aforementioned configurations, where $p \in \hat{T}$ and $p \notin \hat{T}$ constitute the coalition I. Since our setting assumes that participants are semi-honest, any communicated values between parties in I are information-theoretically protected by pseudo-random element(s) that the recipient does not have access to, and are thus indistinguishable in the real and simulated views. □

Protocol 25: $[c] \leftarrow \mathsf{MulSparse}([a], [\hat{b}])$

```
// define G_T = PRG(k_T)
// pre-distributed values are [k] and public maps ρ and χ
// shares of [b̂] are sparse, one share T̂ is marked as ''special''
```

for $p \in [1, n]$ **in parallel do**
  let $S_p = \{T \in \mathcal{T} \mid p \notin T\}$
  if $p \notin \hat{T}$ **then** $v^{(p)}_{\chi(p)} = \sum_{T \in \mathcal{T}\colon \rho(T, \hat{T}) = p} a_T \hat{b}_{\hat{T}}$
  else $v^{(p)}_{\chi(p)} = 0$
  for $T \in S_p$ **do** $c_T = 0$
  for $p' \in [1, n]$ **subject to** $p' \notin \hat{T}$ **in order do**
    for $T \in S_p$ **do**
      if $(p' \neq p) \wedge (p' \notin T) \wedge (\chi(p') \neq T)$ **then**
        $c_T = c_T + \mathsf{G}_T.\mathsf{next}$
      else if $(p' = p) \wedge (\chi(p) \neq T)$ **then**
        $z = \mathsf{G}_T.\mathsf{next}$
        $c_T = c_T + z$
        $v^{(p)}_{\chi(p)} = v^{(p)}_{\chi(p)} - z$
  if $p \notin \hat{T}$ **then** send $v^{(p)}_{\chi(p)}$ to each $p' \notin \chi(p)$ (other than itself)
  for $p' \in [1, n]$ **subject to** $(p \notin \chi(p')) \wedge (p' \notin \hat{T})$ **do**
    receive $v^{(p')}_{\chi(p')}$ from p', set $c_{\chi(p')} = c_{\chi(p')} + v^{(p')}_{\chi(p')}$
  if $p \notin \hat{T}$ **then** $c_{\chi(p)} = c_{\chi(p)} + v^{(p)}_{\chi(p)}$
return $[c]$

A.2 edaBit Generation for RNTE

Our round-nearest, ties-to-even (RNTE) implementation (Protocol 19) requires a modified version of our edaTrunc algorithm (Protocol 12). In addition to generating the related random values r and $\hat{r}$ (with known bit decompositions), it also produces the random value $\hat{\hat{r}} = \sum_{i=m-2}^{k-1} 2^i r_i$ for the shorter truncation by $m-2$ bits. The logic is identical to the original edaTrunc specification and carries a slightly higher communication cost from $t+1$ parties inputting $\hat{\hat{r}}$ and the t additional carry bits $[\mathrm{cr}_{\delta,m-2}]$ captured by BitAdd. Specifically, the protocol communicates a total of $nt^2k(\log(k)+1)+(3t+1)ntk(t^2-t+1)+4tk(t+1)$ bits across all parties. The round complexity is unchanged.

Protocol 26: $([r], [\hat{r}], [\hat{\hat{r}}][b_{k-1}]) \leftarrow \mathsf{edaTruncRNTE}(k, m)$

for $p = 1, \ldots, t+1$ **in parallel do**
  party p samples $r_0^{(p)}, \ldots, r_{k-1}^{(p)} \in \mathbb{Z}_2$ and computes $r^{(p)} = \sum_{j=0}^{k-1} r_j^{(p)} 2^j$, $\hat{r}^{(p)} = \sum_{j=m}^{k-1} r_j^{(p)} 2^j$, and $\hat{\hat{r}}^{(p)} = \sum_{j=m-2}^{k-1} r_j^{(p)} 2^j$
  simultaneously execute $[r^{(p)}] \leftarrow \mathsf{Input}_k(r^{(p)})$, $[\hat{r}^{(p)}] \leftarrow \mathsf{Input}_k(\hat{r}^{(p)})$, $[\hat{\hat{r}}^{(p)}] \leftarrow \mathsf{Input}_k(\hat{\hat{r}}^{(p)})$, and $[r_i^{(p)}]_1 \leftarrow \mathsf{Input}_1(r_i^{(p)})$ for $i = 0, \ldots, k-1$, with p being the input owner
$[r] = \sum_{p=1}^{t+1} [r^{(p)}]$, $[\hat{r}] = \sum_{p=1}^{t+1} [\hat{r}^{(p)}]$, $[\hat{\hat{r}}] = \sum_{p=1}^{t+1} [\hat{\hat{r}}^{(p)}]$
$s = t+1$
for $i = 1, \ldots, \lceil \log(t+1) \rceil$ **do**
  for $j = 1, \ldots, \lfloor s/2 \rfloor$ **in parallel do**
    $\delta = j + s \cdot (i-1)$
    $\left\langle \left[r_0^{(j)}\right]_1, \ldots, \left[r_{k-1}^{(j)}\right]_1 \right\rangle, [\mathrm{cr}_{\delta,m-2}]_1, [\mathrm{cr}_{\delta,m}]_1, [\mathrm{cr}_{\delta,k-1}]_1$
      $\leftarrow \mathsf{BitAdd}\left(\left\langle \left[r_0^{(2j-1)}\right]_1, \ldots, \left[r_{k-1}^{(2j-1)}\right]_1 \right\rangle, \left\langle \left[r_0^{(2j)}\right]_1, \ldots, [r_{k-1}^{(2j)}]_1 \right\rangle \right)$
    if $s \bmod 2 = 0$ **then** $s = s/2$
    else
      $\left\langle \left[r_0^{(\frac{s+1}{2})}\right]_1, \ldots, \left[r_{k-1}^{(\frac{s+1}{2})}\right]_1 \right\rangle = \left\langle \left[r_0^{(s)}\right]_1, \ldots, \left[r_{k-1}^{(s)}\right]_1 \right\rangle$
      $s = (s+1)/2$
$[b_0]_1, \ldots, [b_{k-1}]_1 = \left[r_0^{(1)}\right]_1, \ldots, \left[r_{k-1}^{(1)}\right]_1$
$[b_{k-1}], ([\mathrm{cr}_{\delta,m-2}], [\mathrm{cr}_{\delta,m}], [\mathrm{cr}_{\delta,k-1}])_{\delta=1}^{t} \leftarrow \mathsf{B2A}([b_{k-1}]_1, ([\mathrm{cr}_{\delta,m-2}]_1, [\mathrm{cr}_{\delta,m}]_1, [\mathrm{cr}_{\delta,k-1}]_1)_{\delta=1}^{t})$
$[\hat{r}] = [\hat{r}] - [b_{k-1}] \cdot 2^{k-m-1} + \sum_{\delta=1}^{t} ([\mathrm{cr}_{\delta,m}] - [\mathrm{cr}_{\delta,k-1}] 2^{k-m-1})$
$[\hat{\hat{r}}] = [\hat{\hat{r}}] - [b_{k-1}] \cdot 2^{k-m-3} + \sum_{\delta=1}^{t} ([\mathrm{cr}_{\delta,m-2}] - [\mathrm{cr}_{\delta,k-1}] 2^{k-m-3})$
return $([r], [\hat{r}], [\hat{\hat{r}}], [b_{k-1}], [b_0]_1, \ldots, [b_{k-1}]_1)$

Appendix B

Neural Network Applications

Privacy-preserving machine learning (PPML) has become the *de facto* benchmark for secure multi-party frameworks, specifically Neural network (NN) inference. We briefly summarize the theoretical foundations of NNs and describe a mechanism for improving the efficiency of secure NN inference.

A *neural network* is a series of interconnected layers consisting of neurons. Each neuron has an associated weight and bias used for computation on some input data and outputs a prediction based on that data. A NN network layer takes the form $\mathbf{y} = g(\mathbf{x}\mathbf{W} + \mathbf{b})$, where $\mathbf{x}$ is the input vector from the previous layer, $\mathbf{W}$ is the weight tensor, $\mathbf{b}$ is the bias vector, and g is some activation function. Sample activation functions are Rectified Linear Unit (ReLU), which on input $\mathbf{x} = (x_1, \ldots, x_N)$ computes $\mathbf{y} = (y_1, \ldots, y_N)$ where each $y_i = \max(0, x_i)$, and its variant ReLU6 which computes $y_i = \min(\max(0, x_i), 6)$.

B.1 Related Works

We distinguish between two-party solutions, where one party holds the model and the other holds the input on which the model is to be evaluated, and between multi-party (typically, three-party) solutions. Publications from the first category include MiniONN [119] and Gazelle [100], both of which studied NN evaluation using SS, homomorphic encryption (HE), and garbled circuits (GC).

Multi-party constructions provide protocols for training and inference across multiple parties. ABY3 [130] combines techniques based on replicated and binary SS with GCs in the three-party setting with an honest majority. SecureNN [160] provides three-party protocols for a variety of NN functions under the same security assumption as ABY3. Their protocols are asymmetric, where parties have dedicated roles in a computation. This work is improved upon with FALCON [161] by adding malicious security with an honest majority and combining the techniques from SecureNN and ABY3.

ASTRA [49] is a three-party framework that uses SS over the ring $\mathbb{Z}_{2^k}$ under both semi-honest and malicious security assumptions. Similar to SecureNN, protocols are asymmetric. Abspoel et al. [11] apply the MP-SPDZ [102] framework for secure outsourced training of decision trees. Their system operates under the three-party, honest-majority assumption with RSS. Dalskov et al. [60] were the first to address quantized NN inference using secure multi-party computation. Their system is built into MP-SPDZ and benchmarked on the MobileNets [92] network architecture. Keller et al. [104] conducts quantization-based training and inference with three parties and one semi-honest corruption.

B.2 Quantized Neural Networks

To improve the efficiency of NN inference, it is common to employ quantization, which makes the resulting models suitable for deployment in constrained environments and is a well-studied field (see, e.g., [87]). We outline the standard quantization approach from [99] and its privacy-preserving realization from [60] for quantized TFLite models and consequently describe our optimizations.

For a vector $\mathbf{x}$, each real-valued x_i is represented as $x_i = m(\bar{x}_i - z)$, where $m \in \mathbb{R}$ is the scale and z and $\bar{x}_i$ are 8-bit integers with z being the zero point. Given an input column vector $\mathbf{x} = (x_1, \ldots, x_N)$ and a row vector $\mathbf{w} = (w_1, \ldots, w_N)$ of $\mathbf{W}$ with quantization parameters (m_1, z_1) and (m_2, z_2), respectively, the dot product of $\mathbf{x}$ and $\mathbf{w}$, $y = \sum_{i=1}^{N} x_i w_i$, is specified with quantization parameters (m_3, z_3). Since $y \approx m_3 \cdot (\bar{y} - z_3)$, $x_i \approx m_1 \cdot (\bar{x}_i - z_1)$, and $w_i \approx m_2 \cdot (\bar{w}_i - z_2)$, quantized $\bar{y}$ is computed as

$$\bar{y} \approx z_3 + \frac{m_1 m_2}{m_3} \cdot \sum_{i=1}^{N} (\bar{x}_i + z_1) \cdot (\bar{w}_i - z_2) = z_3 + m \cdot s.$$

Computing s requires integer-only arithmetic and is guaranteed to fit in $16 + \log N$ bits. The scale $m = m_1 m_2 / m_3$ is a small real number. It can be written as $m = 2^{-e} m'$ with normalized $m' \in [0.5, 1)$ which informs the value of e and represented as a 32-bit integer m'', where $m' \approx 2^{-31} m''$.

Two-dimensional convolutions typically add a quantized bias $\bar{b}$ once the dot product is computed. This is handled by setting the scale of the bias to $m_1 m_2$ and the zero-point to 0, such that the bias can be added to s prior to scaling. The last step of a convolution layer is to apply an activation function such as ReLU6. In a quantized NN, this functions as a clamping operation that eliminates values outside of range $[0, 255]$ and uses $m_3 = 6/255$ and $z_3 = 0$. This guarantees correct range

while maximizing precision with 8-bit quantized values. Going forward, m_3 becomes m_1 for the next layer and thus all intermediate layers share the same $m_1 = m_3 = 6/255$. Other activation functions such as sigmoid would be handled differently, but we only consider clamping-based functions like [60].

Computing the convolution layer securely requires the model owner to enter private quantization parameters into the computation, including all zero points z_i, modified scale m'', and integer scale adjustment 2^{M-e-31}, where M is an upper bound set to 63. After privately computing the dot product $[s]$ and adding the bias vector $[\bar{b}]$, the result is multiplied by $[m'']$ and must be truncated by a private amount $31 + e$. The truncation is accomplished by multiplying the scaled dot product by $\left[2^{M-n-31}\right]$ and $[m{\cdot}s]$ and consequently truncating by M bits. Lastly, after adding $[z_3]$ locally, clamping the result to the interval $[0, 255]$ is performed using two comparisons.

A limitation of [60]'s approach is it requires large scaling factors and consequently a large ring size of $k = 72$ for working with real numbers, using M-bit truncation with $M = 63$. We propose a modified approach where scales are folded into other aspects of the layer computation and conduct smaller truncation at the end of each layer, which guarantees a compact representation of intermediate results.

Let superscript $\langle i \rangle$ denote the layer number. Starting from layer 0, the entire layer computation (dot product, scaling, and clamping) can be interpreted as computing $0 \leq \bar{y}^{\langle 0 \rangle} \leq 255$, where

$$\bar{y}^{\langle 0 \rangle} = \frac{m_1^{\langle 0 \rangle} m_2^{\langle 0 \rangle}}{m_3^{\langle 0 \rangle}} \left(\left(\sum_{i=1}^{N} \left(\bar{x}_i^{\langle 0 \rangle} - z_1^{\langle 0 \rangle} \right) \cdot \left(\bar{w}_i^{\langle 0 \rangle} - z_2^{\langle 0 \rangle} \right) \right) + \bar{b}^{\langle 0 \rangle} \right)$$

and $z_3^{\langle i \rangle}$ was set to 0, as prescribed by the clamping operation, for all layers except the last one.

Note, since $m_3^{\langle 0 \rangle} = 6/255$, we scale the equation to redefine $\bar{y}^{\langle 0 \rangle}$ as

$$\bar{y}^{\langle 0 \rangle} = \sum_{i=1}^{N} \left(\bar{x}_i^{\langle 0 \rangle} - z_1^{\langle 0 \rangle} \right) \cdot \left(\bar{w}_i^{\langle 0 \rangle} - z_2^{\langle 0 \rangle} \right) + \bar{b}^{\langle 0 \rangle},$$

where $0 \leq \bar{y}^{\langle 0 \rangle} \leq 6/m_1^{\langle 0 \rangle} m_2^{\langle 0 \rangle}$. Now, our clamping operation can use these bounds, with the upper bound being privately entered by the model owner to avoid division. As before, the output of this layer becomes the input for the subsequent layer, i.e., $\bar{x}^{\langle i \rangle} = \bar{y}^{\langle i-1 \rangle}$. Our modified incoming vector, denoted $\hat{x}^{\langle 1 \rangle}$, is coupled with an additional scaling factor of $(255 m_1^{\langle 0 \rangle} m_2^{\langle 0 \rangle})/6$, such that

$$\bar{x}^{\langle 1 \rangle} = \frac{255 m_1^{\langle 0 \rangle} m_2^{\langle 0 \rangle}}{6} \hat{x}^{\langle 1 \rangle} = \delta^{\langle 1 \rangle} \hat{x}^{\langle 1 \rangle}.$$

Using $\bar{x}^{\langle 1 \rangle} = \delta^{\langle 1 \rangle} \hat{x}^{\langle 1 \rangle}$ gives us

$$\bar{y}^{\langle 1 \rangle} = \left(\sum_{i=1}^{N} \left(\hat{x}_i^{\langle 1 \rangle} - z_1^{\langle 1 \rangle} / \delta^{\langle 1 \rangle} \right) \cdot \left(\bar{w}_i^{\langle 1 \rangle} - z_2^{\langle 1 \rangle} \right) \right) + \bar{b}^{\langle 1 \rangle} / \delta^{\langle 1 \rangle}$$

with $0 \leq \bar{y}^{\langle 1 \rangle} \leq 6/\left(\delta^{\langle 1 \rangle} m_1^{\langle 1 \rangle} m_2^{\langle 1 \rangle} \right)$. This expression can be evaluated securely without needing fixed-point multiplication or large truncation, and all bounds are computed by the model owner prior to privately entering them in the computation.

Evaluating subsequent layers in this fashion causes the outputs to grow by a factor $\delta^{\langle i+1 \rangle}$, which can be computed as

$$\delta^{\langle i+1 \rangle} = \frac{255 m_1^{\langle i \rangle} m_2^{\langle i \rangle}}{6} \cdot \delta^{\langle i \rangle}$$

with $\delta^{\langle 0 \rangle} = 1$. However, we can ensure values remain small by truncating the output $\bar{y}^{\langle i+1 \rangle}$ by $\ell^{\langle i \rangle}$

bits. With the right choice of $\ell^{\langle i \rangle}$ we are able to maintain the necessary accuracy, and the value of $\delta^{\langle i+1 \rangle}$ consequently becomes

$$\delta^{\langle i+1 \rangle} = \delta^{\langle i \rangle} . \frac{255 m_1^{\langle i \rangle} m_2^{\langle i \rangle}}{6 \cdot 2^{\ell^{\langle i \rangle}}}.$$

The maximum number of bits we can truncate in a layer must comply with the constraint

$$\delta^{\langle i \rangle} \cdot \frac{255 m_1^{\langle i \rangle} m_2^{\langle i \rangle}}{6 \cdot 2^{\ell^{\langle i \rangle}}} \geq 1,$$

which leads to

$$\ell^{\langle i \rangle} \leq \left\lfloor \log_2 \left(255 \delta^{\langle i \rangle} m_1^{\langle i \rangle} m_2^{\langle i \rangle} / 6 \right) \right\rfloor$$

Once again, these values are *independent* of the input data and become a part of the model. We thus can use TruncPriv outlined in Section 4.2 for truncation by a private amount. The net result is that we can use a significantly smaller bound M and consequently substantially shorter ring size k. In practice, the coefficients introduced in our methodology can reasonably be folded into the scaling factors m themselves.

Other layers such as average pooling can be approximated by substituting the division by some integer d with truncation by $\lfloor \log d \rfloor$ bits, and softmax can be replaced with argmax when computing the final prediction. These changes can slightly impact the scaling factors but have no impact on the accuracy since we leverage basic algebraic properties, without changing the fundamental calculation itself.

		Ours				MP-SPDZ $\mathbb{Z}_{2^k}$, [60]			
	α	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
ρ	128	3.19	6.47	9.92	13.3	3.19	6.26	9.88	14.0
	160	4.94	10.0	15.1	20.7	4.15	8.17	13.6	19.3
	192	7.17	14.3	22.0	29.7	5.00	11.0	17.8	26.7
	224	9.71	19.9	30.0	40.9	6.57	14.1	23.1	34.9

Table B.1: Performance of 3PC quantized MobileNets prediction in seconds. MP-SPDZ results are over a ring $\mathbb{Z}_{2^k}$.

		Ours				MP-SPDZ $\mathbb{F}_p$, [60]			
	α	0.25	0.5	0.75	1.0	0.25	0.5	0.75	1.0
ρ	128	20.4	41.3	63.2	85.5	442	688	992	1343
	160	30.1	62.3	113	132	904	1414	2031	2765
	192	45.6	93.4	141	194	1398	2182	3156	4269
	224	59.2	123	183	263	1919	3005	4324	5877

Table B.2: Performance of 5PC quantized MobileNets prediction in seconds. MP-SPDZ results are over a field $\mathbb{F}_p$.

B.3 Experimental Results

Benchmarks for quantized NNs were based on the MobileNets [92] architecture, which consists of 28 layers and 1000 output classes. The network alternates between 3×3 depthwise convolutions and 1×1 pointwise convolutions. A resolution multiplier ρ (128–224) scales the dimensions of the input image, and a width multiplier α (0.25–1.0) scales the size of the input and output channels. The models we used are hosted on TensorFlow's online repository [5] and are trained on the ImageNet [66] dataset. We experimentally determined that an upper bound of $M = 16$ is sufficient for truncation by a private value since all computed $\ell^{\langle i \rangle}$s are ≤ 9 for all model configurations.

The performance of quantized MobileNets inference is presented in Tables B.1 and B.2 with 3 and 5 parties, respectively. Our methodology from Section B.2 allowed us to reduce the ring size from $k = 72$ to $k = 30$ or less, potentially reducing the time by a factor of 2. For an accurate

comparison, we executed [60]'s implementation on our machines using the same setting. Since a 5-party honest-majority ring implementation is not available in [60], or more generally in MP-SPDZ, we use a field-based implementation for the 5-party case from MP-SPDZ. Recall that the ability to generalize ring-based honest-majority protocols to more participants is our main objective.

The results our 3-party solution achieves are comparable to those in [60] despite ring reduction and can be explained by the differences in the algorithms. That is, Escudero et al. [80] experimentally determined that [60]'s implementation with ABY3's local conversion was superior to edaBits (which we use) only in one setting that we use (semi-honest, honest majority setting over $\mathbb{Z}_{2^k}$). In addition, MP-SPDZ's optimization for large computation also aids its efficiency. This demonstrates that our quantized NN solution can aid efficiency. Furthermore, our gain in the 5-party case is significant, leading to the reduction in time by a factor of 16–32.

Bibliography

[1] Inpher. https://inpher.io/.

[2] Nth party. https://www.nthparty.com/.

[3] OpenSSL – Cryptography and SSL/TLS toolkit. https://www.openssl.org/. Version: 1.1.1.

[4] Partisia. https://partisia.com/.

[5] TensorFlow repository. https://tensorflow.org/lite/guide/hosted_models. Last accessed: 6/14/22.

[6] IEEE Standard for Floating-Point Arithmetic. Technical Report 10.1109/IEEESTD.2019.8766229, IEEE, 2019.

[7] Replicated secret sharing over a ring. https://github.com/abaccarini/RSS_ring_ppml, 2022. Commit: d921581401301c35660e15aaf329f41436699389.

[8] CrypTFlow: An end-to-end system for secure TensorFlow inference. https://github.com/mpc-msri/EzPC, 2023. Commit: 8b07f73e187c5eb6ab98b0bf09b9bd276cd43949.

[9] Multi-protocol SPDZ (MP-SPDZ). https://github.com/data61/MP-SPDZ, 2023. Commit: 5b50ff21a3bd36c072ace5e3d5a49b5155f088db.

[10] M. Abspoel, A. Dalskov, D. Escudero, and A. Nof. An efficient passive-to-active compiler for honest-majority MPC over rings. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 122–152, 2021.

[11] M. Abspoel, D. Escudero, and N. Volgushev. Secure training of decision trees with continuous attributes. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2021(1):167–187, 2021.

[12] P. Ah-Fat and M. Huth. Secure multi-party computation: Information flow of outputs and game theory. In *International Conference on Principles of Security and Trust*, pages 71–92, 2017.

[13] P. Ah-Fat and M. Huth. Optimal accuracy-privacy trade-off for secure computations. *IEEE Transactions on Information Theory*, 65(5):3165–3182, 2019.

[14] P. Ah-Fat and M. Huth. Protecting private inputs: Bounded distortion guarantees with randomised approximations. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2020(3):284–303, 2020.

[15] P. Ah-Fat and M. Huth. Two and three-party digital goods auctions: Scalable privacy analysis. arXiv preprint arXiv:2009.09524, 2020.

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

[17] M. Alvim, A. Scedrov, and F. Schneider. When not all bits are equal: Worth-based information flow. In *Principles of Security and Trust (POST)*, pages 120–139, 2014.

[18] M. S. Alvim, K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith. Additive and multiplicative notions of leakage, and their capacities. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 308–322, 2014.

[19] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 265–279, 2012.

[20] T. Araki, A. Barak, J. Furukawa, M. Keller, K. Ohara, and H. Tsuchida. How to choose suitable secure multiparty computation using generalized SPDZ. In *ACM Conference on Computer and Communications Security (CCS)*, pages 2198–2200, 2018.

[21] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM Conference on Computer and Communications Security (CCS)*, pages 805–817, 2016.

[22] A. Baccarini, M. Blanton, and C. Yuan. Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2023(1):608–626, 2023.

[23] R. Barakat. Sums of independent lognormally distributed random variables. *Journal of the Optical Society of America*, 66(3):211–216, 1976.

[24] G. Barthe and B. Kopf. Information-theoretic bounds for differentially private mechanisms. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 191–204, 2011.

[25] N. C. Beaulieu, A. A. Abu-Dayya, and P. J. McLane. Estimating the distribution of a sum of independent lognormal random variables. *IEEE Transactions on Communications*, 43(12):2869, 1995.

[26] N. C. Beaulieu and Q. Xie. An optimal lognormal approximation to lognormal sum distributions. *IEEE Transactions on Vehicular Technology*, 53(2):479–489, 2004.

[27] D. Beaver and A. Wool. Quorum-based secure multi-party computation. In *Advances in Cryptology – EUROCRYPT*, pages 375–390, 1998.

[28] J.-F. Bercher and C. Vignat. Estimating the entropy of a signal with applications. *IEEE Transactions on Signal Processing*, 48(6):1687–1694, 2000.

[29] T. B. Berrett, R. J. Samworth, and M. Yuan. Efficient multivariate entropy estimation via k-nearest neighbour distances. *The Annals of Statistics*, 47(1):288–318, 2019.

[30] A. Bhowmick, D. Boneh, S. Myers, and K. T. K. Tarbe. The Apple PSI system. https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf, 2021.

[31] G. R. Blakley. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.

[32] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 20(2-3):259–305, 2012.

[33] M. Blanton and F. Bayatbabolghani. Improving the security and efficiency of private genomic computation using server aid. *IEEE Security and Privacy*, 15(5):20–28, 2017.

[34] M. Blanton, M. T. Goodrich, and C. Yuan. Secure and accurate summation of many floating-point numbers. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2023(3):432–445, 2023.

[35] M. Blanton, A. Kang, and C. Yuan. Improved building blocks for secure multi-party computation based on secret sharing with honest majority. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 377–397, 2020.

[36] M. Blanton, A. Kang, and C. Yuan. Improved building blocks for secure multi-party computation based on secret sharing with honest majority. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 377–397, 2020.

[37] M. Blanton and S. Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *European Symposium on Research in Computer Security (ESORICS)*, pages 384–406, 2015.

[38] M. Blanton, Y. Zhang, and K. B. Frikken. Secure and verifiable outsourcing of large-scale biometric computations. *ACM Transactions on Information and System Security (TISSEC)*, 16(3):1–33, 2013.

[39] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security (ESORICS)*, pages 192–206, 2008.

[40] S. Bu, L. Lakshmanan, R. Ng, and G. Ramesh. Preservation of patterns and input-output privacy. In *IEEE International Conference on Data Engineering*, pages 696–705, 2006.

[41] C. C. S. Caiado and P. N. Rathie. Polynomial coefficients and distribution of the sum of discrete uniform variables. In *Conference of the Society of Special Functions and their Applications (SSFA)*, 2007.

[42] L. Cao, T. Tong, D. Trafimow, T. Wang, and X. Chen. The a priori procedure for estimating the mean in both log-normal and gamma populations and robustness for assumption violations. *Methodology*, 18(1):24–43, 2022.

[43] O. Catrina. Efficient secure floating-point arithmetic using Shamir secret sharing. In *International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 49–60, 2019.

[44] O. Catrina. Evaluation of floating-point arithmetic protocols based on Shamir secret sharing. In *International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 108–131, 2020.

[45] O. Catrina. Optimizing secure floating-point arithmetic: sums, dot products, and polynomials. In *Proceedings of the Romanian Academy*, pages 21–28, 2020.

[46] O. Catrina. Performance analysis of secure floating-point sums and dot products. In *IEEE International Conference on Communications (COMM)*, pages 465–470, 2020.

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

[48] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, pages 35–50, 2010.

[49] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh. ASTRA: High throughput 3PC over rings with application to secure prediction. In *ACM Workshop on Cloud Computing Security (CCSW)*, pages 81–92, 2019.

[50] H. Chaudhari, R. Rachuri, and A. Suresh. Trident: Efficient 4PC framework for privacy preserving machine learning. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[51] M. Cheraghchi. Expressions for the entropy of basic discrete distributions. *IEEE Transactions on Information Theory*, 65(7):3999–4009, 2019.

[52] H. Cho, D. J. Wu, and B. Berger. Secure genome-wide association analysis using multiparty computation. *Nature Biotechnology*, 36(6):547–551, 2018.

[53] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot. Label-only membership inference attacks. In *International Conference on Machine Learning*, pages 1964–1974, 2021.

[54] D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. *Electronic Notes in Theoretical Computer Science*, 59(3):238–251, 2002.

[55] F. Clementi and M. Gallegati. Pareto's law of income distribution: Evidence for Germany, the United Kingdom, and the United States. In *Econophysics of Wealth Distributions*, pages 3–14. 2005.

[56] B. R. Cobb, R. Rumí, and A. Salmerón. Approximating the distribution of a sum of log-normal random variables. *Statistics and Computing*, 16(3):293–308, 2012.

[57] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, second edition, 2006.

[58] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *Advances in Cryptology – CRYPTO*, pages 769–798, 2018.

[59] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference (TCC)*, pages 342–362, 2005.

[60] A. Dalskov, D. Escudero, and M. Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2020(4):355–375, 2020.

[61] A. Dalskov, D. Escudero, and M. Keller. Fantastic four: Honest-majority four-party secure computation with malicious security. In *USENIX Security Symposium*, pages 2183–2200, 2021.

[62] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1102–1120, 2019.

[63] I. Damgård and J. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology – CRYPTO*, pages 572–590, 2007.

[64] I. Damgård, C. Orlandi, and M. Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. In *Advances in Cryptology – CRYPTO*, pages 799–829, 2018.

[65] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO*, pages 643–662, 2012.

[66] J. Deng, W. Dong, R. Socher, L. Li, K Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[67] D. Denning. *Cryptography and Data Security*. Addison-Wesley Reading, 1982.

[68] V. Deshpande, L. B. Schwarz, M. J. Atallah, M. Blanton, and K. B. Frikken. Outsourcing manufacturing: Secure price-masking mechanisms for purchasing component parts. *Production and Operations Management*, 20(2):165–180, 2011.

[69] V. Deshpande, L. B. Schwarz, M. J. Atallah, M. Blanton, K. B. Frikken, and J. Li. Secure collaborative planning, forecasting and replenishment (SCPFR). CERIAS Tech Report 2006-65, 2005.

[70] V. Deshpande, L. B. Schwarz, M. J. Atallah, M. Blanton, K. B. Frikken, and J. Li. Secure collaborative planning, forecasting and replenishment (SCPFR). In *Multi-Echelon/Public Applications of Supply Chain Management Conference*, pages 165–180, 2006.

[71] X. Dong, D. A. Randolph, C. Weng, A. N. Kho, J. M. Rogers, and X. Wang. Developing high performance secure multi-party computation protocols in healthcare: a case study of patient risk stratification. *AMIA Summits on Translational Science Proceedings*, 2021:200, 2021.

[72] T. Dugan and X. Zou. A survey of secure multiparty computation protocols for privacy preserving genetic tests. In *IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 173–182, 2016.

[73] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 1–19, 2008.

[74] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284, 2006.

[75] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[76] C. Dwork, G. N. Rothblum, and S. Vadhan. Boosting and differential privacy. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 51–60, 2010.

[77] H. Eerikson, M. Keller, C. Orlandi, P. Pullonen, J. Puura, and M. Simkin. Use your brain! arithmetic 3PC for any modulus with active security. In *Conference on Information-Theoretic Cryptography (ITC)*, pages 5:1–5:24, 2020.

[78] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Elsevier, 2004.

[79] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Advances in Cryptology – CRYPTO*, pages 823–852, 2020.

[80] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. IACR Cryptology ePrint Archive Report 2020/338, 2020.

[81] R. J. Evans and J. Boersma. The entropy of a Poisson distribution (C. Robert Appledorn). *SIAM Review*, 30(2):314–317, 1988.

[82] L. Fenton. The sum of log-normal probability distributions in scatter transmission systems. *IRE Transactions on Communications Systems*, 8(1):57–67, 1960.

[83] M. Franz and S. Katzenbeisser. Processing encrypted floating point signals. In *ACM Multimedia Workshop on Multimedia and Security (MM&Sec)*, pages 103–108, 2011.

[84] W. Gao, S. Kannan, S. Oh, and P. Viswanath. Estimating mutual information for discrete-continuous mixtures. *Proceedings on Advances in Neural Information Processing Systems (NeurIPS)*, 30:5988–5999, 2017.

[85] W. Gao, S. Oh, and P. Viswanath. Demystifying fixed k-nearest neighbor information estimators. *IEEE Transactions on Information Theory*, 64(8):5629–5661, 2018.

[86] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Principles of Distributed Computing (PODC)*, pages 101–111, 1998.

[87] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. 2022.

[88] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In *ACM Symposium on Theory of Computing (STOC)*, pages 351–360, 2009.

[89] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[90] R. E. Goldschmidt. Applications of division by convergence. Master's thesis, MIT, 1964.

[91] B. Hilprecht, M. Härterich, and D. Bernau. Monte Carlo and reconstruction membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2019(4):232–249, 2019.

[92] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[93] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth. Differential privacy: An economic method for choosing epsilon. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 398–410, 2014.

[94] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 2022. 10.1145/3523273.

[95] M. Ion, B. Kreuter, A. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389, 2020.

[96] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structures. In *IEEE Global Telecommunication Conference (GLOBECOM)*, pages 99–102, 1987.

[97] M. Ito, N. Takagi, and S. Yajima. Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification. *IEEE Transactions on Computers*, 46(4):495–498, 1997.

[98] M. Iwamoto and J. Shikata. Information theoretic security for encryption based on conditional rényi entropies. In *International Conference on Information Theoretic Security*, pages 103–121, 2013.

[99] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

[100] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium*, pages 1651–1669, 2018.

[101] L. Kamm and J. Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, 14(6):531–548, 2015.

[102] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1575–1590, 2020.

[103] M. Keller, D. Rotaru, N. P. Smart, and T. Wood. Reducing communication channels in MPC. In *International Conference on Security and Cryptography for Networks (SCN)*, pages 181–199, 2018.

[104] M. Keller and K. Sun. Secure quantized training for deep learning. In *International Conference on Machine Learning*, pages 10912–10938, 2022.

[105] L. Kerik, P. Laud, and J. Randmets. Optimizing MPC for robust and scalable integer and floating-point arithmetic. In *International Conference on Financial Cryptography and Data Security*, pages 271–287, 2016.

[106] L. Kerik, P. Laud, and J. Randmets. Optimizing MPC for robust and scalable integer and floating-point arithmetic. In *International Conference on Financial Cryptography and Data Security Workshops*, pages 271–287, 2016.

[107] B. Köpf and D. Basin. Automatically deriving information-theoretic bounds for adaptive side-channel attacks. *Journal of Computer Security*, 19(1):1–31, 2011.

[108] R. Kotecha and S. Garg. Preserving output-privacy in data stream classification. *Progress in Artificial Intelligence*, 6:87–104, 2017.

[109] A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, 2004.

[110] B. Kreuter. Secure multiparty computation at Google. Real World Crypto, 2017. Available from https://www.youtube.com/watch?v=ee7oRsDnNNc.

[111] T. Krips and J. Willemson. Hybrid model of fixed and floating point numbers in secure multiparty computations. In *International Conference on Information Security (ISC)*, pages 179–197, 2014.

[112] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma. CrypTFlow: Secure TensorFlow inference. In *IEEE Symposium on Security and Privacy (S&P)*, pages 336–353, 2020.

[113] A. Lapets, N. Volgushev, A. Bestavros, F. Jansen, and M. Varia. Secure MPC for analytics as a web application. In *IEEE Cybersecurity Development (SecDev)*, pages 73–74, 2016.

[114] P. Laud and J. Randmets. A domain-specific language for low-level secure multiparty computation protocols. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1492–1503, 2015.

[115] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. In *USENIX Security Symposium*, pages 5377–5394, 2023.

[116] Ligero. Secure and private collaboration for blockchains and beyond. https://ligero-inc.com/, 2022. Last accessed: 2022-08-16.

[117] Y. Lindell and A. Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest majority. In *ACM Conference on Computer and Communications Security (CCS)*, pages 259–276, 2017.

[118] F. Liu. Generalized gaussian mechanism for differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 31(4):747–756, 2018.

[119] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM Conference on Computer and Communications Security (CCS)*, pages 619–631, 2017.

[120] Y.-C. Liu, Y.-T. Chiang, T.-S. Hsu, C.-J. Liau, and D.-W. Wang. Floating point arithmetic protocols for constructing secure data analysis application. *Procedia Computer Science*, 22:152–161, 2013.

[121] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. In *International Conference on Financial Cryptography and Data Security*, pages 249–270, 2021.

[122] P. Mardziel, M. Hicks, J. Katz, and M. Srivatsa. Knowledge-oriented secure multiparty computation. In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 1–12, 2012.

[123] P. Markstein. Software division and square root using Goldschmidt's algorithms. In *Real Numbers and Computers*, pages 146–157, 2004.

[124] M Marwan, A. Kartit, and H. Ouahmane. Security enhancement in healthcare cloud using machine learning. *Procedia Computer Science*, 127:388–397, 2018.

[125] J. L. Massey. Guessing and entropy. In *IEEE International Symposium on Information Theory (ISIT)*, page 204, 1994.

[126] U. Maurer. Secure multi-party computation made simple. In *Security in Communication Networks (SCN)*, pages 14–28, 2002.

[127] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 94–103, 2007.

[128] R. Mendes and J. Vilela. Privacy-preserving data mining: methods, metrics, and applications. *IEEE Access*, 5:10562–10582, 2017.

[129] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *Advances in Cryptology – CRYPTO*, pages 126–142, 2009.

[130] P. Mohassel and P. Rindal. ABY^3: A mixed protocol framework for machine learning. In *ACM Conference on Computer and Communications Security (CCS)*, pages 35–52, 2018.

[131] A. Monreale and W. Wang. Privacy-preserving outsourcing of data mining. In *IEEE International Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 583–588, 2016.

[132] K. R. Moon, J. S. Stanley III, D. Burkhardt, D. van Dijk, G. Wolf, and S. Krishnaswamy. Manifold learning-based methods for analyzing single-cell RNA-sequencing data. *Current Opinion in Systems Biology*, 7:36–46, 2018.

[133] M. Nasr, R. Shokri, and A. Houmansadr. Machine learning with membership privacy using adversarial regularization. In *ACM Conference on Computer and Communications Security (CCS)*, pages 634–646, 2018.

[134] J. P. Near, D. Darais, N. Lefkovitz, and G. Howarth. Guidelines for evaluating differential privacy guarantees. Technical Report BUCS-TR-2016-008, NIST, 2023.

[135] L. Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6):1191–1253, 2003.

[136] A. Patra and A. Suresh. BLAZE: Blazing fast privacy-preserving machine learning. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[137] A. Rahimzamani, H. Asnani, P. Viswanath, and S. Kannan. Estimators for multivariate information measures in general probability spaces. *Proceedings on Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[138] A. Rastogi, P. Mardziel, M. Hicks, and M. A. Hammer. Knowledge inference for optimizing secure multi-party computation. In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 3–14, 2013.

[139] D. Rathee, A. Bhattacharya, D. Gupta, R. Sharma, and D. Song. Secure floating-point training. In *USENIX Security Symposium*, pages 6329–6346, 2023.

[140] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi. SecFloat: Accurate floating-point meets secure 2-party computation. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1553–1553, 2022.

[141] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi. SiRnn: A math library for secure RNN inference. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1003–1020, 2021.

[142] J. F. Reiser and D. E. Knuth. Evading the drift in floating-point addition. *Information Processing Letters*, 3(3):84–87, 1975.

[143] D. Rotaru and T. Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *INDOCRYPT*, pages 227–249, 2019.

[144] K. Sasaki and K. Nuida. Efficiency and accuracy improvements of secure floating-point addition over secret sharing. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 105(3):231–241, 2022.

[145] S. C. Schwartz and Y.-S. Yeh. On the distribution function and moments of power sums with log-normal components. *Bell System Technical Journal*, 61(7):1441–1462, 1982.

[146] SecureSCM. Deliverable D9.2, EU FP7 Project Secure Supply Chain Management (SecureSCM). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.393&rep=rep1&type=pdf, 2009.

[147] D. Senaratne and C. Tellambura. Numerical computation of the lognormal sum distribution. In *IEEE Global Telecommunication Conference (GLOBECOM)*, pages 1–6, 2009.

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

[149] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[150] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–18, 2017.

[151] M. Skórski. Strong chain rules for min-entropy under few bits spoiled. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1122–1126, 2019.

[152] G Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302, 2009.

[153] L. Song and P. Mittal. Systematic evaluation of privacy risks of machine learning models. In *USENIX Security Symposium*, pages 2615–2632, 2021.

[154] L. Song, R. Shokri, and P. Mittal. Membership inference attacks against adversarially robust deep learning models. In *IEEE Symposium on Security and Privacy Workshops (SPW)*, pages 50–56, 2019.

[155] W. Souma. Physics of personal income. In *Empirical Science of Financial Fluctuations*, pages 343–352, 2002.

[156] S. P. Strong, R. Koberle, R. R. D. R. van Steveninck, and W. Bialek. Entropy and information in neural spike trains. *Physical Review Letters*, 80(1):197, 1998.

[157] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing*, 14(6):2073–2089, 2021.

[158] M. Veeningen, S. Chatterjea, A. Z. Horváth, G. Spindler, E. Boersma, P. van der Spek, O. Van Der Galiën, J. Gutteling, W. Kraaij, and T. Veugen. Enabling analytics on sensitive medical data with secure multi-party computation. In *Building Continents of Knowledge in Oceans of Data: The Future of Co-Created eHealth*, pages 76–80. 2018.

[159] J. D. Victor. Binless strategies for estimation of information from neural data. *Physical Review E*, 66(5):051903, 2002.

[160] S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2019(3):26–49, 2019.

[161] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2021(1):188–208, 2021.

[162] A. Walker, S. Patel, and M. Yung. Helping organizations do more without collecting more data. *Google Security Blog*, jun 2019. Last accessed: 2022-08-16.

[163] T. Wang and L. Liu. Output privacy in data mining. *ACM Transactions on Database Systems (TODS)*, 36(1):1–34, 2011.

[164] J. Wu, N. B. Mehta, and J. Zhang. Flexible lognormal sum approximation method. In *IEEE Global Telecommunication Conference (GLOBECOM)*, pages 3413–3417, 2005.

[165] Y. Zhang, A. Steele, and M. Blanton. PICCO: A general-purpose compiler for private distributed computation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 813–826, 2013.