Abstract

Messages, Secrets, and Catalysts in Population Protocols With Probabilistic Scheduling: Theory Motivated By Application

Talley Amir

2023

The population protocol model [AAD+06] offers a theoretical framework for designing and analyzing distributed algorithms among limited-resource mobile agents. The original model elegantly abstracts computation in complex systems of simple devices. However, some of these simplifications detract from the pragmatism of the theory in properly modeling real-world systems. In this work, we study various adaptations of the original population protocol model that are inspired by the networks that these algorithms aim to simulate. We focus on three such modifications to the model.

First, we consider the separation of agent memory into two components: an externally visible "message" and an internally hidden state. This distinction is inspired by the difference between computational capacity and communication bandwidth in sensors and other limited-resource devices. We design a synchronous broadcast primitive which we then implement to perform more complex computations. We show that symmetric functions can be computed by population protocols with exactly 1-bit messages when the internal state space is unbounded. Then, we show that 1-bit messages are sufficient to stably simulate Turing machine computations with known time and space complexity under a uniform random scheduler.

Next, we formalize private input computation in the population protocol model with probabilistic scheduling. While the original population protocol model considers the concept of anonymity, the issue of privacy is not investigated thoroughly. However, there is a need for time- and space-efficient privacy-preserving techniques in the population protocol model if these algorithms are to be implemented in settings handling sensitive data, such as within wireless sensor networks or systems of medical wearables. In this work, we introduce various formal definitions of privacy and apply these definitions to both existing and novel protocols. We show that the `Remainder`-computing protocol from [DGFGR07] (which is proven to satisfy output independent privacy under adversarial scheduling) is not information-theoretically private under probabilistic scheduling. In contrast, we provide a new algorithm and show that it correctly and information-theoretically privately computes `Remainder` in populations with probabilistic scheduling.

Finally, we introduce and study a variation of the population protocol model wherein certain states are persistent. This modification to the original model is motivated by applications of population protocols in chemical reaction networks. We formally define a variant of the population protocol model where some number of agents are understood to never change their state, and the remaining agents in the population aim to compute some function over these persistent states. We show that for total population size N, `Parity` cannot be computed in this new model in fewer than $\Omega(N^2)$ steps, and a lower bound of $\Omega(\sqrt{N})$ on the initial input margin is needed to compute `Majority` in $O(N \log N)$ steps. We also study the effect of spontaneous state changes, or *leaks*, on the computation of the `Majority` problem in the presence

of persistent states. We prove that an adaptation of third-state dynamics can be used to solve approximate majority in the presence of persistent states and leaks, both separately and together. Lastly, we form a comparison between the effects of leaks and that of previously studied Byzantine processes on population protocols.

Messages, Secrets, and Catalysts in Population Protocols
With Probabilistic Scheduling:
Theory Motivated By Application

A Dissertation
Presented to the Faculty of the Graduate School
Of
Yale University
In Candidacy for the Degree of
Doctor of Philosophy

By
Talley Amir

Dissertation Director: James Aspnes

May 2023

This dissertation is dedicated to Gryphon and Rudy,

who inspire me to be positive, passionate, and endlessly curious.

Acknowledgment

I would like to express deeply sincere gratitude to my research advisor, James Aspnes. Your guidance extends far beyond the study of population protocols — I have greatly appreciated the hilarious anecdotes, profound wisdom, and dead-pan humor you bring to each of our meetings, not to mention all of the time you have generously invested in my academic development. “Thank you” is not enough.

I would like to recognize my dissertation committee members: Michael J. Fischer, Joan Feigenbaum, and David Doty. The feedback and advice you have given me throughout the pursuit of my degree has shaped me as a researcher and as a thinker. I am so grateful to each of you for the impact you have had on me.

I would like to thank my collaborators for many thought-provoking and eye-opening discussions, both early in the morning and late at night, without which this dissertation could not exist in its current state. A special thanks to my coauthors: James Aspnes, David Doty, Mahsa Eftekhari, John Lazarsfeld, and Eric Severson.

Finally, I must thank my incredible family and friends, whose support and encouragement helped keep my head held high through repeated rejection and failure — we finally did it.

Funding This work was funded by the Yale Graduate School of Arts & Sciences University Fellowship.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Population protocols, introduced by [AAD$^+$06], are a class of distributed algorithms which model computation in networks of mobile finite-state agents. The motivation for studying this theoretical model is to study population dynamics in settings where interaction is random, but frequent. For example, population protocols can be used to compute a function over the states of various "agents" in a network, such as resource-restricted sensors attached to animals, cars, or people. Such agents are typically presumed to have limited communication range, making population protocols a suitable model for networks of drones, Internet of Things (IoT) devices, and technological wearables. These protocols can also be used to predict the behavior and concentrations of molecules in a well-mixed solution. It is worth noting that population protocols consider only systems wherein agents interact pairwise and update their states as a consequence of that interaction (and therefore population protocols do not model all chemical reaction networks, for example where the total molecule count is not preserved).

In this model, agents interact with one another asynchronously in pairs chosen by a scheduler at discrete time steps. Agents interact in pairs to mimic the event that any two agents travel within close enough proximity to communicate with one another (or in the case of molecules, to initiate a chemical reaction). The agents can collectively compute a function by updating their states according to some transition function, where the state of the agent corresponds to some output indicating the evaluation of the function on the initial collection of states of the agents. After a certain amount of time, we want all subsequent interactions to result in no change in any agent's state, thereby making the function stably computable [AAFJ08]. Some examples of functions that are stably computable by population protocols include detecting a majority input value and electing a leader [AAD$^+$06].

Population protocols have been used to study a multitude of systems consisting of simple mobile computers [AAFJ05, DGFGR06, DGFGR07] as well as chemical reaction networks [CDS12] and DNA strand displacement [CDS$^+$13, TWS15]. Various well-studied problems in computer science have been solved exactly or nearly optimally in the population protocol model, including leader election [BGK20], majority [AAG18a, GDE$^+$21], and counting [ABBS16]. Even general computations by a LOGSPACE Turing machine can be simulated by populations with an initial leader [AAE06a].

In many ways, population protocols (which were originally developed in order to study real-world systems) do not perfectly parallel the networks they claim to model. Often times, wireless sensor networks and other interconnected systems of simple computational devices must consider trade-offs between computing and communication efficiency, data privacy, and authoritative sources of information. Yet, these considerations were not included in the original population protocol model. As such, we investigate various modifications to the original model in order to adapt the population protocol theory to better serve practical purposes.

In this work, we formally define concepts which are novel in the context of population protocols and that are motivated by the applications of the model.

In Chapter 3, we introduce messages, a distinct component of an agent's state that it shared when it interacts with another agent, to signify the separation between a device's communication space and internal computation state. With this new abstraction, we construct a 1-bit synchronous convergent broadcast primitive and show that $O(1)$ external messages are powerful enough to simulate a Turing Machine with probability 1 correctness when local memory is unbounded. These results appeared previously in [AAD$^+$20].

In Chapter 4, we study privacy within the model of population protocols when the interaction pattern of the agents is probabilistic rather than adversarial, and we demonstrate how existing privacy definitions fail in this setting. We provide new definitions for privacy in the probabilistic scheduling model and give an algorithm for computing the `Remainder` predicate that satisfies these definitions.

In Chapter 5, we define and study a variation of the population protocol model wherein certain states are persistent and show that this yields new lower bounds on conditions for solving these problems. We then study the effects of these persistent states, as well as spontaneous state changes, on the well-studied third-state dynamics process [AAE08, PVV09] in solving approximate majority, the problem of detecting which of two opinions is initially more popular in a population conditioned on there being a sufficiently large margin between them. These results appeared previously in [AAL20].

First, we begin with Chapter 2, where we introduce the definitions, notation, and foundational concepts of population protocols.

Chapter 2

Preliminaries

In this chapter, we will cover the definitions and fundamental concepts of population protocols that are needed in order to understand the results of the subsequent chapters. In general, we will use the convention that random variables appear in mathematical boldface.

2.1 Definitions and Notation

A **population protocol** $\mathcal{P}$ is a tuple $(Q, \delta, \Sigma, \mathcal{I}, O, \mathcal{O})$ consisting of **state set** Q, **transition function** δ, **input set** Σ, **input function** $\mathcal{I}$, **output set** O, and **output function** $\mathcal{O}$ [AAD+06]. The state set is some finite set of states allowed by the protocol; each agent in the population is in some state $q \in Q$. The transition function $\delta : Q \times Q \to Q \times Q$ designates how the agents update their states upon interacting with each other in pairs. As a shorthand for saying $\delta(q_1, q_2) = (q_1', q_2')$, we may write $q_1, q_2 \to q_1', q_2'$ where δ is implied.

Before starting the protocol computation, each agent has some input in the set Σ which is converted to a state in Q via $\mathcal{I} : \Sigma \to Q$. In the early population protocol literature, $\mathcal{I}$ is only ever considered to be a deterministic function; however, in this work, we extend the model to allow for $\mathcal{I}$ to be randomized, thereby changing the function specification to be $\mathcal{I} : \Sigma \times \Omega \to Q$, where $\Omega = \{0,1\}^\omega$ represents an input of some uniform ω bits used to randomize the computation of $\mathcal{I}$ on some input from Σ. Within this work, we assume that each agent has access to independent fair coin flips from which to generate samples from Ω when computing $\mathcal{I}$.

The protocol aims to compute some function (whose output is in the output set O) on the inputs of the agents in the population. An agent's computed output value is determined by $\mathcal{O} : Q \to O$.

Protocols are run by a population, which consists of a set of n agents $\{A_j\}_{j=1}^n$ each with some input $i_j \in \Sigma$. The protocol $\mathcal{P}$ may be annotated with the size of the population n (for instance, $\mathcal{P}_n$) because the population size may affect components of the protocol (e.g. a protocol that counts the exact size of the population needs at least n states, which lower bounds the size of Q). However, if $\mathcal{P}$ does not depend on n then we call the protocol **uniform**. Let the size of the state set $|Q| = m$; then for uniform protocols, $m = O(1)$ is a constant.

The collection of agents' inputs is denoted as a vector $I \in \Sigma^n$, where each index of I reflects the input of a particular agent in the population. Adopting the terminology from [DGFGR07], we refer to I as an **input vector**. When the size of the state space is $O(1)$, the protocol cannot distinguish between two agents in the same state; therefore, we may want to refer to the multiset of input values in the input vector I, denoted $\text{multiset}(I)$.

Each agent applies $\mathcal{I}$ to its input to obtain its initial state $q_j = \mathcal{I}(i_j)$. The global state of the population at any instance is called a **configuration** and is represented as a vector $C \in Q^n$, where the i-th entry of the vector denotes the state of the i-th agent. Abusing notation, we say that $\mathcal{I}(I) = \langle \mathcal{I}(i_j) \rangle_{j=1}^n$ is the configuration resulting from applying the input function $\mathcal{I}$ to each of the agent inputs in $I = \langle i_j \rangle_{j=1}^n$. As a shorthand, we denote by $|q_j|$ the count of agents in the population in state q_j.

Agents update their states via interactions with one another which are performed at discrete intervals, called **steps**. At each step, an ordered pair of agents (A_i, A_j) is selected from the population by the **scheduler**, whose selection process may either be adversarial or probabilistic (see Section 2.2). To distinguish between the two agents in the ordered pair, we call the first element of the pair the **Initiator** and the second element the **Responder**. Some works consider protocols where the Initiator does not update its state, and may not know that an interaction has even taken place; such protocols are referred to as **one-way** protocols [AAER07]. Where both agents are aware of the interaction, and either or both may update their states as a result, the protocol is called **two-way** [AAER07].

When an interaction takes place, the two selected agents update their states according to the transition function δ which may change the counts of states in the population, thereby updating the configuration. Let $\mathcal{C}$ be the configuration space, or the set of all possible configurations for a population of n agents with state space Q. We say that a configuration $D \in \mathcal{C}$ is **reachable** from $C \in \mathcal{C}$ via δ if there exists some series of ordered agent pairs such that starting from C, if the configuration is updated according to δ on those ordered pairs, then the resulting configuration is D [AAD+06]. If D is reachable from C, then we write $C \to D$.

The infinite sequence of configurations induced by the scheduler's infinite choice of interaction pairs, called an **execution**, is denoted by $E = (C^0, C^1, C^2, ...)$. Note that we may have two executions beginning at the same value $C^0 = C$ resulting in different subsequent configurations due to the non-determinism of the scheduler. If there exists some element $\omega \in \Omega$ such that $\mathcal{I}(I, \omega) = C^0$, as well as some selection of ordered agent pairs yielding the subsequent configurations $C^1, C^2, ...$ in the same order as in E, then we say that E is **derivable** from I and write this as $I \vdash E$ (in simple terms, I is a possible input vector that could yield the execution E).

An execution of a protocol is said to **converge** at a step τ when, for every step $t > \tau$, the output of each agent's state at step t is the same as it is at step τ (i.e.

the output of every agent converges to some value and never changes thereafter). A stronger notion of termination is for a protocol to **stabilize**, meaning that after some point τ, the only configurations that are reachable from C^τ result in the same outputs at every agent as in C^τ.

2.2 Scheduling

In population protocols, the scheduler determines the order in which pairs of agents interact. The scheduler's choice of agent pairs may either be adversarial or probabilistic.

An **adversarial scheduler** chooses pairs of agents to interact at each step as it desires, subject to a fairness condition. The condition used most commonly is called **strong global fairness**, and it states that if some configuration C occurs infinitely often, and $C \to C'$, then C' must occur infinitely often as well [AAD$^+$06]. This means that if some configuration *can* occur, it eventually *must* occur, even if the adversarial scheduler wishes to delay its occurrence indefinitely. In works adopting adversarial scheduling, it can be claimed that a protocol eventually stabilizes to the correct answer, but not how quickly.

A random or **probabilistic scheduler** instead selects pairs of agents to interact with one another according to some fixed probability distribution over the ordered pairs of agents, typically assumed to be the uniform distribution. Unlike adversarial scheduling, random scheduling allows us to infer a notion of time. Although population protocols consider each pairwise interaction to occur in sequence, the systems they are modelling typically consist of agents participating in interactions in parallel. As such, a natural estimation of **parallel time** is to divide the total number of interactions by n, as this roughly estimates the average number of interactions initiated by a particular agent in the population. Note that in population protocols with *non-uniform* random scheduling, this notion of time is no longer necessarily suitable.

In protocols with non-uniform random scheduling, we are no longer guaranteed to satisfy the global fairness condition because the non-uniform distribution may define certain interaction events to be impossible. Thus for non-uniform random schedulers, we must restrict that every ordered pair of agents has some non-zero probability of being selected to interact at each step in order to avoid some such pairs from being excluded from the protocol computation indefinitely.

2.3 Predicate and Function Computation

Recall that the output function $\mathcal{O}$ maps each state $q \in Q$ to an output in the output set O. Abusing notation, we say $\mathcal{O}(C) = \lambda$ (or, the output of the *configuration* is λ) if $\mathcal{O}(q_j) = \lambda$ for every $q_j \in C$. Suppose that we aim to compute some function Φ on the input vector $I \in \Sigma^n$. Our aim is to design an algorithm δ that results in the population eventually stabilizing towards a set of configurations $\mathcal{D} \subseteq \mathcal{C}$ for which $\mathcal{O}(D) = \Phi(I)$ for all $D \in \mathcal{D}$.

Notice that we can induce an equivalence relation on the input vector space by partitioning Σ^n into subsets wherein all input vectors evaluate to the same output. We let

$$\mathcal{S}_\lambda = \{I \in \Sigma^n : \Phi(I) = \lambda\}$$

represent the set of all input vectors I which map to the same output λ via Φ.

Chapter 3

Message Complexity

In this chapter, we explore the impact of differentiating between the internal state complexity and the message complexity of population protocols. Motivated by real-world differences between computational and communication bandwidth, we study the capabilities and limitations of protocols which use constant message space and superconstant internal state space to solve classical problems in the population protocol model.

We devise a synchronous broadcast primitive which we then implement to perform more complex computations: In Theorem 3.1, we show that symmetric functions can be computed by population protocols with exactly 1-bit messages when the internal state space is unbounded. Then, in Theorem 3.2, we show that 1-bit messages are sufficient to stably simulate Turing machine computations with known time and space complexity under a uniform random scheduler.

The results of this chapter were produced in a joint work with James Aspnes, David Doty, Mahsa Eftekhari, and Eric Severson (see [AAD$^+$20]).

3.1 Introduction

The original population model [AAD$^+$06] limited agents to $O(1)$ states, independent of the population size n. This restricted the computational power (to only semilinear predicates [AAE06b]) and the optimal time efficiency in performing fundamental tasks (such as the linear time lower bound for leader election [DS15]).

By using $\omega(1)$ states, recent algorithms in population protocols are able to perform fundamental tasks in the population model with improved efficiency over $O(1)$-state protocols (e.g., [AAE$^+$17, AG15, AGV15, AAG18a, BGK20, GS18, GSU19, KU18, SOI$^+$19]). Superconstant state space results in provably faster protocols, begging the question: Is the improved performance a consequence of higher communication throughput or higher local storage capacity?

The original model supposes that agents can view the entirety of the other's local state upon interacting with another agent, which we call an **open** protocol. We introduce a new variant of this model that draws a distinction between the state of the agent and the segment of the state that is externally visible to its interacting partner, called the **message**. This variant generalizes previous work in the context of consensus that examines the particular case of **binary signaling** [PVV09, AAC16], where the message is limited to a single bit. In this chapter, we study the computational power of population protocols that have $O(1)$ message complexity and varying local state complexity, ranging from $\omega(1)$ to unbounded.

The motivation for introducing this distinction to the theoretical model is rooted in applications of population protocols. The population protocol framework was originally conceived to model passively mobile ad hoc sensor networks. In this setting the amount of communication bandwidth can be a tighter constraint than the local computation performed by a sensor. These two constraints—bandwidth efficiency and energy efficiency—are viewed as distinct in the networking literature. In some scenarios it makes more sense to optimize for one or the other, or to strike a balance

[UVV13, LLZ+12, DY13]. The restriction to constant messages but superconstant internal states is germane when the communication in an interaction is more costly than the accompanying local computation. Naturally, it would make sense to ask how this separation of resources might influence computation in the population model.

Synthetic chemistry is another domain in which population protocols are an appropriate abstract model of computation. They are a richly-featured subclass of chemical reaction networks, which are known to have similar computational power [SCWB08, CDS12]. Using a physical primitive called **DNA strand displacement** [YTM+00], *every* chemical reaction network with $O(1)$ species (**states** in the language of population protocols) can be theoretically implemented by a set of DNA complexes [SSW08], justifying the use of chemical reactions as an implementable programming language. Using this approach, nontrivial chemical systems have been synthesized in the wet lab, resulting in pure DNA implementations of a chemical oscillator [SPS+17] and the "approximate majority" population protocol [CDS+13, AAE08]. Some theoretical [QSW10] and experimental [WDM+19] systems are able to assemble unbounded-length heteropolymers such as DNA in an algorithmic way. For such systems, reactions may best be modeled as allowing arbitrarily many states (exponential in the polymer length) but only $O(1)$ messages modeling the smaller "locally visible region" near one or both ends of the polymer.

Finally, protocols with superconstant internal states and $O(1)$ external messages are a natural mathematical intermediate between the original $O(1)$ states/messages model and the more recent superconstant states/messages model. Because population protocols with superconstant states and messages are provably more powerful than those with constant state space [CMN+11], it is intrinsically interesting to determine how powerful is this intermediate model.

3.2 Related Work

Most protocols with state space of size $\omega(1)$ [AAG18a, GS18, GSU19, BKR19, DE19, AAE$^+$17, AG15, SOI$^+$19, BKKO18, BCER17, BCC$^+$21, MAS16, MAA$^+$15, AAG18a, BEF$^+$18, BEF$^+$20] crucially use $\omega(1)$-size messages. Key transitions in such protocols involve comparing two integers or unique identifiers of size $\omega(1)$ in a single step, which is not possible with $O(1)$-size messages. Sending a superconstant-size message over multiple interactions using $O(1)$-size messages is not efficient (nor realistic) since there is not enough time for the two agents to wait for another interaction (which takes $\Theta(n)$ expected time), nor is there any way to re-identify an agent in less than $\Theta(n)$ expected time due to lack of space to hold unique identifiers using $O(1)$-states.

Instead, we will aim to circumvent this limitation by using unbounded internal state complexity to perform timed broadcasts in the population.

3.3 Contribution

We explore the extreme limits of the message model for population protocols where message complexity is limited to exactly 1 bit. We construct a 1-bit synchronous convergent broadcast primitive, showing that $O(1)$ external messages are powerful enough to simulate a Turing Machine with probability 1 correctness when local memory is unbounded.

3.4 Message Model for Population Protocols

Adopting notation from [AAD$^+$20], we represent the **internal state** space as a set S and the set of **messages** which can be sent between the agents as M. Since each agent has both an internal and external state component, the total state space is then the Cartesian product of these two sets, $Q = S \times M$. This means that δ is now a function computed locally at each agent according to its own state and the "message received" by its interacting partner. So δ is modified to be a mapping from $S \times M \times M \times \{\mathsf{I}, \mathsf{R}\}$ to $S \times M$, where I represents that the agent is the Initiator of the interaction and R represents that it is the Responder. This new mapping enforces the restriction that an agent can only use the message component of its interacting partner's state to update its own state, and it does not observe the update to its interacting partner's state. For convenience, we may use the original shorthand notation $\langle s_0, m_0\rangle, \langle s_1, m_1\rangle \rightarrow \langle s_0', m_0'\rangle, \langle s_1', m_1'\rangle$ to signify the changes which occur to an agent's state in a given transition, where it is understood that the state update of either agent A_b is computed independently of s_{1-b}.

3.5 Computability with 1-bit Messages

We will show that with 1-bit messages, it is possible to simulate a synchronous system that provides a 1-bit broadcast channel. This will be used to simulate more complex systems. We sacrifice stabilization for convergence and rely on unbounded counters to ensure convergence in the limit with probability 1. Let us begin by defining the simulated system.

A **synchronous broadcast system** consists of n synchronous agents that carry out a sequence of **rounds**. In a broadcast round, each agent generates a 1-bit outgoing message. We refer to b-messages as message states equalling b. These messages are combined using the OR function to produce the outcome for a round.

Broadcast operations can be used to detect conditions such as the presence of a leader, or ordinary message transmission if a unique agent is allowed to broadcast in a particular round. However, because broadcast operations are symmetric, they cannot be used for symmetry breaking. For the purpose of electing a leader, we assume that agents have the ability to flip coins; once we have a leader, additional agents may be recruited for particular roles using an auxiliary protocol that allows the leader to select a single agent from the population in some round. The broadcast and selection protocols are mutually exclusive: either all agents participate in a broadcast in some round or all agents participate in selection. This is possible by showing that all agents eventually agree on the round number forever with probability 1.

Simulating this model in a population protocol requires (a) enforcing synchrony across agents, so that each agent updates its state consistently with the round structure; (b) implementing the broadcast channel that computes the OR of the agents' outputs; and (c) implementing the selection protocol. We show how to do this in the remainder of this chapter.

3.5.1 Utilities

In this section we provide a few lemmas which give general results about timing and epidemic propagation in population protocols. These lemmas will be useful in proving the correctness of this chapter's main results.

Clock Drift Lemma

Lemma 3.1. *Consider some interval of an execution of a population protocol with uniform random scheduling. Let $\boldsymbol{A_{is}}$ be the indicator variable for the event that agent i is one of the two agents that interact in step s of this interval. Let $\boldsymbol{C_{it}} = \sum_{s=1}^{t} \boldsymbol{A_{is}}$ be the cumulative number of interactions involving agent i during the first t steps of the interval. Fix two agents i and j, and let τ be the first time at which $\boldsymbol{C_{i\tau}} + \boldsymbol{C_{j\tau}} = m$. Then* $\Pr(\max_{t\le\tau}(\boldsymbol{C_{it}} - \boldsymbol{C_{jt}}) > b) \le e^{-b^2/2m}$.

Proof. Because only steps involving at least one of i or j change $\boldsymbol{C_{is}}$ and $\boldsymbol{C_{js}}$, we can restrict our attention to the sequence of steps $s_1, s_2, \dots$ at which at least one of i or j interacts. Let $\boldsymbol{X_k} = \boldsymbol{A_{it_k}} - \boldsymbol{A_{jt_k}}$; then $\mathbb{E}(\boldsymbol{X_k} \mid \boldsymbol{X_1}, \dots, \boldsymbol{X_{k-1}}) = 0$ and the $\boldsymbol{X_k}$ form a martingale difference sequence with $\mathbb{E}(|\boldsymbol{X_k}|) \le 1$. We also have that

$\boldsymbol{C_{it_k}} - \boldsymbol{C_{jt_k}} = \sum_{\ell=1}^{k} \boldsymbol{X_\ell}$, so $\max_{t \le \tau}(\boldsymbol{C_{it}} - \boldsymbol{C_{jt}}) > b$ if and only if there is some k with $t_k \le \tau$ such that $\sum_{\ell=1}^{k} \boldsymbol{X_\ell} > b$. Define the truncated martingale difference sequence $\boldsymbol{Y_k} = \boldsymbol{X_k}$ if $t_k \le \tau$ and $\sum_{\ell=1}^{k-1} \boldsymbol{X_\ell} \le b$, and $\boldsymbol{Y_k} = 0$ otherwise. Let $\boldsymbol{S_k} = \sum_{\ell=1}^{k} \boldsymbol{Y_\ell}$.

We have defined $\boldsymbol{S_k}$ so that it tracks $\boldsymbol{C_{it_k}} - \boldsymbol{C_{jt_k}}$ until that quantity reaches $b+1$ or t_k reaches τ, after which $\boldsymbol{S_k}$ does not change. The condition $t_k = \tau$ occurs for some $k \le m$, so if $\boldsymbol{C_{it_k}} - \boldsymbol{C_{jt_k}}$ reaches $b+1$ before $t_k > \tau$, it must do so for some $k \le m$, after which $\boldsymbol{S}$ will not change, giving $\boldsymbol{S_m} = \boldsymbol{S_k}$. So $\Pr(\max_{t \le \tau}(\boldsymbol{C_{it}} - \boldsymbol{C_{jt}}) > b) = \Pr(\boldsymbol{S_m} > b) \le e^{-b^2/2m}$, by the Azuma-Hoeffding inequality. □

Corollary 3.1. *For any agents i and j, and any m and b,* $\Pr(\exists\, t : \boldsymbol{C_{it}} = m \wedge \boldsymbol{C_{jt}} > m + b) < e^{-b^2/(2m+b+1)}$.

Proof. If $\boldsymbol{C_{jt}} - \boldsymbol{C_{it}} > b$ when $\boldsymbol{C_{it}} = m$, then there is some first time s at which $\boldsymbol{C_{js}} - \boldsymbol{C_{is}} > b$. Because $s \le t$, $\boldsymbol{C_{is}} \le \boldsymbol{C_{it}} = m$, and because this is the first time at which $\boldsymbol{C_{js}} - \boldsymbol{C_{is}} > b$, we have $\boldsymbol{C_{js}} = \boldsymbol{C_{is}} + b + 1 \le m + b + 1$. So s is a time at which $\boldsymbol{C_{is}} + \boldsymbol{C_{js}} \le 2m + b + 1$ with $\boldsymbol{C_{js}} - \boldsymbol{C_{is}} > b$. Now apply Lemma 3.1. □

Epidemic-based Broadcast

The following lemma is given and proven by [AAE06a].

Lemma 3.2 (see [AAE06a], Lemma 2)**.** *Let $\boldsymbol{T}(k)$ be the number of interactions before a one way epidemic starting with a single infected agent infects k agents. For any fixed $\epsilon > 0$ and $c > 0$, there exist positive constants c_1 and c_2 such that for sufficiently large n and any $k > n^\epsilon$, $c_1 n \ln k \le \boldsymbol{T}(k) \le c_2 n \ln k$ with probability at least $1 - n^{-c}$.*

3.5.2 Convergent Broadcast Algorithm

Broadcasts are implemented by epidemics that propagate 1-messages, separated by barrier phases in which all agents display 0. Selection is implemented by having the leader display a 1 to the first agent it meets. Both protocols depend on the number of steps at each agent being approximately synchronized with high probability; after $t(n/2)$ steps, all agents' step counts should be within the range $t \pm O(\sqrt{t \log n})$ with high probability (by Lemma 3.1); the time to carry out a broadcast is also $O(\log n)$ with high probability (by Lemma 3.2). By increasing the length of each round over time, the total probability across all rounds of an error occurring in either the broadcast or selection protocol due to out-of-sync agents or slow broadcasts converges to a finite value. Applying the Borel-Cantelli lemma then shows that there is a round after which no further failures occur with probability 1.

Algorithm Description

Observe that the probability that a particular agent A_i participates in an interaction is exactly $2/n$, and that the events that A_i participates in distinct interactions are independent. If we let $\boldsymbol{X_i^t}$ be the indicator variable that agent A_i participates in the t-th interaction, then $\boldsymbol{S_i^t} = \sum_{j=1}^{t} \boldsymbol{X_i^t}$ is a sum of independent Bernoulli random variables, and obeys the Chernoff bound $\Pr(|\boldsymbol{S_i^t} - \mu| > \mu\delta) < 2e^{-\mu\delta^2/3}$, where $\mu = 2t/n$ and $0 \leq \delta \leq 1$.

The execution of each agent is organized as a sequence of rounds, where each round r for $r = 1, 2, \ldots$ consists of exactly $5r^2$ steps. The first $2r^2$ steps will be a **barrier phase** during which the agent displays message 0 and updates its state during an interaction only by incrementing its step counter. The remaining $3r^2$ steps will be an **interaction phase** in which the agents may execute one of two protocols. In a **broadcast phase**, each agent will propagate an epidemic represented by message 1, recording if it observed such an epidemic and possibly initiating the epidemic itself if instructed to do so by the protocol. In a **selection phase**, a leader agent displays 1 for its first encounter, and the agent interacting with the leader receives a special mark. The choice of broadcast/selection phase is determined by the controlling protocol and is the same for all agents. As in a barrier phase, an agent in an interaction phase continues to update its step counter with each interaction.

The controlling protocol updates the state of the agent at the end of each round. Each agent v has a state $v.\mathsf{state}$ that is one of broadcasting (agent is initiating a broadcast of value 1), receiving (agent is waiting to detect a 1), received (agent has detected a 1), selecting (agent is attempting to select another agent), candidate (agent is a candidate for selection), selected (agent has been selected), or idle (agent has selected another agent and is now waiting for the end of the round). We assume the controlling protocol assigns consistent values to the agents in each phase: if one or more agents start in state broadcasting, the rest should start in state receiving; while if some agent starts in state selecting, the rest should start in state candidate. Pseudocode for the communication protocol is given in Algorithm 3.1. The correctness of this protocol relies on carefully chosen phase interval lengths which allow us to simulate synchronous rounds wherein the above operations are executed in sequence.

Proof of Correctness

An **epidemic** process in a population protocol starts with a single agent **infected** (i) and all others **susceptible** (s), and every encounter between an infected and uninfected agent causes the latter to become infected (i.e., the transition $i, s \to i, i$). Let $H_k = \sum_{i=1}^{k} \frac{1}{i} \sim \ln n$ be the k'th harmonic number. The following lemma, due to [MAS16], was proved in its current form in [BCC+21].

Lemma 3.3 (see [BCC+21], Lemma 2.7)**.** *Starting from a population of size n with a single infected agent, let $\boldsymbol{T_n}$ be the number of interactions until all agents are infected. Then $\mathbb{E}(\boldsymbol{T_n}) = (n-1)H_{n-1} \sim n \ln n$, and for $n \geq 8$ and $\delta \geq 0$,*

$$\Pr\left(\boldsymbol{T_n} > (1+\delta)\mathbb{E}(\boldsymbol{T_n})\right) \leq 2.5 \ln(n) \cdot n^{-2\delta}$$

We now use this lemma to prove the correctness of Algorithm 3.1: Let us define $s_r = \sum_{j=1}^{r-1} 5r^2$; this is the total length of all rounds up to but not including r.

Algorithm 3.1: Convergent broadcast (Agent v seeing message m)

```
$v.\mathsf{tick} \leftarrow v.\mathsf{tick} + 1$;
if $v.\mathit{tick} < 2r^2$ then
    // Barrier phase:  do nothing
else if $v.\mathit{tick} = 2r^2$ and $v.\mathit{state} = \mathit{broadcasting}$ then
    // End of barrier phase:  start epidemic
    $v.m \leftarrow 1$;
else if $v.\mathit{tick} = 5r^2$ then
    // End of interaction phase
    Update $v.\mathsf{state}$ according to controlling protocol;
    $r \leftarrow r + 1$;
    $v.\mathsf{tick} \leftarrow 0$;
else if $v.\mathit{tick} > 2r^2$ and $v.\mathit{state} = \mathit{receiving}$ and $m = 1$ then
    // Receive and propagate epidemic
    $v.\mathsf{state} \leftarrow \mathsf{received}$;
    $v.m \leftarrow 1$;
else if $v.\mathit{tick} = 3r^2$ and $v.\mathit{state} = \mathit{selecting}$ then
    // Attempt to select
    $v.m \leftarrow 1$;
else if $v.\mathit{tick} > 3r^2$ and $v.\mathit{state} = \mathit{selecting}$ and $m = 0$ then
    // Selected a candidate
    $v.m \leftarrow 0$;
    $v.\mathsf{state} \leftarrow \mathsf{idle}$;
else if $v.\mathit{tick} > 2r^2$ and $v.\mathit{state} = \mathit{candidate}$ and $m = 1$ then
    // We are the selected candidate
    $v.\mathsf{state} \leftarrow \mathsf{selected}$;
```

Observe that $s_r = \Theta(r^3)$. Consider the midpoint $a_r = s_r + r^2$ of the barrier phase of round r. Let A_{ir} (not to be confused with the notation for an agent A_i) be the event $|\boldsymbol{S_i^t} - a_r| > r^2 - 1$, where $t = (n/2)a_r$ so that $\mathbb{E}(\boldsymbol{S_i^t}) = a_r$. Then the Chernoff bound gives $\Pr(A_{ir}) < 2e^{-a_r((r^2-1)/a_r)^2/3} = e^{-\Theta(r)}$. Similarly define $b_r = s_r + 3r^2$ and $c_r = s_r + 4r^2$ as the steps 1/3 and 2/3 of the way through the interaction phase of round r, and define B_{ir} as the event $|\boldsymbol{S_i^t} - b_r| > r^2 - 1$ when $t = (n/2)b_r$ and C_{ir} as the event $|\boldsymbol{S_i^t} - c_r| > r^2 - 1$ when $t = (n/2)c_r$. Then we also have $\Pr(B_{ir}) = e^{-\Theta(r)}$ and $\Pr(C_{ir}) = e^{-\Theta(r)}$.

Finally, define D_{ir} as the event that the schedule of interactions is such that an epidemic that has infected agent A_i after $(n/2)b_r$ steps has not infected all agents after $(n/2)c_r$ steps. Note that this definition does not depend on whether an actual epidemic is in progress after $(n/2)b_r$ steps; instead, we consider a hypothetical epidemic starting at A_i running on the same schedule. From Lemma 3.3, we have that for any two-way epidemic on n processes, the expected value $\mathbb{E}(\boldsymbol{T_n})$ of the number of interactions to infect all agents is $O(n \log n)$ and $\Pr(\boldsymbol{T_n} > (1+\delta)\mathbb{E}(\boldsymbol{T_n})) \leq 2.5 \ln(n) \cdot n^{-2\delta}$. For D_{ir} to occur, we need $\boldsymbol{T_n} > r^2$, giving $\delta = r^2/O(n \log n) - 1$ and thus $\Pr(D_{ir}) = e^{-\Omega(r^2/n \log n)} \ln n = e^{-\Omega(r^2/n \log n)}$.

Call a round r **safe** if none of the events A_{ir}, B_{ir}, C_{ir}, or D_{ir} occur for any $i \in \{1, \ldots, n\}$. These events are not even remotely independent, but the the union bound still applies, giving a probability that round r is not safe of at most $3ne^{-\Omega(r)} + ne^{-\Omega(r^2/n \log n)} = e^{\Omega(\log n - r)} + e^{\Omega(\log n - r^2/n \log n)}$. The sum of of these bounds over all rounds converges to a finite value for any fixed n, so by the Borel-Cantelli lemma, with probability 1 all but finitely many rounds are safe.

The following lemmas demonstrate that the protocol does what it is supposed to, once we reach the suffix of the execution containing only safe rounds. We start by excluding false positive broadcasts.

Lemma 3.4. *If rounds r and $r+1$ are both safe, then no process observes a 1-message in round r unless some process initiates a broadcast or selection in round r.*

Proof. If rounds r and $r+1$ are both safe, then the events A_{ir} and $A_{i,r+1}$ do not occur for any A_i. In particular, this means that at time $t = (n/2)a_r$, all agents have an internal clock $\boldsymbol{S_i^t}$ that is within the interval $a_r \pm r^2 - 1$, which lies within the barrier phase for round r. So at this time all agents display message 0, have completed the interaction phase for round $r-1$, and have not yet started the interaction phase for round r. A similar constraint holds at time $t' = (n/2)a_{r+1}$. It follows that any 1-message observed by an agent during its round-r interaction phase must result from some process setting its message to 1 either because it initiated an epidemic or selection during its own round-r interaction phase, or because it is propagating an epidemic initiated by such a process. □

Similarly, a safe round has no false negative broadcasts:

Lemma 3.5. *If round r is safe and all agents start round r in either a* broadcasting *or* receiving *state, then any epidemic initiated in round r is observed by all agents.*

Proof. Because B_{ir} does not occur for any A_i, after $(n/2)b_r$ steps, all agents are in their round-r interaction phase, and because C_{ir} does not occur for any A_i, all agents remain in their round-r interaction phase until at least $(n/2)c_r$ steps. If some agent A_i initiates an epidemic in round r, then $A_i.\mathsf{state} = \mathsf{broadcasting}$ after $(n/2)b_r$ steps, and under the assumptions of the lemma, every other agent is either in the broadcasting, receiving, or received state. A simple induction shows that the set of infected agents in the real process throughout the $[(n/2)b_r, (n/2)c_r]$ interval is bounded below by the set of infected agents in the hypothetical epidemic considered in the definition of D_{ir}. This means that if D_{ir} does not occur, both such sets contain all processes after $(n/2)c_r$ interactions. □

And finally, a safe round allows selection. Selection is not necessarily uniform conditioned on safety, but each agent has an $\Omega(1/n)$ chance of being selected when r is sufficiently large:

Lemma 3.6. *If round r is safe, exactly one agent A_i starts round r in a* selecting *state, and all other agents start round r in a* candidate *state, then exactly one agent finishes round r in a* selected *state. For each agent A_j, the probability p that it is chosen conditioned on the safety of round r and the events of previous rounds is at least $\frac{1}{2n}$ for sufficiently large r.*

Proof. Use the non-occurrence of any B_{ir} or C_{ir} to argue that agent A_i reaches tick $3r^2$ while all agents are in the interaction phase. Then the next interaction between A_i and any A_j causes A_j to observe a 1 and switch to a selected state.

We would like to argue that the next interaction between A_i and another agent A_j chooses each A_j with independent probability $1/n$. Unfortunately, we are conditioning on safety of round r. Let A be the event that A_i selects A_j and B the event that round r is unsafe. Then $\Pr(A|\neg B) = \frac{\Pr(A \wedge \neg B)}{\Pr(\neg B)} \geq \Pr(A \wedge \neg B) = \Pr(A) - \Pr(A \wedge B) \geq \Pr(A) - \Pr(B) = 1/(n-1) - \left(e^{-\Omega(r)} + e^{-\Omega(r^2/n \log n)}\right) \geq \frac{1}{2n}$ for sufficiently large r. □

3.5.3 Convergent Computation of Symmetric Functions

Early rounds produce incorrect results, so we need an error-recovery mechanism. We describe a basic protocol for electing a leader and having it gather inputs from the other agents. This allows the leader to compute the output of an arbitrary symmetric function and broadcast it to the other agents. The protocol guarantees termination with probability 1 even in executions where some of the rounds exhibit errors in the underlying broadcast mechanism. By restarting the protocol when it terminates, we guarantee that the protocol eventually runs without errors, thus converging to the correct output.

Each agent v has a Boolean field $v.\mathsf{leader}$ that marks it as a leader (or candidate leader) and a field $v.\mathsf{processed}$ that marks whether it has reported its input $v.\mathsf{input}$ to the leader. Agents cycle through 7 rounds, where the round number is $r \bmod 7$, organized as follows:

Round 0 Any leader broadcasts 1. A non-leader that receives 0 sets its leader bit. This round allows recovery from states with no leaders.

Round 1 Any leader broadcasts 1 with probability 1/2. A leader that does not broadcast but receives a 1 clears its leader bit.

Round 2 Any agent that cleared its leader bit in the previous round broadcasts 1. This causes any remaining leaders that receive a 1 to restart the information-gathering protocol and causes any non-leaders that receive a 1 to clear their processed bits. Broadcasting a 1 in this round is also used by the leader to restart the protocol after completion.

Round 3 Any agent v with $v.\mathsf{processed} = 1$ broadcasts 1. This is used by the leader and other agents to detect unprocessed inputs.

Round 4 If a leader received a 1 in the previous round, and there is no transmission in progress from a non-leader agent, the leader executes a selection operation. The selected agent sets its processed bit and transmits its input if its processed bit is not already set. If the processed bit is set, the agent transmits nothing in the following two rounds.

Rounds 5 and 6 These are used to transmit either (a) one bit of a selected agent's input, or (b) one bit of the protocol output. In either case the bit is encoded as two bits using the convention $01 = 0$, $10 = 1$, $00 = \mathsf{stop}$. Note that the absence of a broadcast in both rounds is interpreted as stop, which both allows a selected agent to signal it has already been processed and guarantees eventual termination after an agent finishes transmitting its input even if some of the broadcasts are garbled. Two agents may transmit simultaneously (e.g. if there are multiple surviving leaders), thus agents must be prepared to handle receiving 11. Either we can decide to have agents interpret it as a fixed value: $11 = 1$, or interpret it as a signal to restart the protocol by broadcasting a 1 in the next Round 2.

The protocol terminates when the leader has collected all inputs (detected by the absence of a signal in Round 3) and transmits the computed output to all agents (using Rounds 5 and 6 over however many iterations are needed). We assume that the computed output has finite length for any combination of inputs. After transmitting the output, the leader broadcasts a 1 in Round 2 to restart the information-gathering component of the protocol.

Lemma 3.7. *In any execution with finite errors in the underlying broadcast protocol, with probability* 1*, the above protocol converges to a single leader and then restarts infinitely often.*

Lemma 3.7 is proven correct by demonstrating that, as defined, this protocol elects exactly one leader by Round 2 which correctly processes all non-leader agents with probability 1 by the end of Round 6.

Proof. Consider a sequence of iterations in which no errors occur.

If there are no leaders initially, the first execution of Round 0 sets the leader bit in all agents. The only way that an agent can lose its leader bit is if it sees another leader broadcast a 1-message in Round 1, but this always leaves at least one leader. If there is more than one leader, then half of the remaining leaders on average will drop out in each execution of Round 1. This guarantees that there will eventually be exactly one leader with probability 1.

If there is a leader, the leader believes that there is no transmission in progress, and at least one agent v with $v.\mathsf{processed} = \textsc{False}$, then v is selected with probability at least $\frac{1}{2n}$ for sufficiently large r in Round 3 (Lemma 3.6). This causes some agent to be selected in Round 3 eventually with probability 1, reducing the number of unprocessed agents by one.

If there is a transmission in progress, each transmitting agent sends finitely many bits before stopping. Once all transmitting agents have stopped, any agent waiting for a transmission to finish will observe 00 in Rounds 5 and 6.

It follows that starting from an arbitrary initial configuration, with probability 1 the protocol reaches a configuration with exactly one leader, the leader finishes waiting for any outstanding transmissions, and the leader then selects an unprocessed agent and collects its input until no unprocessed agents are left. After the leader transmits its computed (though possibly incorrect) output, the protocol restarts. □

Once the protocol restarts with a single leader, any subsequent error-free execution produces correct output. This follows from the fact that the leader collects the input from every agent exactly once. Since each agent records as its output the last output broadcast by the leader, this causes all agents to converge to holding the correct output with probability 1.

Because the leader has unbounded states, it can simulate an arbitrary Turing machine. This allows the output to converge to the value of any computable symmetric function. The restriction to symmetric functions follows from uniformity of the agents in the initial configuration, but can be overcome, assuming inputs include indexes. We have thus shown:

Theorem 3.1. *For any computable symmetric function f, there is a population protocol using* 1*-bit messages and unbounded internal states that starts in an initial configuration where each agent A_i is distinguished only by its input x_i, that converges to having each agent holding output $f(x_1, \ldots, x_n)$.*

Our construction exploits the unbounded state at each agent to allow the leader to simulate the entire computation. While probability-1 convergence requires unbounded state in the limit (otherwise there is a nonzero probability that any round fails), it may be desirable to put off expanding the state as long as possible. In the following section, we argue that with some small tweaks, the construction can be adapted to distribute the contents of a Turing machine tape of s bits across all agents of the population as in [CMN$^+$11], reducing the storage overhead at each agent for the Turing machine computation to $O(s/n + \log s)$ bits.

3.5.4 Simulating a Turing Machine

In this section, we show how to adapt the construction of Section 3.5.3 to simulate a Turing machine directly. For the most part, we retain the round structure of the previous construction, but make some adjustments to how the leader interacts with the other agents.

As in the construction in Section 3.5.3, we elect a leader using Rounds 0 and 1, which resets the other agents by broadcasting in Round 2. Non-leader agents reset to an **unallocated** state in which they hold only their input while waiting to be recruited to hold tape cells; such agents will set processed to FALSE until recruited to hold a tape cell. The leader agent holds the state of the finite-state controller and the index for the current head position and manages communication with the other agents through Round 5 and 6 broadcasts. Using the same self-delimiting encoding as before allows transmission of messages of arbitrary length, so long as all agents agree on which agent's turn it is to speak.

A very high level overview of the simulation is given in Algorithms 3.2 and 3.3. Algorithm 3.2 is written from the perspective of the leader, and assumes that we have already elected a unique leader and reset all the other agents. The function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$ is the transition function for the simulated Turing machine. Algorithm 3.3 is written from the perspective of a non-leader and describes how it responds to transmissions from the leader.

The simulation starts by organizing the agents into a Turing machine tape. This involves selecting agents one at a time and assigning them indices. Because an agent might be selected more than once, the expected number of rounds to find all agents scales as $O(n \log^2 n)$, where $O(n \log n)$ comes from the expected time to finish a coupon collector process and the extra $O(\log n)$ comes from the time to transmit indexes one bit at a time. We assume that counting n is also enough for the leader to compute a bound $f(n)$ on the number of steps used by the Turing machine; this is needed to enforce restarts if the simulated machine does not terminate on its own.

For simplicity we assume that inputs can be placed in arbitrary order on the first $n - 1$ cells of the tape (this will require special handling of any input on the leader,

Algorithm 3.2: TM Simulation: Leader Code

```
$h \leftarrow 0$;                                          // initial head position
$q \leftarrow q_0$;                                        // initial state
$n \leftarrow 1$;                                          // count of agents
counted $\leftarrow$ FALSE;                                // indicates if $n$ is correct
Restart computation by sending 1 in Round 2;
// Initialize tape
while counted $=$ FALSE do
    if at least one agent reported processed $=$ FALSE in Round 3 then
        Select an agent in Round 4;
        Transmit Recruit$(n+1)$;
        if some agent responds then
            $n \leftarrow n+1$;
    else
        Transmit PopulationSize$(n)$;                      // no agents remaining!
        counted $\leftarrow$ TRUE;
        $s \leftarrow f(n)$;                               // initialize run-time bound
for each Turing machine step do
    Read cell at current head position $h$ by transmitting Read$(h)$;
    if some agent responds with $c$ then
        $(q', c', d) \leftarrow \delta(q, c)$;
        Write $c'$ to cell $h$ by transmitting Write$(h, c')$;
        $q \leftarrow q'$;
        $h \leftarrow h + d$;
        $s \leftarrow s - 1$;
        if $q$ is a halting state then
            Transmit result of computation;
            Jump to start of algorithm;
        else if $s = 0$ then
            Jump to start of algorithm;                    // run-time bound exceeded
    else
        Jump to start of algorithm;                        // no agent holds $h$, init failed
```

which we omit for simplicity of presentation). A complication is that inputs to the protocol might exceed the size of the constant tape alphabet. This does not affect the simulation directly, since no restriction on tape alphabet is assumed, but it may require adding a preamble to the Turing machine computation that unpacks large-alphabet inputs into the constant-size TM alphabet. We leave the details of this

Algorithm 3.3: TM Simulation: Follower Code

```
// Initialization at restart
processed ← False;                                  // I am unallocated
$j \leftarrow \bot$;                                // $j$ indexes which cells I hold
// We use the convention that $x \bmod \infty = x$ and $x/\infty = 0$
$n \leftarrow \infty$;                              // $n$ is initially unknown
// $T[i]$ holds cell $(n-1)i + c$
Clear list $T[]$;                                   // unset locations default to blank
Set $T[0]$ to my input;
upon receiving Read($i$)
    if $i \bmod (n-1) = j$ then
        Transmit $T[\lfloor i/n \rfloor]$;
upon receiving Write($i, c$)
    if $i \bmod (n-1) = j$ then
        $T[\lfloor i/n \rfloor] \leftarrow c$;
upon receiving Recruit($h$)
    if I have been selected then
        if $j = \bot$ then
            $j = h$;
            Transmit acknowledgment;
    Clear selected status;
upon receiving PopulationSize($m$)
    $n \leftarrow m$;
upon receiving result of computation
    Record result as output;
```

tedious and unenlightening preamble to the imagination of the reader.

The analysis of Algorithms 3.2 and 3.3 essentially follows the proof of Theorem 3.1. Once the simulation reaches the safe phase of the construction, it reaches a configuration with one leader after some finite time with probability 1. At this point the leader may already have an inaccurate estimate $\hat{n}$ of n, but whether the estimate is accurate or not, each iteration of the main loop will require at most $O(\log f(\hat{n}))$ rounds to finish, leading to a restart after at most $O(n \log^2 n + f(\hat{n}) \log f(\hat{n}))$ rounds on average. Each subsequent iteration will run the Turing machine to completion and produce the correct output. The space complexity at each agent, measured in bits, is bounded by $O(s/n + \log s)$ during non-faulty simulations, where s is the largest tape cell index used. For faulty executions, we accept a small probability that a larger estimate of $\hat{n}$ at some leader agent may lead to larger space overhead. In either case the state complexity is dominated in the limit by the unbounded round and tick counters.

The final result is that we can use our synchronous broadcast primitive to stably simulate a Turing machine by having the leader orchestrate the computation using individual agent states to store the contents of the machine's tape cells:

Theorem 3.2. *Algorithms 3.2 and 3.3 use the synchronous broadcast primitive to simulate a Turing machine with known time complexity $f(n)$, converging to the correct output with probability 1. In any execution, the additional space required at each agent to simulate a Turing machine that uses s tape cells is bounded after an initial prefix by $O(s/n + \log s)$ with probability 1.*

Chapter 4

Input Privacy

While the original population protocol model considers the concept of anonymity, the issue of privacy is not investigated thoroughly. In this chapter, we will assess and refine the definitions of privacy in the population protocol model. We introduce several formal definitions of privacy, ranging from assuring only plausible deniability of the population input vector to having a full information-theoretic guarantee that the probability of a particular input vector conditioned on an agent's input and the protocol output is unchanged by further conditioning on any additional information that is accessible to the agent. We show in Theorem 4.1 that the strongest definition among these is that of information-theoretic privacy.

Then, we apply these definitions to both existing and novel protocols. In Theorem 4.2, we show that the `Remainder`-computing protocol from [DGFGR07] (which is proven to satisfy output independent privacy under adversarial scheduling) is not information-theoretically private under probabilistic scheduling. In contrast, Theorems 4.4, 4.5, and 4.6 show that our new protocol correctly and information-theoretically privately computes `Remainder` under probabilistic scheduling.

The results of this chapter were produced in collaboration with James Aspnes.

4.1 Introduction

Various issues arise when trying to apply the theoretical population protocol model to real-world systems. One of the most critical of these issues is preserving privacy in networks which gather and transmit sensitive data. The motivation for furthering the study of privacy within population protocols is to better adapt these algorithms to the real-world systems that they aim to model, such as sensor networks, systems of IoT devices, and swarms of drones, all of which harbor sensitive data. Previous research in private population protocols only considers adversarial scheduling, which makes very generous assumptions about our obliviousness to the scheduler's interaction choices and offers only very weak criteria for satisfying the definition of "privacy." In this work, we aim to further refine these definitions, bearing in mind a realistic range of threat models and security concerns.

Private-input predicate computation in population protocols was first introduced by Delporte-Gallet et al. in their 2007 work titled "Secretive Birds: Privacy in Population Protocols" [DGFGR07]. The mention of birds in the title stems from the application of the theoretical population model given in [AAD$^+$06], which considered a network of sensors affixed to birds (illustrating a network of "agents" communicating in unpredictable patterns, particularly when two birds flew close enough together for their sensors to transmit data). In [DGFGR07], privacy in population protocols was studied with adversarial scheduling, which (unlike probabilistic scheduling) has no concept of parallel running time or even bounds on the number of steps expected to take to converge to an output. If we know that the scheduler is actually random, then there is also the issue of data leakage that arises when interaction patterns can be inferred (or at least inferred with some probability) from assumptions about how the scheduler operates.

4.2 Related Work

The body of literature regarding private computation in ad hoc networks is distributed (pun intended) over multiple academic fields. We will limit our review of the literature to the works that most closely relate to the theoretical model we aim to study in this work.

4.2.1 Population Protocols

The problem of private-input computation was introduced and solved in the original population protocol model by Delporte-Gallet et al. in [DGFGR07]. The scheduler was assumed to be adversarial and protocols were restricted to $O(1)$ states. It is well known that under these circumstance, all and only the **semilinear predicates**, or predicates which are definable in first-order Presburger arithmetic, are stably computable in the population protocol model [AAE06b]. The semilinear predicates can be expressed using only `Threshold`, `Remainder`, `Or`, and `Not`. The latter predicate, which is defined as the negation of a Boolean expression, can always be computed by simply negating the protocol output, or any intermediary computation therein. The rest of these predicates are defined formally for population protocols as follows:

Definition 4.1. *Given integer constant T and input vector $I \in \mathbb{Z}^n$, let*

$$\texttt{Threshold}(I) = \begin{cases} \textsc{True} & \textit{if } \sum_{j=1}^{n} i_j > T \\ \textsc{False} & \textit{otherwise} \end{cases}$$

Definition 4.2. *Given positive integers k and n, non-negative integer $r < k$, and input vector $I \in \mathbb{Z}_k^n$, let*

$$\texttt{Remainder}(I) = \begin{cases} \textsc{True} & \textit{if } \sum_{j=1}^{n} i_j \equiv r \pmod{k} \\ \textsc{False} & \textit{otherwise} \end{cases}$$

Definition 4.3. *Given input vector $I \in \{\textsc{True}, \textsc{False}\}^n$, let*

$$\texttt{Or}(I) = \begin{cases} \textsc{True} & \textit{if } i_1 \vee i_2 \vee ... \vee i_n \\ \textsc{False} & \textit{otherwise} \end{cases}$$

Thus, showing that these three predicates can be computed without revealing any information about agents' individual input values is sufficient to show that all semilinear predicates can be computed without revealing this information, as well.

The authors of [DGFGR07] define a notion of privacy called *output independent privacy* and proved by construction that there exist protocols satisfying this definition when computing predicates `Remainder`, `Threshold`, and `Or`. Output independent privacy basically states that for any given execution yielding a particular output at a given honest but curious agent A, there exists an execution for another input vector (yielding the same output at A) where A has the same "view" as it does in the given execution. While the work in [DGFGR07] assumed adversarial scheduling and $O(1)$ state complexity, a more complex strategy may be needed in order to achieve a similar level of privacy in a setting with probabilistic scheduling or agents having $\Omega(1)$ states.

Furthermore, due to adversarial scheduling, the *existence* of an execution is sufficient to achieve plausible deniability of one's input because there is no probability distribution imposed on the scheduling; therefore, agents have no metric for estimating time elapsed nor approximating how many interactions in which another agent has participated. As a consequence, the observed states of agents cannot be used to infer those agents' inputs as they may have deviated drastically from their original states over the course of many interactions.

4.2.2 Sensor Networks

Population protocols are designed to model sensor networks, but there is a large body of literature on sensor networks that is not connected to the population protocol model. In this section, we give an overview of some of this work and describe why it is not directly applicable to our research problem.

By and large, the presumed capacity of agents in the general sensor network domain is much larger than is assumed in the population protocol model. This is evidenced by the fact that much of the privacy-preserving algorithms in this area involve encryption, which requires at least linear state space in the size of the population, not to mention auxiliary processes like key management.

In recent years, viral exposure notification via Bluetooth has become a popular area of study [CFG$^+$20, CKL$^+$20], and one that demands verifiable privacy guarantees due to widespread laws governing protected health data. However, the solutions in [CFG$^+$20, CKL$^+$20] are based on storage of exposures in a centralized database, which necessitates a high storage capacity implementation.

A similar problem called anonymous source detection (i.e. anonymously alerting an entire network of agents to the presence of a special source state in the system) is studied in [ADG$^+$05] and [BC18]. In effect, these algorithms "silently" notify all agents that the source state has been detected without identifying the agent with the source state. However, these solutions both require superconstant state space and are only able to perform this one task (which is essentially equivalent to a private `Or` computation).

Other research in wireless sensor networks investigates the problem of data aggregation while preserving the privacy of individual devices, which most closely resembles the goal of our research [CMT05, TG09, LLZC13]. As before, these works require high computation and local memory as they implement their solutions using homomorphic encryption primitives. Where alternative methods are used in order to avoid relying on encryption, a specialized network topology is needed for success [STE19] or only very specific functions are computable [TG09].

There do exist low-storage privacy-preserving protocols in the literature, such as [DGGK09]. This paper achieves information-theoretically secure *secret sharing*, but requires a centralized trusted party to compute and securely disseminate shares (i.e. the share algorithm is not distributed). It also does not discuss distributed aggregation of the agents' data.

While far from comprehensive, this sample of related works in the domain of wireless sensor networks suggests that much of the research in this area is limited by network topology or relies on computationally intensive encryption solutions. This is why we are interested in studying the problem of privacy with respect to data aggregation in decentralized settings, namely in ad hoc mobile networks modeled by population protocols, in a more theoretical sense. Our main goal is to find privacy-preserving solutions to data aggregation in this model without overlaying encryption so as to minimize local storage at each limited-resource device.

4.3 Contribution

In this work, we study the privacy of population protocols in the random scheduling model. We demonstrate how existing privacy definitions fail in the context of certain conditions or adversarial capabilities, give new precise definitions of desired security in these settings, and give and prove the security of a new protocol in the uniform random scheduling model satisfying the new definitions. In this work, we restrict our focus to computing the `Remainder` predicate. We leave the problem of privately computing `Threshold` and `Or` under probabilistic scheduling as open for future work.

4.4 Adversarial Model

Before defining a notion of input privacy in the population model, we must first consider what forms an adversary may take within this theoretical framework. There are various possibilities, which we discuss in detail below, that are motivated and justified by practical applications. Please note that the "adversary" in this context, which is different from the *adversarial scheduler* that determines the interaction pattern of the population, aims to learn information that is designated as sensitive or *private*. In this work, we will consider the agent inputs (from the input set Σ) to be private information.

4.4.1 Capabilities

In order to evaluate the extent to which private information can be learned by an adversary in a population protocol, we must define the nature of the adversary and its capabilities. We will consider the adversary to take the form of an agent interacting in the protocol, meaning that it can observe the population only as other agents do, i.e., by participating in interactions as they are slated by the scheduler. However, we do not preclude the possibility that the adversary may have greater computational capabilities than ordinary honest agents. We imagine that any agent may be construed as an "adversary," and by doing this, we demonstrate that no matter which agent it is or what strategy it employs, it cannot learn any sensitive information beyond that to which it already has access.

We assume that the adversary is **semi-honest**, meaning that it must adhere to the protocol rules exactly, but may try to infer additional knowledge from the system [Lin17]. As such, the adversary can only gather knowledge by interacting with other agents as prescribed by the transition function δ.

4.4.2 Limitations

Supposing that an adversary presents as an agent in the population, we can imagine that multiple adversaries may infiltrate the population. However, we restrict that each adversary be **non-colluding**, meaning that it cannot communicate with other adversarial nodes in the system beyond participating in the protocol interactions honestly.

This is because we could imagine that an adversary may disguise itself as multiple agents in the population making up any fraction of the system. If the adversary is dishonest, it may even perhaps perform denial of service attacks, causing certain agents to be effectively eliminated from the system entirely. Such an adversary could then, in theory, eliminate every agent in the system but one and then pretend to be every other agent in the population by flooding the network with "dummy" agents. In this scenario, the adversary and the one honest agent may perform a computation yielding some output, in some cases allowing the adversary to deterministically compute the one honest agent's input (e.g. if the computation is an `Or` of the population's inputs, and the adversary simulates all of its dummy agents to be of input 0).

Even if the adversary is honest, if it is allowed to collude then it can sometimes learn some information about honest agent inputs using a similar strategy: For example, suppose that the agents are computing the `Remainder` predicate. If the adversarial nodes can collude with each other, then they can remove their inputs from the final computation to obtain the remainder of the sum of the honest agent inputs. Although not studied within this work, it is of interest to find bounds on the fraction of agents that can be simulated by the adversary in any network and still successfully hide honest agents' inputs from the adversary when adversarial nodes *are* allowed to collude for more general computations.

Notice that the assumption that the adversary is non-colluding is equivalent to assuming that there is only a single adversarial agent in the population. This is because from the point of view of the adversary, all other agents appear to be honest because they must behave honestly and cannot share additional information beyond that which is allowed by the protocol (due to the semi-honesty assumption).

Finally, we allow a distinction between externally visible messages and internally hidden states as in Chapter 3 to allow agents to conceal a portion of their states toward the end goal of achieving privacy. The distinction between messages and the internal state will be crucial to studying privacy in the population model as without it, there is no mechanism for hiding information from an adversarial agent. Specifically, if an agent cannot conceal a portion of its state, then the agent's input can be reconstructed from its state by any other agent with which it interacts[1]).

4.5 Definitions of Input Privacy

Here we will assess what information may be collected from the adversary given the constraints detailed in the previous section. We will explore the previous definition of input privacy considered in the adversarial scheduling model [DGFGR07], and then devise a new definition in the random scheduling model.

In both definitions, the general problem specification and ultimate goal of the population are each the same: Each agent A_j in the population begins in some initial state in Q mapped to from their input i_j. There is some predicate Φ that we wish to compute over the agents' initial inputs (described by the input vector I), meaning that we want the entire population to eventually converge to a configuration where each agent is in a state in $Q^* = \{q_j : \mathcal{O}(q_j) = \Phi(I)\}$. Some protocols do not require that every agent learn the output of the protocol computation, but here we will assume that this is the final goal of the population.

4.5.1 Output Independent Privacy

It is well-known that all and only semilinear predicates are stably computable by population protocols in the original model (where agents are restricted to $O(1)$ states and interact in pairs chosen by an adversarial scheduler that obeys certain fairness conditions) [AAER07]. The privacy-preserving population protocol from [DGFGR07] operates in this model, and therefore demonstrates privacy in the context of comput-

[1] Or, its input can be reduced to a smaller subset of all possibilities if the input function is not injective.

ing semilinear predicates only. Such predicates can be converted to a form wherein only the operators `Threshold`, `Remainder`, and `Or` are needed.

At a high level, privacy means that no honest but curious agent in the population learns more about any other agent's initial input than it does from knowledge of its own input and the final computation of the protocol, similar to the definition of secure multiparty computation [Yao82]. The phrasing "honest but curious" does not imply that the process is not adversarial, but rather it just restricts that the process must obey the rules of the protocol, as deviations are presumed to be detectable by the truly honest agents. From [DGFGR07], we have a notion called **output independent privacy** which states the following:

> "A population protocol has this property if and only if there is a constant n_0 such that for any agent p and any inputs I_1 and I_2 of size at least n_0 in which p has the same input, and any execution E_1 on input I_1, and any T, there exists an execution E_2 on input I_2, such that the histories of p's interactions up to T are identical in E_1 and E_2."

Essentially, this definition states that any honest but curious process p cannot tell whether the input vector is I_1 or I_2 given its sequence of observed interactions because either input could have yielded the same observations.

4.5.2 Motivation for New Definitions

The definition of privacy given in Section 4.5.1 was successful in [DGFGR07] because the scheduling in this work is assumed to be adversarial; however, we argue that the protocols in [DGFGR07] are no longer private by this definition when randomized scheduling is used instead.

With adversarial scheduling, there is no inference that can be made about the interaction pattern whatsoever. If one agent interacts with another and views the entirety of their state, the observing process has no mechanism for detecting whether the other agent has been idle (not interacting) for an indefinite amount of time. During this time, it is possible that the other agent may have interacted frequently and therefore changed its state drastically from its initial value. In short, the protocol in [DGFGR07] leverages this to achieve privacy which is at best framed as "plausible deniability" – an agent may directly observe another agent's input, but the unpredictability of the adversarial scheduler disallows the observer to claim with certainty that the observed value is indeed the input.

This argument breaks down when the scheduler is probabilistic because now an agent can infer a probability distribution on the interaction pattern, and therefore also infer a probability distribution on the input value of the agent's interacting partner (see Section 4.6.1). In light of this insight, we now introduce novel definitions for the purpose of assessing privacy in population protocols with probabilistic scheduling.

4.5.3 Probabilistic Definitions of Privacy

As the population performs state updates according to δ, each agent observes the message of its interacting partner. This observed message, combined with the agent's entire state (both internal and message), gives the agent's new internal and message states. We cannot necessarily assume that an agent can infer the other agent's state changes as they may depend on that agent's internal state, which is not observable to its interacting partner.

Thus we can imagine that each agent has some set of observations which it can store locally and use to try to infer the input of other agents. Although we consider only protocols with constant state, we must allow for an adversarial agent, potentially having unbounded computational abilities, to infiltrate the population. Therefore, even though we do not expect an ordinary agent to be able to recall all of its previous interactions, we consider the case where we have an honest but curious adversarial agent in the population with unbounded memory such that this agent can store all such interactions and try to use them to infer other agents' inputs.

Consider one agent A with initial state $q_0^A = (s_0^A, m_0^A)$. Given the sequence of observed messages of its interacting partner and the role ρ ($\mathsf{Initiator}$ or $\mathsf{Responder}$) played by A in each interaction, we can deterministically compute each of its subsequent state updates. Let's call these messages (observed by A) $o_1^A, o_2^A, o_3^A, ...$, and denote by $q_\varepsilon^A = \delta(\rho_\varepsilon^A, s_{\varepsilon-1}^A, m_{\varepsilon-1}^A, o_\varepsilon^A) = (s_\varepsilon^A, m_\varepsilon^A)$ the updated state of A, originally in state $q_{\varepsilon-1}^A = (s_{\varepsilon-1}^A, m_{\varepsilon-1}^A)$, upon interacting as $\rho_\varepsilon^A \in \{\mathsf{Initiator}, \mathsf{Responder}\}$ with another agent with message o_ε^A in its ε-th interaction.

Adopting notation from [Lin17], we denote the **view** of an agent A participating in protocol $\mathcal{P}$ in an execution E by $\mathsf{view}_A^{\mathcal{P}}(E) = \langle i_A; q_0^A; (\rho_1^A, o_1^A), (\rho_2^A, o_2^A), ...\rangle$. This view consists of A's input, the initial state of A (i.e. $\mathcal{I}(i_A)$, which is computed in an execution according to the random tape of A), as well as a list of A's interactions over the course of the execution. Because an execution is technically an infinite sequence of configurations, we denote the view of A up until some fixed step τ as: $\mathsf{view}_A^{\mathcal{P}}(E)^\tau = \langle i_A; q_0^A; (\rho_1^A, o_1^A), (\rho_2^A, o_2^A), ..., (\rho_{\alpha_A}^A, o_{\alpha_A}^A)\rangle$, where α_A is the number of interactions A participates in from the start of the execution to a fixed step τ of the execution. The only information about interactions that A keeps track of in its view is a tuple of the role ρ_ε^A played by A in its i-th interaction and the observed message of its interacting partner in that interaction, since every subsequent state of A can be computed from this information.[2]

Via a mild abuse of notation, we will denote by $\boldsymbol{\mathsf{view}_A^{\mathcal{P}}(C)}$ a random variable representing the view of agent A drawn uniformly from all possible executions starting from configuration C (i.e. all realizable executions starting from C resulting from the possible randomness used by the scheduler). Similarly, we will denote by $\boldsymbol{\mathsf{view}_A^{\mathcal{P}}(I)}$ a

[2] If δ is a randomized function, then we assume A has a fixed tape of random bits that it uses to compute its updated state at each interaction; therefore, it can still reconstruct its entire view from the specified information.

random variable representing the view of agent A drawn from all possible executions starting from any configuration C in the range of $\mathcal{I}(I)$ according to the probability distribution given by the randomness of $\mathcal{I}$.

Privacy, like many other security-related key terms, has a wide range of technical interpretations. As such, we now offer several distinct formal definitions of privacy in the population model.

Plausible Deniability

Perhaps the weakest form of privacy we can possibly define is that of *plausible deniability*, meaning that an adversary always doubts its guess of an agent's input value (even if it has unbounded resources). This is not a novel concept [DGFGR07, MT19], but in the context of input vector privacy for probabilistic population protocols, we define this notion using the following terminology and notation:

Let $\mathcal{M}_\lambda = \{\text{multiset}(I) : \Phi(I) = \lambda\}$ be the set of all distinct multisets of inputs whose corresponding input vector evaluates to λ.[3] Additionally, let $\mathcal{M}_\lambda^\kappa = \{\text{multiset}(I) : \text{multiset}(I) \in \mathcal{M}_\lambda \wedge \kappa \in \text{multiset}(I)\}$ be the set of all distinct multisets of inputs outputting λ which contain at least one input equal to κ.

Definition 4.4. *Let $\mathcal{P}$ be a population protocol on n agents with input set Σ and let $\mathcal{D}$ be any probability distribution on input vectors in Σ^n. Then $\mathcal{P}$ is* ***weakly private*** *if for every distribution $\mathcal{D}$ on Σ^n, every non-colluding semi-honest unbounded agent $\mathcal{A}$ in a population of size n executing $\mathcal{P}$, and for any view $V = \langle i; q; \{(\rho_\varepsilon^\mathcal{A}, o_\varepsilon^\mathcal{A})\}\rangle$ with output λ (as determined from the view V)*[4] *and with $|\mathcal{M}_\lambda^i| > 1$, there exist I_1 and I_2 in $\mathcal{S}_\lambda$ such that*

1. *both* multiset(I_1) *and* multiset(I_2) *are elements of $\mathcal{M}_\lambda^i$,*

[3] Recall that two agents in the same state are indistinguishable by the protocol; therefore, Φ must map any input vectors with the same multiset of inputs to the same output.

[4] Meaning that there exists some T such that for all $t > T$ we have $\mathcal{O}(q_t^\mathcal{A}) = \lambda$.

2. $\text{multiset}(I_1) \neq \text{multiset}(I_2)$*, and*

3. $\Pr(\boldsymbol{view}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{I_1}) = V) = \Pr(\boldsymbol{view}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{I_2}) = V)$*,*

where the probabilities in the final condition are taken over $\mathcal{D}$, the randomness of $\mathcal{I}$, and the uniform randomness of the scheduler.

The condition $|\mathcal{M}^i_\lambda| > 1$ necessitates that weak privacy only needs to hold for inputs for which plausible deniability is even possible. For example, if the output of the computation for the `Or` predicate is 0, then there is only one possible input vector (and therefore exactly one multiset of inputs) that could have yielded this outcome. Because this multiset of inputs contains only one input value, namely 0, plausible deniability is not achievable for this input vector.

In plain English, Definition 4.4 says that any agent participating in the protocol cannot simply guess the "most likely" input vector to the protocol because for each such vector, if there exists some other input vector yielding the same output, then there exists a distinct input vector yielding the same views for that agent with the same probabilities. This definition differs from the notion of output independent privacy from [DGFGR07] in that it considers adversarial strategies for guessing the input vector which rely on distributional data collected from interactions with other agents.

Computational Indistinguishability

This definition borrows concepts from classical cryptography dealing with proofs of zero-knowledge [GMR85], which rely on what is called the **simulation paradigm**. Essentially, the idea is that there are two versions of a protocol: A "true" version of the protocol is executed exactly as expected; however, an "ideal" version of the protocol is one where agents do not have inputs, but instead choose their inputs randomly based only on the output of the protocol. If an observer, say some agent in the population, cannot tell the difference between these two scenarios, then it must be the case that it learns nothing more about the inputs of other agents than it learns from information to which it already has access (namely, its own input and the output of the computation) because inputs in the latter scenario are chosen totally randomly subject to the protocol output remaining fixed.

Loosely speaking, "indistinguishability" in the population model means that a computationally bounded agent participating in the protocol does not learn more about any other agent's secret input than it does from its own input and the protocol output due to the fact that it cannot distinguish between a "real" execution of the protocol and an "ideal" one.

In order to define this formally, we first introduce some additional terminology:

Definition 4.5. *A function f is* ***negligible*** *[KL14] if for every polynomial $p(\cdot)$ there exists an N such that for all $n > N$ it holds that $f(n) < p(n)$.*

In other words, we call f negligible if it grows more slowly than every polynomial function.

Definition 4.6. *Two distributions $\mathcal{X}$ and $\mathcal{Y}$ on a sample space Λ of size 2^w are* ***computationally indistinguishable*** *[KL14] if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$ with oracle access to a polynomial (in w) number of instances from one of these two distributions, there exists a negligible function ϵ such that*

$$|\Pr(\mathcal{A}^{\mathcal{X}(\cdot)}(w) = 0) - \Pr(\mathcal{A}^{\mathcal{Y}(\cdot)}(w) = 0)| \leq \epsilon(w)$$

That is to say, there does not exist a PPT algorithm which can reliably identify either of these two distributions when given access to only a polynomial number of samples from one of the distributions. If $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable, then we write $\mathcal{X} \equiv \mathcal{Y}$.

Now, we define a notion of privacy related to computational indistinguishability in the population model:

Definition 4.7. *A protocol $\mathcal{P}$ satisfies **input indistinguishability** if for every semi-honest non-colluding PPT agent $\mathcal{A}$ and any polynomial $\tau(n)$, there exists a simulator S such that for every possible output λ and every $I \in \mathcal{S}_\lambda$,*

$$\{S(n, i_{\mathcal{A}}, \lambda)\} \equiv \{\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{I})^{\boldsymbol{\tau(n)}} : \Phi(I) = \lambda \wedge i_{\mathcal{A}} \in I\}$$

In words, given only the computation output λ, a fixed agent $\mathcal{A}$ and their input $i_{\mathcal{A}}$, as well as the size of the population n, there exists a PPT simulator S that can compute some polynomial-length prefix of a $\tau(n)$-length view at $\mathcal{A}$ that is computationally indistinguishable from $\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{I})$ in a real execution (up to some fixed step $\tau(n)$), so long as this view, computed by S, is consistent with inputs n, $i_{\mathcal{A}}$, and λ. By the simulation paradigm, this definition implies that no matter what the prefix of the execution, $\mathcal{A}$ learns as much about the input vector as it learns from its own input $i_{\mathcal{A}}$, public knowledge (such as n and an approximation of time elapsed τ), and the output of the computation λ.

Note that the definition claims that the property of indistinguishability holds true *for each* input vector and is not just a property which holds true in the aggregate when averaged over the input vector space.

Information-Theoretic Privacy

A slightly stronger notion of privacy similarly concerns the view of each agent in isolation, but instead assumes a computationally unbounded adversary. Naturally, this prompts our next definition of information-theoretic input privacy.

Let us first introduce some notation: Let $\mathcal{P}$ be a population protocol with input set Σ and let $\mathcal{D}$ be a probability distribution on input vectors in Σ^n. Let $\boldsymbol{I} \sim \mathcal{D}$ be a random variable representing the input vector provided to the computation. Additionally, let $\boldsymbol{i_{\mathcal{A}}}$ and $\boldsymbol{\lambda_{\mathcal{A}}}$ be random variables representing the input and output at agent $\mathcal{A}$, and let $\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda})$ be a random variable representing the view of agent

$\mathcal{A}$ participating in an honest execution of $\mathcal{P}$ that is consistent with a fixed input i at $\mathcal{A}$ and observed output λ.

Definition 4.8. *Protocol $\mathcal{P}$ satisfies **information-theoretic input privacy** if for every non-colluding semi-honest unbounded agent $\mathcal{A}$ and every input $i \in \Sigma$, output $\lambda \in O$, view V, input vector $I \in \mathcal{S}_\lambda$, and distribution $\mathcal{D}$ on Σ^n,*

$$Pr(\boldsymbol{I} = I \mid \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) = Pr(\boldsymbol{I} = I \mid \boldsymbol{i}_{\mathcal{A}} = i, \boldsymbol{\lambda}_{\mathcal{A}} = \lambda),$$

where V is consistent with input i and output λ.

The above definition essentially states that conditioned on knowing one's own initial state and the output of the computation, the rest of the agent's view when participating in the protocol's computation gives no advantage in guessing the input vector (and more importantly, another agent's input).

4.5.4 Comparison of Definitions

It is clear that information-theoretic privacy is the strongest notion of privacy among the definitions presented above, as it presumes a computationally unbounded adversary and necessitates that the probability distribution of the agent states do not change (even negligibly) when conditioned on any particular view of an execution. Thus, if a protocol satisfies information-theoretic privacy, it must also satisfy both indistinguishability and weak privacy. Let us prove this claim formally with the following theorem:

Theorem 4.1. *If $\mathcal{P}$ is information-theoretically private, then $\mathcal{P}$ also satisfies both weak privacy and input indistinguishability.*

Proof. We prove this claim in two parts, in each one by showing the contrapositive.

i. *If $\mathcal{P}$ is not weakly private, then it is not information-theoretically private.*

Because $\mathcal{P}$ is not weakly private, there exists some adversary $\mathcal{A}$ with view $V = \langle i; q; \{(\rho_\varepsilon^{\mathcal{A}}, o_\varepsilon^{\mathcal{A}})\}\rangle$ and output λ such that $|\mathcal{M}_\lambda^i| > 1$, but there is no pair of input vectors $I_1, I_2 \in \mathcal{S}_\lambda^2$ such that all three conditions in Definition 4.4 hold. Thus for every pair of input vectors I_1 and I_2 with distinct multisets that are members of $\mathcal{M}_\lambda^i$ (i.e. they are consistent with the adversary's input i and the computation output λ), we have that

$$\Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{I_1}) = V) \neq \Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{I_2}) = V)$$

As a result, we can say that

$$\begin{aligned}\Pr(\boldsymbol{I} = I_1 \mid \textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) &= \frac{\Pr(\boldsymbol{I} = I_1 \wedge \textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)}{\Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\ &= \Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{I_1}) = V) / \Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \\ &\neq \Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{I_2}) = V) / \Pr(\textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \\ &= \Pr(\boldsymbol{I} = I_2 \mid \textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)\end{aligned}$$

Using this information, we will show that for some distribution $\mathcal{D}$ on Σ^n it is the case that

$$Pr(\boldsymbol{I} = I \mid \textsf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \neq Pr(\boldsymbol{I} = I \mid \boldsymbol{i_{\mathcal{A}}} = i, \boldsymbol{\lambda_{\mathcal{A}}} = \lambda)$$

for some $I \in \Sigma^n$, indicating that $\mathcal{P}$ is not information-theoretically private.

Consider the uniform distribution on Σ^n, which means that for all $I \in \Sigma^n$ $\Pr(\boldsymbol{I} = I) = 1/|\Sigma|^n$, so for any pair of input vectors I_1, I_2 in $\mathcal{M}_\lambda^i$,

$$\Pr(\boldsymbol{I} = I_1) = \Pr(\boldsymbol{I} = I_2)$$

Due to the fact that these inputs come from $\mathcal{M}_\lambda^i$, we can also say that

$$\Pr(\boldsymbol{I} = I_1 \mid \boldsymbol{i_{\mathcal{A}}} = i, \boldsymbol{\lambda_{\mathcal{A}}} = \lambda) = \Pr(\boldsymbol{I} = I_2 \mid \boldsymbol{i_{\mathcal{A}}} = i, \boldsymbol{\lambda_{\mathcal{A}}} = \lambda)$$

However, because

$$\Pr(\boldsymbol{I} = I_1 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \neq \Pr(\boldsymbol{I} = I_2 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)$$

for some I_1, I_2 in $\mathcal{M}^i_\lambda \subseteq \mathcal{S}_\lambda$, we must have that either

$$Pr(\boldsymbol{I} = I_1 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \neq Pr(\boldsymbol{I} = I_1 \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda)$$

or

$$Pr(\boldsymbol{I} = I_2 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \neq Pr(\boldsymbol{I} = I_2 \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda)$$

Otherwise, we would have to have

$$\begin{aligned} \Pr(\boldsymbol{I} = I_1 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) &= \Pr(\boldsymbol{I} = I_1 \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda) \\ &= \Pr(\boldsymbol{I} = I_2 \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda) \\ &= \Pr(\boldsymbol{I} = I_2 \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \end{aligned}$$

which would contradict our initial assumption that $\mathcal{P}$ is not weakly private. Thus, there exists some $I \in \mathcal{S}_\lambda$ and distribution on Σ^n for which

$$\Pr(\boldsymbol{I} = I \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \neq Pr(\boldsymbol{I} = I \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda)$$

so $\mathcal{P}$ must also not be information-theoretically private.

ii. *If $\mathcal{P}$ is not input indistinguishable, then it is not information-theoretically private.*

To make these two definitions comparable, let's assume that the adversary always has unbounded computational power. Even if this is the case, we show that information-theoretic privacy is still stronger than unbounded input indistinguishability (and so weakening the adversary as in the definition of computational indistinguishability yields the same relationship between these two definitions).

Let $\mathcal{V} = \Sigma \times Q \times \{\{\mathsf{Initiator}, \mathsf{Responder}\} \times M\}^{\tau(n)}$ (for some fixed τ that is polynomial in n) be the space of all possible views of an agent in an execution of $\mathcal{P}$ with $\tau(n)$ interactions in a view. Denote by $\mathcal{X}(I)$ the distribution $\{S(n, i, \lambda)\}$ over $\mathcal{V}$; and denote by $\mathcal{Y}(I)$ the distribution $\{\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{I})^{\boldsymbol{\tau(n)}} : \Phi(I) = \lambda \wedge i \in I\}$ over $\mathcal{V}$. By assumption, $\mathcal{P}$ is not input indistinguishable, so there exists some input vector I^* for which $\mathcal{X}(I^*) \not\equiv \mathcal{Y}(I^*)$, as in $\Pr(\mathcal{X}(I^*) = V) \neq \Pr(\mathcal{Y}(I^*) = V)$ for some $V \in \mathcal{V}$. Suppose this I^* has input i at $\mathcal{A}$ and output λ.

Because $\mathcal{Y}$ reflects the real distribution of views for an agent $\mathcal{A}$ with input i and output λ when I^* is the true input vector, we have

$$\Pr(\mathcal{Y} = V) = \Pr(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \wedge \boldsymbol{I} = I^*)$$

In addition, because $\mathcal{A}$ is computationally unbounded, it can simulate the protocol on every possible input vector using every possible random schedule (for at least $\tau(n)$ observations at $\mathcal{A}$) to compute the exact probability with which input vectors containing i and outputting λ yield view V, so

$$\Pr(\mathcal{X} = V) = \Pr(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \wedge \boldsymbol{i}_{\mathcal{A}} = i, \boldsymbol{\lambda}_{\mathcal{A}} = \lambda)$$

Further, since $i \in I^*$ and $\Phi(I^*) = \lambda$, we know that

$$\Pr(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{I} = I^*) \leq \Pr(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{i}_{\mathcal{A}} = i, \boldsymbol{\lambda}_{\mathcal{A}} = \lambda)$$

However, this must actually be a strict inequality because otherwise I^* would be the only input vector in $\mathcal{M}^i_\lambda$ which would imply

$$\Pr(\boldsymbol{i}_{\mathcal{A}} = i, \boldsymbol{\lambda}_{\mathcal{A}} = \lambda \wedge \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) = \Pr(\boldsymbol{I} = I^* \wedge \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathcal{A}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)$$

and therefore also $\Pr(\mathcal{X} = V) = \Pr(\mathcal{Y} = V)$ (contradicting our initial assumption).

Therefore, (taking $\varkappa$ to be the event that $\boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i$ and $\boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda$) we have

$$
\begin{aligned}
& \Pr(\boldsymbol{I} = I^* \mid \textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \\
= & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{I} = I^*) \Pr(\boldsymbol{I} = I^*)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
= & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{I} = I^*) \Pr(\boldsymbol{I} = I^* \wedge \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
= & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{I} = I^*) \Pr(\boldsymbol{I} = I^* \wedge \varkappa)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
= & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \boldsymbol{I} = I^*) \Pr(\boldsymbol{I} = I^* \mid \varkappa) \Pr(\varkappa)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
< & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \mid \varkappa) \Pr(\boldsymbol{I} = I^* \mid \varkappa) \Pr(\varkappa)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
= & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V \wedge \varkappa) \Pr(\boldsymbol{I} = I^* \mid \varkappa)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
\leq & \frac{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V) \Pr(\boldsymbol{I} = I^* \mid \varkappa)}{\Pr(\textsf{\textbf{\textit{view}}}^{\mathcal{P}}_{\boldsymbol{\mathcal{A}}}(\boldsymbol{i}, \boldsymbol{\lambda}) = V)} \\
= & \Pr(\boldsymbol{I} = I^* \mid \varkappa) \\
= & \Pr(\boldsymbol{I} = I^* \mid \boldsymbol{i}_{\boldsymbol{\mathcal{A}}} = i, \boldsymbol{\lambda}_{\boldsymbol{\mathcal{A}}} = \lambda)
\end{aligned}
$$

Thus we have shown that $\mathcal{P}$ must also not be information-theoretically private.

□

As shown above, information-theoretic privacy is the strongest notion among those presented in this section. However, it is unclear what is the relative strength of privacy between the notions of input indistinguishability and weak privacy. It would seem as though indistinguishability should be stronger than weak privacy as it requires that no input even need be known by the simulator to construct a view which an adversary can distinguish from a real execution, and weak privacy makes no such claim. However, weak privacy presumes a computationally unbounded adversary, and imposes a probability requirement on the distribution of views for *each* input vector in Σ^n. We leave the relationship between these two definitions as an open problem for future work.

4.6 Adversarial Schedule Privacy: Secretive Birds

Now we analyze an existing algorithm claiming to preserve the privacy of inputs in the population model under adversarial scheduling. In this section, we focus our attention to the problem of computing the `Remainder` predicate. Its definition is restated below for convenience.

Definition 4.2. *Given positive integers k and n, non-negative integer $r < k$, and input vector $I \in \mathbb{Z}_k^n$, let*

$$\texttt{Remainder}(I) = \begin{cases} \textsc{True} & \textit{if } \sum_{j=1}^{n} i_j \equiv r \pmod{k} \\ \textsc{False} & \textit{otherwise} \end{cases}$$

Note that any protocol computing the remainder itself can be translated into a protocol computing the `Remainder` predicate simply by taking the output s of the protocol and evaluating $s \overset{?}{\equiv} r \pmod{k}$.

4.6.1 Overview of `Remainder` Protocol

Consider the protocol given in [DGFGR07] for determining whether the remainder of the sum of agents' inputs modulo a constant k is equal to some fixed r, shown in Algorithm 4.1. We will refer to this protocol as SecretiveBirdsRemainder. Similar protocols are also offered in [DGFGR07] for computing `Threshold` and `Or` which, when composed, can compute every semilinear predicate. None of these protocols distinguish between internal state space and message space, so the entirety of each agent's state is seen by its interacting partner (i.e. the protocol is *open*).

In SecretiveBirdsRemainder, the states are tuples consisting of two components (v, f), where v is the value of the agent and f is a flag bit denoting whether or not the agent has decided its output yet. In essence, the protocol accumulates the total sum (modulo k) of all of the agents' inputs by transferring the values in "pieces" rather than aggregating them all at once upon interacting. As shown by transition (M1), the protocol subtracts 1 from one of the inputs, and adds it to the other input, maintaining the invariant that the sum of all the values in the population is the same at each step. Because all computations are done modulo k, transition (M1) could be repeated indefinitely. For this reason, transitions (M2) and (M3) handle the flag bit, ensuring that the (M1) transition occurs an unbounded but finite number of times.[5] Finally, (M4) collects all remaining values into one agent, and the remaining transitions place and propagate the decided value of the predicate in the place of v at each agent.

Intuition Behind Output Independent Privacy

The crux of the proof that SecretiveBirdsRemainder satisfies output independent privacy focuses on transition (M1). When an adversarial process p interacts with an honest agent A in state (v, f), p cannot know how close v is to A's original input value because, for sufficiently large population size n, we can construct multiple executions wherein A has value v upon interacting with p. For example, we could have some agent B with value k (or 0) transfer as many units to A as needed to get A's value to be v, and as long as p and B do not interact before p interacts with A, p's view is the same. In fact, to p this looks exactly the same as if A did not interact with B at all and instead started with the value v initially. Thus p may have the same "view" for any input A had to begin with, and we can construct some additional agent B which does not interact with p to manipulate A's value and still maintain

[5] As written, this protocol dictates that the partial transfer of agent values only occurs when the flag bit is 1; however, it does not specify whether to use (M1) versus (M2) when the Initiator has a 1-flag. It is thus presumed that there is some non-determinism in the protocol, which may either be decided by the agents themselves or the scheduler. This distinction is not addressed in the paper itself.

Algorithm 4.1: Output Independent Private `Remainder` [DGFGR07]

$$(v_1, 1), (v_2, 1) \to (v_1 + 1, 1), (v_2 - 1, 1) \tag{M1}$$

$$(*, 1), (*, *) \to (*, 0), (*, *) \tag{M2}$$

$$(*, 0), (*, 1) \to (*, 1), (*, 1) \tag{M3}$$

$$(v_1, 0), (v_2, 0) \to (v_1 + v_2, 0), (0, 0) \tag{M4}$$

$$(v_1, 0), (0, 0) \to (v_1, 0), (\perp_0, 0) \tag{M5}$$

$$(\perp_i, *), (*, 1) \to (0, 0), (*, 1) \tag{M6}$$

$$(r, 0), (\perp_i, 0) \to (r, 0), (\perp_1, 0) \tag{M7}$$

$$(v_1, 0), (\perp_i, 0) \to (v_1, 0), (\perp_0, 0), \text{ if } v_1 \neq r \tag{M8}$$

$$q_1, q_2 \to q_2, q_1 \tag{M9}$$

Protocol from [DGFGR07] computing the `Remainder` predicate $\sum_{i=1}^{n} m_0^i \overset{?}{\equiv} r \pmod{k}$, where m_0^i is the initial input value of agent A_i. Each agent's state is initially $(m_0^i, 1)$. Arithmetic is done modulo k, and $*$ is a wildcard that can match any value. The output values are $\{\perp_0, \perp_1\}$, denoting that the predicate is False or True, respectively. The protocol has converged when all agents but one are in a state of the form $(\perp_i, 0)$ for $i \in \{0, 1\}$.

the same overall output of the population on a particular predicate.

Intuition Behind Lack of Probabilistic Schedule Privacy

This definition does not successfully carry over to the random scheduling model because we can no longer construct *any* execution fooling the process p, as some such executions are of very low probability. For instance, the probability that agents A and B interact v' times in a row, during which time p does not interact with B at all, becomes small for large values of v'. This means that it is less probable that an agent's value will deviate from its original input value early on in the execution. We will now formally demonstrate why the protocol from Algorithm 4.1 does not satisfy the definitions of security given in Section 4.5.3.

4.6.2 Proof of Insecurity

We now formally prove that the SecretiveBirdsRemainder protocol does not preserve privacy in the probabilistic scheduling model.

Information-Theoretic Input Privacy

Theorem 4.2. SecretiveBirdsRemainder *is not information-theoretically input private under a uniform random scheduler.*

Proof. First, we observe that an agent can compute the probability with which it interacts with another agent in their initial state under the uniform random scheduler conditioned on the event that it is also that agent's first interaction. Fix an agent A_j and consider its first interaction. With probability $2/n$, agent A_j participates in an interaction at any particular step. Conditioned on this event at step t, let us compute the probability that this is the first interaction for *both* agents in the execution (meaning that neither agent has been selected to interact at any prior step). Under the uniform scheduler, this probability is equal to

$$\left(\frac{\#\text{ pairs not including the two agents}}{\#\text{ total possible pairs}}\right)^{t-1} = \left(\frac{\binom{n-2}{2}}{\binom{n}{2}}\right)^{t-1} = \left(\frac{n^2-5n+6}{n^2-n}\right)^{t-1}$$

Summing this over all possible steps at which an interaction with A_j can occur, we have that the probability that A_j's first interaction is with another agent which has also never interacted before (and is therefore still in their initial state) is

$$\sum_{t=1}^{\infty} \frac{2}{n}\left(\frac{n^2-5n+6}{n^2-n}\right)^{t-1} = \frac{n-1}{2n-3} > \frac{1}{2}$$

Thus, with probability at least $1/2$, an agent can view another agent's initial state. Although an agent cannot necessarily detect that this is the case, the fact that this happens with such a large probability changes the conditional probability that an observed input exists in the population conditioned on the view of an agent:

The probability that any particular agent sampled uniformly from the population has input σ is

$$\begin{aligned} p_\sigma &= \frac{1}{n}\sum_{j=1}^{n} \Pr(\boldsymbol{i_j} = \sigma) \\ &= \frac{1}{n}\sum_{I\in\Sigma^n}\left(\sum_{j=1}^{n}\Pr(\boldsymbol{i_j} = \sigma \mid \boldsymbol{I} = I)\right)\Pr(\boldsymbol{I} = I) \\ &= \frac{1}{n}\sum_{I\in\Sigma^n}(\text{count of } \sigma \text{ in } I)\Pr(\boldsymbol{I} = I) \end{aligned}$$

Given that $\sum_{\sigma\in\Sigma} p_\sigma = 1$, there must be some $\sigma \in \Sigma$ for which $p_\sigma \leq 1/|\Sigma|$, and for any non-trivial population protocol $|\Sigma| \geq 2$. Thus, given the strategy described above, the probability that an agent's first observed state is the actual input of its interacting partner is strictly greater than 1/2, and for some observed input in the input set, this is not equal to its *a priori* probability. As such, SecretiveBirdsRemainder is not information-theoretically private. □

The protocols for solving `Threshold` and `Or` from [DGFGR07] can similarly be proven to lack privacy. We omit these proofs for concision.

Input Indistinguishability and Weak Privacy

Notice that the SecretiveBirdsRemainder protocol does not attribute probabilities to the nondeterministic transitions in δ. For example, two agents in states $(0, 1)$ and $(1, 1)$ could take transition (M1) or transition (M2) from Algorithm 4.1; however, we have no probabilistic distribution from which to draw this decision. Therefore, it makes it more complicated to ascertain whether or not this algorithm satisfies these definitions of input privacy.

The notion of privacy presented in [DGFGR07] relies on there being an unbounded number of times that (M1) could be applied to any particular agent's state, thereby transforming the input vector without restriction. If the probability of transition

(M1) is too small, then agents' state values (while of the form $(*, 1)$) are confined to a random walk near their bona fide input values. Conversely, even if the probability of transition (M1) is large enough, the first few interactions still reveal the input values of each agent because the entirety of an agent's state is viewed by its interacting partner with non-negligible probability, as demonstrated previously in the evaluation of information-theoretic privacy. In either case, the view of an agent narrows the input vector space significantly, sometimes down to a single multiset of inputs.

For example, consider the case where $k = 2$. Then any view of $\mathcal{A}$ resulting from an execution wherein two agents besides $\mathcal{A}$ interact with one another can be obfuscated by supposing that the order of the agents in the interaction is flipped. However, in executions where the only such interactions involve one agent in state 0 and another agent in state 1, this actually does not yield a new multiset of inputs (but rather just swaps the inputs held by the two agents). In fact, the only way to obfuscate the input multiset would be to change both of the inputs of two agents with equal inputs, but this would necessarily have to change the view of $\mathcal{A}$. While this case is contrived and rare, it still precludes SecretiveBirdsRemainder from being even weakly private. This motivates the need for novel protocols satisfying these privacy definitions in the probabilistic scheduling population model.

4.7 Probabilistic Schedule Privacy

In this section, we introduce a novel algorithm for computing `Remainder` in the population model with probabilistic scheduling. In fact, we demonstrate how to achieve the strongest notion of privacy proposed in Section 4.5.

Our algorithm is inspired by the famous introductory example of cryptographically secure multiparty computation of `Remainder` in a ring network. We refer to this algorithm as RingRemainder, and it works as follows:

RingRemainder: Information-Theoretically Secure `Remainder` on a Ring

There are n agents arranged in a circle. For ease of notation, we will select a random agent as our starting point and label the agents $A_1, ..., A_n$ clockwise. Agent A_1 performs the leader's role, which is to add a uniformly random element $r \in \mathbb{Z}_k$ to their input (and reduce the result modulo k). Agent A_1 then passes the sum to agent A_2. For each remaining agent A_i, upon receiving a value from A_{i-1}, A_i adds its own input to that value and passes the resulting sum to $A_{i+1 \pmod{n}}$. When A_1 receives a value from A_n, it subtracts r and broadcasts the result to everyone.

Suppose that the agents each have an input value $i_1, ..., i_n$. Then A_1 sends $m_1 = i_1 + r \pmod{k}$ to A_2, A_2 sends $m_2 = i_1 + r + i_2 \pmod{k}$ to A_3, and so on, until A_n sends $m_n = r + \sum_{j=1}^{n} i_j \pmod{k}$ to A_1. Thus, the value broadcast to all agents $m_n - r \pmod{k}$ is exactly equal to $\sum_{j=1}^{n} i_j \pmod{k}$, the remainder of the sum of the agents' inputs modulo k.

This algorithm requires honest participants because there is no mechanism enforcing the authenticity of the values incorporated into the aggregate. Additionally, it needs secure peer-to-peer communication between agents as without it, any intermediate results can be used to recover an agent's original input. For example, if an adversary $\mathcal{A}$ can eavesdrop on any of the pairwise communication channels, it can simply listen for the values sent from A_{j-1} to A_j and from A_j to A_{j+1} to obtain the input i_j (it simply takes the difference between these two values to get $m_j - m_{j-1} = (m_{j-1} + i_j) - m_{j-1} = i_j$). However, with secure pairwise communication channels, this protocol achieves information-theoretic privacy:

Theorem 4.3. RingRemainder *is information-theoretically private.*

Proof. Let $\boldsymbol{I} = \langle \boldsymbol{i_1}, ..., \boldsymbol{i_n} \rangle$ be a random variable representing the input vector to the protocol and let $\boldsymbol{m_j}$ be a random variable representing the message in $\mathbb{Z}_k$ passed from agent A_j to A_{j+1}.

Assuming secure peer-to-peer communication and non-colluding participants, each agent A_j only learns its own input, its received message, and the final value broadcast by the leader (which exactly equals the output of the computation). Thus the view of A_j consists of three values in $\mathbb{Z}_k$.

We know $\boldsymbol{m_j} = \boldsymbol{r} + \sum_{x=1}^{j} i_x \pmod k$, where $\boldsymbol{r}$ is a random variable representing the uniformly drawn group element of $\mathbb{Z}_k$. Thus,

$$\Pr\left(\boldsymbol{m_j} = m^* \mid \boldsymbol{I} = I\right) = \Pr\left(\boldsymbol{r} = m^* - \sum_{x=1}^{j} i_x \pmod k\right) = 1/k$$

and

$$\Pr(\boldsymbol{m_j} = m^*) = \sum_{I \in \Sigma^n} \Pr(\boldsymbol{m_j} = m^* \mid \boldsymbol{I} = I) \Pr(\boldsymbol{I} = I) = 1/k \sum_{I \in \Sigma^n} \Pr(\boldsymbol{I} = I) = 1/k$$

As such, we may conclude that $\boldsymbol{I}$ and $\boldsymbol{m_j}$ are independent (and this holds true for every $j \in \{1, ..., n\}$).

Consider the view $V = \langle i^*, m^*, \lambda^* \rangle$ at A_j. Then

$$\begin{aligned}\Pr(\boldsymbol{I} = I \mid \textbf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i_j}, \boldsymbol{\lambda_j}) = V) &= \Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = i^*, \boldsymbol{m_{j-1}} = m^*, \boldsymbol{\lambda_j} = \lambda^*) \\ &= \Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = i^*, \boldsymbol{\lambda_j} = \lambda^*)\end{aligned}$$

because $\boldsymbol{I}$ and $\boldsymbol{m_{j-1}}$ are independent. As a result, we have

$$\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = i^*, \boldsymbol{\lambda_j} = \lambda^*) = \Pr(\boldsymbol{I} = I \mid \textbf{\textit{view}}_{\mathcal{A}}^{\mathcal{P}}(\boldsymbol{i_j}, \boldsymbol{\lambda}) = V)$$

for every V consistent with input i^* and output λ^* (regardless of the value of m^*), as needed. □

In short, learning any agent's input (or any partial sum along the ring) reduces to guessing the exact random element added to the aggregate sum by A_1.

We now adapt this scheme to compute `Remainder` in the population model with information-theoretic privacy.

4.7.1 Algorithm Overview

Our algorithm aims to simulate the transfer of information exactly as in RingRemainder. We do this by assuming that the protocol has an initial leader with a special token that is passed between the agents. Each time an agent receives the token and some accompanying value, it adds its own input to that value and passes it along with the token to another agent. This signifies that the current owner of the token holds the aggregate sum of the agents' inputs who have previously held the token. When an agent passes the token to another agent, it labels itself as having already been visited by the token so as to ensure that its input is included in the sum exactly one time. Once the token has visited all of the agents in the population, it is returned to the leader (along with the total sum of all of the agents' inputs). In order to achieve this functionality, there are two crucial obstacles we must overcome:

First, we need some mechanism for securely transferring a message between two agents such that no other agent in the population learns this message except the sender of the message and the intended recipient of the message. Without such a mechanism, some holder of the special token may try to send its intermediate computation of the remainder to another agent who has previously been visited by the token, thereby leaking sensitive information (namely, the remainder of a subset of the population). In order to do this, we provide a secure peer-to-peer transfer subroutine in Section 4.7.2.

Second, we need a way to determine whether or not every agent in the population has been visited by the token. When this happens, we want the final token owner to pass the token back to the leader so that the leader can remove the randomness it initially added to the aggregate that has been passed among the agents. We must try to prevent passing the aggregate back to the leader before all inputs have been incorporated into the aggregate as this would cause some agents to be "skipped over" in the computation which would also leak the remainder of a subset of the population. In order to achieve this, we use the probing protocol from [AAE06a] which we describe in further detail in Section 4.7.3.

In the following subsections, we provide subroutines for performing these tasks which we then incorporate into the design of our main algorithm for computing `Remainder` with information-theoretic privacy in Section 4.7.4.

4.7.2 Secure Peer-to-Peer Transfer

As mentioned previously, in order for our algorithm to guarantee input privacy, the communication of the intermediate sums between any two agents must remain secure. Therefore, we now introduce a novel secure peer-to-peer transfer protocol (defined below), which itself is of independent interest.

Definition 4.9. *Let M be a message space, $\mathcal{D}$ be some distribution on M, and I be any fixed input vector in Σ^n. A* ***secure peer-to-peer transfer routine*** *is a protocol $\mathcal{P}$ that transfers data $m \overset{\mathcal{D}}{\leftarrow} M$ from one agent Sender to another $\mathsf{Receiver}$ such that there exist PPT algorithms W_1, W_2 where*

$$\Pr\left(W_1(\textbf{\textsf{view}}^{\mathcal{P}}_{\mathsf{Sender}}(\boldsymbol{I})) = m\right) = \Pr\left(W_2(\textbf{\textsf{view}}^{\mathcal{P}}_{\mathsf{Receiver}}(\boldsymbol{I})) = m\right) = 1$$

and for all $i : A_i \notin \{\mathsf{Sender}, \mathsf{Receiver}\}$ and PPT algorithm W'

$$\Pr\left(W'(\textbf{\textsf{view}}^{\mathcal{P}}_{\boldsymbol{A_i}}(\boldsymbol{I})) = m\right) = \Pr(m \overset{\mathcal{D}}{\leftarrow} M)$$

In other words, a secure peer-to-peer transfer routine allows a Sender to transfer a message m to a $\mathsf{Receiver}$ such that both Sender and $\mathsf{Receiver}$ are privy to m, but all other agents cannot guess m from their own view of the protocol with probability better than random (knowing only the *a priori* distribution on the message space).

Algorithm Description

Each agent's state consists of three components $\langle \mu, (r, L) \rangle$: a hidden secret, and a public randomness value and label. The goal of the protocol is to pass a secret

Algorithm 4.2: Population Protocol for Secure P2P Transfer

$$\langle \mu, (r, \mathfrak{S}) \rangle, \langle *, (*, \overline{\mathfrak{u}}) \rangle \rightarrow \langle \mu, (r', \mathfrak{S}) \rangle, \langle *, (*, \overline{\mathfrak{u}}) \rangle \tag{S1}$$

$$\langle \mu, (r, \mathfrak{S}) \rangle, \langle *, (*, \mathfrak{u}) \rangle \rightarrow \langle \perp, (\mu - r, \mathfrak{S}') \rangle, \langle r, (*, \mathfrak{R}) \rangle \tag{S2}$$

$$\langle \perp, (x, \mathfrak{S}') \rangle, \langle y, (*, \mathfrak{R}) \rangle \rightarrow \langle \perp, (\perp, \overline{\mathfrak{u}}) \rangle, \langle x + y, (*, \mathfrak{S}) \rangle \tag{S3}$$

Algorithm specifying secure peer-to-peer transfer protocol.

message from one agent (marked as the Sender using label $\mathfrak{S}$, of which there may only be one in the population) to another agent meeting some specified criteria labeled by $\mathfrak{u}$. There may be multiple agents, exactly one agent, or no agents with $\mathfrak{u}$.

The Sender waits to meet an agent with label $\mathfrak{u}$. Until it does, it must refresh its randomness at each interaction in order to make sure that the randomness it transmits to the Receiver is fresh. When the Sender finally meets some agent with $\mathfrak{u}$, it marks that agent as the Receiver and transmits its fresh randomness value r. It also updates its own token to $\mathfrak{S}'$ to remember that it has already met and labeled a Receiver. Then, the Sender waits to meet the Receiver again, at which point it gives it a message masked with the randomness it sent in the previous interaction and marks itself with the label $\overline{\mathfrak{u}}$. This algorithm has state space $(\mathbb{Z}_k \cup \{\perp\})^2 \times \{\mathfrak{S}, \mathfrak{S}', \mathfrak{R}, \mathfrak{u}, \overline{\mathfrak{u}}\}$, which for constant k is of size $O(1)$. Transitions are specified in Algorithm 4.2.

Transition (S1) implements the condition that the Sender refresh its randomness each time it interacts with an agent that has $\overline{\mathfrak{u}}$ (taken to mean that it does not meet some specified criteria for receiving a message). Transition (S2) designates an appropriate agent as the Receiver and transfers over r from the Sender, which the Receiver stores internally. This also updates the Sender's external value to be the difference of its message with the randomness (modulo k). Finally, (S3) detects that the Sender and specified Receiver are interacting with one another for the second time (as they are now each marked with tokens corresponding to their respective roles $\mathfrak{S}'$ and $\mathfrak{R}$) - at this point the Sender relinquishes its token to the Receiver and the

Receiver computes

$$x + y = \mu - r + r = \mu \pmod{k}$$

A precondition for running this protocol is that the Sender must have a specially marked token $\mathfrak{S}$ of which there may be exactly one in the population. By the end of the protocol, exactly one agent is selected as the Receiver, and it now stores μ internally. We claim that Algorithm 4.2 is a secure peer-to-peer transfer routine, i.e. for any agent not equal to the Sender or the Receiver, the probability of correctly guessing μ is no better than it is when the above protocol is not performed at all:

Theorem 4.4. *Algorithm 4.2 is a secure peer-to-peer transfer routine.*

Proof. First note that the label component L of the message state is updated independently of μ and therefore leaks no information about μ. Thus we will focus our attention to the state updates of the hidden secret and the public randomness.

There is one specified Sender with token $\mathfrak{S}$ and some number of eligible Receiver agents. By (S1), the randomness used in the Sender's external message is refreshed for each interaction, so when an agent is finally labeled as the designated Receiver in (S2), the random value which it copies to its internal state is known (with probability 1) only by the Sender and the Receiver. Let $\boldsymbol{r} \leftarrow \mathbb{Z}_k$ be a random variable representing the actual random value transferred in this transition between the Sender and the Receiver.

At this point, the Sender changes its external message to be $\boldsymbol{s} = \mu - \boldsymbol{r} \pmod{k}$. Recall that $\boldsymbol{r}$ is uniformly drawn from $\mathbb{Z}_k$, so for all $x \in \mathbb{Z}_k$ it is the case that $\Pr(\boldsymbol{r} = x) = 1/k$. As such, for all $x \in \mathbb{Z}_k$, $\Pr(\boldsymbol{s} = \mu - \boldsymbol{r} = x) = \Pr(\boldsymbol{r} = \mu - x) = 1/k$, so $\boldsymbol{s}$ is also uniformly distributed over $\mathbb{Z}_k$ (regardless of the a priori distribution $\mathcal{D}$ from which the hidden secret μ is drawn). Because the Receiver is the only other agent that knows the exact value of $\boldsymbol{r}$, it learns μ on its next interaction with the Sender.

Define W_1 to be an algorithm that selects the hidden secret of a Sender agent when it interacts as the $\mathsf{Initiator}$ with another agent with label $\mathfrak{u}$, and define W_2 to be an algorithm that selects the hidden secret of a $\mathsf{Receiver}$ agent when it interacts as the $\mathsf{Responder}$ with another agent with label $\mathfrak{S}'$. Then,

$$\Pr\left(W_1(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathsf{Sender}}(\boldsymbol{I})) = \mu\right) = \Pr\left(W_2(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\mathsf{Receiver}}(\boldsymbol{I})) = \mu\right) = 1$$

However, for every other agent A_j in the population, every observed external message (including $\mu - \boldsymbol{r}$) appears to be a uniformly random value, independent of the distribution from which the hidden secret μ is drawn. Thus any W' must guess the value of μ in the absence of additional information beyond the *a priori* distribution on the message space:

$$\Pr\left(W'(\boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{A_j}}(\boldsymbol{I})) = \mu\right) = \Pr(\mu \xleftarrow{\mathcal{D}} \mathbb{Z}_k)$$

As a result, only the Sender and the chosen $\mathsf{Receiver}$ can know μ, making Algorithm 4.2 a secure peer-to-peer transfer routine. □

4.7.3 Probing Protocol

Recall that we want to use Algorithm 4.2 to securely transmit the intermediate sums of agent inputs along some path traversing every agent in the population, starting with the leader. However, one final component that we need in order for this algorithm to succeed is for us to detect when every agent in the population has been included in the aggregation so that the final agent in the traversal can pass the sum back to the leader (who has knowledge of the value to remove from this sum to get the protocol output). For this, we will use a probing protocol.

A **probing protocol**, or **probe**, is a population protocol that detects whether there exists an agent in the population satisfying a given predicate [AAE06a]. Algorithm 4.3 gives transition rules for performing a probe for a predicate π using three

Algorithm 4.3: Population Protocol for Epidemic-Driven Probing

$$
\begin{aligned}
&x, y \to x, \max(x, y) && \text{(P1)}\\
&0, y \to 0, y && \text{(P2)}\\
&x, y \to x, 2 \quad [x > 0] && \text{(P3)}
\end{aligned}
$$

Transition rules for the probing protocol from [AAE06a], where x and y are in $\{0, 1, 2\}$.

states, 0, 1, and 2: When the Responder does not satisfy π, the agents transition via (P1) and when the Responder does satisfy π, they transition according to (P2) and (P3). In essence, the probe (initiated by the leader) sends out a 1-signal through a population of agents in state 0. If the 1-signal reaches an agent satisfying the predicate, that agent initiates a 2-signal which spreads back to the leader by epidemic. Higher number epidemics overwrite lower ones, so if some agent in the population satisfies π then the leader eventually sees the 2-signal. The probe, used in conjunction with the phase clock from the same work [AAE06a], allows the leader to detect the presence of an agent satisfying π in $O(n \log n)$ interactions using $O(1)$ states with probability $1 - n^{-c}$ for any fixed constant $c > 0$.

The following lemma gives bounds on the running time of the probing protocol in Algorithm 4.3 and quantifies its probability of error:

Lemma 4.1 (see [AAE06a], Lemma 6)**.** *For any $c > 0$, there is a constant d such that for sufficiently large n, with probability at least $1 - n^{-c}$ it is the case that after $dn \ln n$ interactions in the probing protocol either (a) no agent satisfies the predicate and every agent is in state 1, or (b) some agent satisfies the predicate and every agent is in state 2.*

For the sake of simplicity, we will use Algorithm 4.3 in a black-box manner: We consider the "output" of the protocol (computed only at the leader) to be 0 for states 0 and 1, and 1 for state 2 (i.e. the leader's probe subroutine outputs 1 if and only if some agent in the population satisfies π). At the start of each round of the phase clock, all agents reset their value to 0 and allow the leader to initiate a new probe. Both the probe and the phase clock states are components of the message space, and the transitions for these subroutines are independent of the transitions for the main protocol, therefore we consider the two "protocols" to be taking place in parallel.

4.7.4 `Remainder` with Information-Theoretic Privacy

We provide here a novel protocol which computes `Remainder` with high probability and achieves *information-theoretic input privacy* in the population model. The algorithm works under the fairness assumption that the scheduler selects ordered pairs of agents to interact at each step uniformly at random from all possible ordered pairs of agents. It also presumes the message model, where the state of each agent is divided into a concealed internal portion s and a publicly visible external portion m, denoted $\langle s, m\rangle$.

Algorithm Description

First, each agent applies the input function $\mathcal{I}$ to their input as follows:

$$\mathcal{I}(i_j, \ell) = \begin{cases} \langle i_j + r^0, (r^j, \mathfrak{S}, 1, Z = Z_0)\rangle & \ell = 1 \\ \langle i_j, (r^j, \mathfrak{u}, 0, Z = Z_0)\rangle & \ell = 0 \end{cases}$$

where r^j is drawn uniformly at random from $\mathbb{Z}_k$ for $j \in \{0, 1, ..., n\}$, and Z (initialized to Z_0) is a probe subroutine (including its associate phase clock). This input function presumes that the population already has a single designated leader, specified by $\ell = 1$. For ease of reference, we will name the components of the state $\langle \mu, (r, L, \ell, Z)\rangle$ as follows: μ is the hidden internal component of the state called the **secret**, r is the **mask**, L is the agent's **label**, ℓ is the **leader bit**, and Z is the **probe** subroutine. The transitions describing the protocol can be found in Algorithm 4.4.

In this algorithm, the general structure of the transitions from the secure peer-

Algorithm 4.4: Information-Theoretically Private Remainder

$$\langle *, (r, \mathfrak{S}, *, *)\rangle, \langle *, (*, \overline{\mathfrak{u}}, *, *)\rangle \rightarrow \langle *, (r', \mathfrak{S}, *, *)\rangle, \langle *, (*, \overline{\mathfrak{u}}, *, *)\rangle \tag{R1}$$

$$\langle u, (r, \mathfrak{S}, *, *)\rangle, \langle v, (*, \mathfrak{u}, *, *)\rangle \rightarrow \langle \perp, (u - r, \mathfrak{S}', *, *)\rangle, \langle v + r, (*, \mathfrak{R}, *, *)\rangle \tag{R2}$$

$$\langle *, (x, \mathfrak{S}', *, *)\rangle, \langle y, (*, \mathfrak{R}, *, *)\rangle \rightarrow \langle *, (\perp, \overline{\mathfrak{u}}, *, *)\rangle, \langle x + y, (*, \mathfrak{S}, *, *)\rangle \tag{R3}$$

$$\langle \perp, (\perp, \overline{\mathfrak{u}}, 1, 1)\rangle, \langle *, (*, *, *, *)\rangle \rightarrow \langle \perp, (\perp, \mathfrak{u}, 1, 1)\rangle, \langle *, (*, *, *, *)\rangle \tag{R4}$$

Algorithm for information-theoretically private remainder computation. Although not notated above, each interaction also performs an update to the probing subroutine by progressing the phase clock and executing probing (advancing the detector by one step respective to the state of the detector at each agent in the interaction). Transitions (R1-R3) are analogous to transitions (S1-S3) from Algorithm 4.2.

to-peer transfer protocol in Algorithm 4.2 is used to send the intermediate sums. However, instead of just storing the message received, the Receiver computes the sum of the message and its own input and stores the result internally. Each subsequent Sender searches the population for an agent whose input has not yet been incorporated into the sum (signified by the $\mathfrak{u}$ state). When no one in the population has $\mathfrak{u}$ anymore, the probe senses this and "goes off," outputting 1 at the leader from this point onward.

When the probe goes off, with high probability every agents' label is set to $\overline{\mathfrak{u}}$, so this alerts the leader to set its own label to $\mathfrak{u}$. This makes the leader the only agent capable of being labeled as the next Receiver by the current Sender. When the leader receives the value stored at the final Sender, we can have the leader place the answer into a separate portion of the external state (not shown in Algorithm 4.4) so that all other agents can copy it, which takes $O(n^2 \log n)$ additional steps with high probability. The leader must also have an additional component to its *hidden* state which stores the randomness used in its initial message transfer via (R2) (also not shown in Algorithm 4.4).

Proof of Correctness

From Algorithm 4.4, we can see that the effective steps of the computation are driven by the Sender via transitions (R1), (R2), and (R3). The only transition not involving the Sender is (R4), wherein the leader sets its label from $\overline{\mathfrak{u}}$ to $\mathfrak{u}$ when it detects that every other agent has label $\overline{\mathfrak{u}}$, signifying that every agent's value is aggregated into a single sum held by the current Sender (this only happens once). When such a point is reached, that Sender should now pass that sum back to the leader via (R2) and (R3), i.e. through secure peer-to-peer transfer. We aim to prove that Algorithm 4.4 computes Remainder in $\Theta(n^3 \log n)$ steps with high probability. First, we will prove a sequence of lemmas that we will later use to prove the overall correctness and running time of Algorithm 4.4 in Theorem 4.5.

Lemma 4.2. *In any configuration of a population executing Algorithm 4.4, exactly one agent has label $\mathfrak{S}$ or $\mathfrak{S}'$.*

Proof. In the initial configuration, this is true by construction of the input function $\mathcal{I}$. At any later step, if we assume that exactly one agent in the population has $\mathfrak{S}$ or $\mathfrak{S}'$, then it either:

- maintains ownership of the unique token $\mathfrak{S}$ via (R1),
- updates its token to $\mathfrak{S}'$ (thereby eliminating the only $\mathfrak{S}$ in the population and replacing it with $\mathfrak{S}'$) via (R2), or
- transfers this token (originally as $\mathfrak{S}'$) to another agent (now $\mathfrak{S}$) via (R3).

Transition (R4) does not involve agents with the label $\mathfrak{S}$ nor $\mathfrak{S}'$. Thus by induction, this claim is true at every step of any execution of Algorithm 4.4. □

Lemma 4.3. *For any $c > 0$, starting from a configuration C with exactly one agent having label $\mathfrak{S}$, exactly t agents with label $\mathfrak{u}$, and the remaining agents with label $\overline{\mathfrak{u}}$,* *with probability at least* $1 - n^{-c}$*, these labels remain unchanged for* $\Theta(\frac{cn^2 \log n}{t})$ *steps, at which point transition (R2) is performed.*

Proof. The probability that the scheduler selects an $\mathsf{Initiator}$ with label $\mathfrak{S}$ and a $\mathsf{Responder}$ with label $\mathfrak{u}$ (resulting in transition (R2)) is $\frac{1}{n} \cdot \frac{t}{n-1}$. Let $\boldsymbol{X}$ be a random variable representing the number of steps for (R2) to occur from this starting configuration. By independence and uniform randomness of the scheduler, $\boldsymbol{X}$ is geometric with parameter $p = \frac{t}{n(n-1)}$, so $\boldsymbol{X} = \frac{n(n-1)}{t}$ in expectation, or is at most k with probability

$$\begin{aligned}
\Pr(\boldsymbol{X} \leq k) &= \sum_{j=0}^{k} (1-p)^j \, p \\
&= \sum_{j=0}^{k} \left(1 - \frac{t}{n(n-1)}\right)^j \frac{t}{n(n-1)} \\
&= 1 - \left(1 - \frac{t}{n(n-1)}\right)^{k+1}
\end{aligned}$$

For $k = \frac{n(n-1) \cdot c \log n}{t} - 1$, this probability is at least $1 - e^{-c \log n}$ (as a result of the fact that $(1 - 1/x)^x < 1/e$ for $x \geq 1$).

The only other non-null transition that can possibly occur from C is between labels $\mathfrak{S}$ and $\mathfrak{u}$ (thereby eliciting transition (R1)), which has no effect on the labels of the agents in the population. Therefore, the labels in the population do not change until (R2) is executed once, which occurs in $\Theta(cn^2 \log n/t)$ steps with probability $1 - e^{-c \log n} = 1 - n^{-c}$ for any choice of $c > 0$. □

Lemma 4.4. *For any* $c > 0$*, starting from a configuration with exactly one agent having label* $\mathfrak{S}'$ *and exactly one agent having label* $\mathfrak{R}$*, we execute transition (R3) in* $\Theta(cn^2 \log n)$ *steps with probability* $1 - n^{-c}$*.*

Proof. By Lemma 4.2, this agent with label $\mathfrak{S}'$ is the only agent having either label $\mathfrak{S}$ or $\mathfrak{S}'$ in the population. Therefore, transition (R3) can eventually be taken when the agents with $\mathfrak{S}'$ and $\mathfrak{R}$ meet. Any particular pair of agents interacts with probability

$1/\binom{n}{2}$. By the uniform randomness of the probabilistic scheduler, we expect to wait $O(n^2)$ steps on average for this event to occur. Moreover, the probability that this event occurs within $cn^2 \log n$ steps is at least $1 - e^{-c \log n} = 1 - n^{-c}$ for any choice of $c > 0$. □

Due to the fact that (R2) results in a configuration with exactly one $\mathfrak{S}'$ and one $\mathfrak{R}$, we have the following corollary:

Corollary 4.1. *For any $c > 0$, after any step where (R2) occurs, (R3) must also occur within $\Theta(cn^2 \log n)$ steps with probability $1 - n^{-c}$.*

Lemma 4.5. *Transitions (R2) and (R3) aggregate the sum of the secrets of the* Sender *and the* Receiver *modulo k into the secret of the* Receiver *in $\Theta(cn^2 \log n)$ steps with probability $1 - n^{-c}$ for any $c > 0$.*

Proof. By Corollary 4.1, (R2) and (R3) are essentially executed as a pair (taking $\Theta(cn^2 \log n)$ steps with high probability). Let the step where (R2) is executed between two agents A_i (as the Initiator) and A_j (as the Responder) be τ_0. The secrets of these agents are u and v respectively, thus we aim to show that after (R2) and (R3) are performed, A_j has secret $u + v \pmod{k}$.

Agent A_i has label $\mathfrak{S}$ and is therefore the Sender, and upon executing transition (R2) we have that A_j is labeled as the Receiver. After this transition, we also have that the Sender relabels itself with $\mathfrak{S}'$ to signify that a Receiver has been selected. A_i has secret $\perp$ and mask $u + r$, and A_j has secret $v - r$.

By Lemma 4.4 and Corollary 4.1, the next state update involving these two agents executes (R3) and this occurs within $\Theta(cn^2 \log n)$ steps with probability $1 - n^{-c}$ for any $c > 0$. The only other effective step which can occur is (R4) because every other transition involves some agent with a $\mathfrak{S}$ label and by Lemma 4.2 there is no other $\mathfrak{S}$ label in the population. Moreover, (R4) does not change the state of A_i nor A_j because their labels are not equal to $\overline{\mathfrak{u}}$.

We know that when (R3) is eventually performed, this must be with A_i as the Initiator and A_j as the Responder because these are the only two agents with the appropriate labels in the population and no other effective steps have been taken to transfer them to any other agents. Thus, A_i still has $u + r$ as its mask and A_j has $v - r$ as its secret. Transition (R3) then results in A_j updating its secret to $x+y = (u-r)+(v+r) = u+v$ and leaves A_i with label $\bar{\mathfrak{u}}$ (indicating that its secret is now $\perp$). □

Lemma 4.6. *For any $c > 0$, starting from a configuration C with exactly one agent having label $\mathfrak{S}$, exactly t agents with label $\mathfrak{u}$, and the remaining agents with label $\bar{\mathfrak{u}}$, with probability at least $1 - n^{-c}$, we reach a configuration C' with exactly one agent having label $\mathfrak{S}$, exactly $t - 1$ agents with label $\mathfrak{u}$, and the remaining agents with label $\bar{\mathfrak{u}}$ in $\Theta(cn^2 \log n)$ steps.*

Proof. By Lemma 4.3, for any $c' > 0$ and configuration C, we execute (R2) in $\Theta(c'n^2 \log n/t)$ steps with probability $1 - n^{-c'}$ (only changing the randomness values for some number of agents). When transition (R2) is executed, the labels $\mathfrak{S}$ and $\mathfrak{u}$ are overwritten by $\mathfrak{S}'$ and $\mathfrak{R}$. By Lemma 4.4, for any $c'' > 0$ we execute (R3) in $\Theta(c''n^2 \log n)$ additional steps from this configuration with probability $1 - n^{-c''}$, at which point the labels $\mathfrak{S}'$ and $\mathfrak{R}$ are overwritten by $\bar{\mathfrak{u}}$ and $\mathfrak{S}$. All in all, the net change in these $\Theta(c'n^2 \log n/t + c''n^2 \log n)$ steps is a decrease in the number of agents with label $\mathfrak{u}$ by 1 (resulting in an increase in the number of agents with label $\bar{\mathfrak{u}}$ by 1). For any $c > 0$, let $c' = c'' = c + 1$. Taking the union bound over the probability of failure for each phase described in these lemmas gives a total probability of failure of $2n^{-(c+1)}$ which is at most n^{-c} for $c > 0$ and $n \geq 2$. Plugging this expression for c' and c'' in terms of c back into the above asymptotic bound on steps, and knowing $1 \leq 1 + 1/t \leq 2$ for $t > 0$, we have a total run time of

$$\Theta(c'n^2 \log n/t + c''n^2 \log n) = \Theta((c+1)(1+1/t)n^2 \log n) = \Theta(cn^2 \log n)$$

□

Lemma 4.7. *For any $c > 0$, in $\Theta(cn^3 \log n)$ steps, with probability $1 - n^{-c}$ Algorithm 4.4 aggregates all agent secrets into one sum modulo k and labels all agents with $\overline{\mathfrak{u}}$ except one agent marked as the* Sender *holding the sum.*

Proof. At the start of the protocol, each agent A_j has state $\langle i_j, (r^j, \mathfrak{u}, 0, 0)\rangle$ except the leader who has state $\langle i_j + r^0, (r^j, \mathfrak{S}, 1, 0)\rangle$. This means that exactly one agent has label $\mathfrak{S}$ while $n - 1$ agents have label $\mathfrak{u}$.

The only transition that can be taken is (R2) followed by (R3), which aggregates the sum of the leader's secret with some other agent in the population by Lemma 4.5. This also results in the leader being labeled with $\overline{\mathfrak{u}}$ and this selected agent being the new Sender (reducing the number of agents with $\mathfrak{u}$ by 1).

By Lemma 4.6, for any $c' > 0$, it takes $\Theta(c'n^2 \log n)$ steps to reduce the number of agents with $\mathfrak{u}$ by 1 with probability $1 - n^{-c'}$. By applying Lemma 4.6 repeatedly, we find that it takes

$$\Theta\left(\sum_{t=1}^{n-1} c'n^2 \log n\right) = \Theta(c'(n-1)n^2 \log n) = \Theta(c'n^3 \log n)$$

steps for the number of agents with label $\mathfrak{u}$ to become 0. Each time the number of agents with label $\mathfrak{u}$ decrements, the agent whose label is overwritten has its secret aggregated into the secret of the Sender and is then labeled with $\overline{\mathfrak{u}}$ to mark itself so it does not become a Sender again. When no agent has $\mathfrak{u}$, the Sender has the total sum of the agent secrets. Taking a union bound over the probability of failure each time we applied Lemma 4.6, we have that the total probability of failure is $n \cdot n^{-c'}$. For any $c > 0$, letting $c' = c + 1$ yields the desired result for $n \geq 1$. □

Lemma 4.8. *Transition (R4) only occurs once, when the probe determines that no agent in the population has label $\mathfrak{u}$. For any $c > 0$, with probability $1 - n^{-c}$, the probe only goes off when all agents have $\overline{\mathfrak{u}}$.*

Proof. By construction, there is only one leader in the population and every agent is initialized with a probe that outputs 0 upon instantiation. Once the probe goes off, it cannot be reset. Only when the probe goes off can (R4) occur, so this can only happen once and only to the leader.

By Lemma 4.1, when some agent in the population satisfies π, for any $c' > 0$, only with probability $n^{-c'}$ do we have that not every agent is in state 2. If one such agent is the leader who is probing for the label $\mathfrak{u}$, then it mistakenly outputs 0 at the end of the phase clock cycle and causes the protocol to skip some number of agents in the protocol computation which may result in an incorrect output. In order to avoid this outcome, we need for the leader's probe to succeed for every phase clock round over the course of the protocol. By Lemma 4.7, for any $c'' > 0$ it takes $\Theta(c''n^3 \log n)$ steps to aggregate all agent inputs into one sum with probability $1 - n^{-c''}$. For $c'' > 0$, because a phase clock has $O(n \log n)$ steps in a round, we need the probe to succeed $O(c''n^2)$ consecutive times. By the union bound, the probability that the probe fails at least once in these $O(c''n^2)$ consecutive trials is $dc''n^2 \cdot n^{-c'}$ for some constant d. Thus for any $c > 0$, letting $c' = c + 3$ gives the desired result for sufficiently large n. □

We are now ready to prove the following theorem:

Theorem 4.5. *For any fixed* $c > 0$*, Algorithm 4.4 computes* `Remainder` *in* $\Theta(n^3 \log n)$ *steps with probability at least* $1 - n^{-c}$*.*

Proof. By Lemma 4.7, for any $c' > 0$, in $\Theta(c'n^3 \log n)$ steps (with probability $1 - n^{-c'}$), the population aggregates all agent secrets into one sum held by a single agent with the singular Sender label, at which point all other agents are marked with $\overline{\mathfrak{u}}$ and the Sender has secret

$$r^0 + \sum_{j=1}^{n} i_j \pmod{k}$$

By Lemma 4.1, for any $c'' > 0$, the probe does not go off before this point and successfully does go off within $O(c''n \log n)$ steps of reaching this point with probability $1-n^{-c''}$. Therefore, by Lemma 4.8, for any $c''' > 0$ (R4) is then executed (which takes constant time in expectation and polylogarithmic time with probability $1 - n^{-c'''}$), changing only the leader's label to $\mathfrak{u}$. This allows the one remaining agent with the $\mathfrak{S}$ token to perform one last effective exchange with the leader, which, by Lemma 4.5, transfers the aggregate sum back to the leader in $\Theta(c''''n^2 \log n)$ additional steps with probability $1 - n^{-c''''}$ for any $c'''' > 0$.

Taking the union bound over these four sources of error gives us a probability of failure of $n^{-c'} + n^{-c''} + n^{-c'''} - n^{-c''''}$. For any fixed $c > 0$, letting $c' = c'' = c''' = c'''' = c + 1$ gives a probability of success that is at least $1 - n^{-c}$ for sufficiently large n.

Finally, the leader can remove r^0 from the secret to obtain the remainder of the sum of agents' inputs modulo k. A separate mechanism may be used to distribute this value to the rest of the population by epidemic. □

Proof of Security

The final result of this chapter demonstrates the privacy guarantee of Algorithm 4.4. The algorithm has some probability of failure in computing the `Remainder` correctly, but when the computation is successful, we always have that the following information-theoretic privacy property holds:

Theorem 4.6. *When Algorithm 4.4 correctly computes the* `Remainder` *predicate, it satisfies information-theoretic input privacy.*

Proof. Assuming that the protocol correctly computes the `Remainder` predicate (i.e. the phase clock does not experience any failures until at least the time T past convergence of the protocol), an agent's view consists of its own input, its initial state, and some number of observed interactions (from which all other information to which it has access, such as the protocol output, can be derived).

Let us consider the view of an agent in the population A_j until some step $\tau >> T$, where the agent has input (i^*, ℓ^*) and output λ^*, letting $\alpha_j(\tau)$ be the number of interactions in which A_j participates until step τ (so that $\sum_{j=1}^{n} \alpha_j(\tau) = \tau$):

$$\boldsymbol{\mathsf{view}}_{\boldsymbol{A_j}}^{\boldsymbol{\mathcal{P}}}\boldsymbol{((i^*, \ell^*), \lambda^*)^\tau} = \langle (i^*, \ell^*); \boldsymbol{q_0^j}; \{\boldsymbol{(\rho_t, (r_t, L_t, \ell_t, Z_t))}\}_{t=1}^{\alpha_j(\tau)} \rangle$$

Recall that $\mathcal{I}$ is randomized, so $\boldsymbol{q_0^j}$ is a random variable representing the actual initial state of agent A_j as determined by the randomness used to compute $\mathcal{I}$.

Denote by $\mathcal{Z}$ the state space of the probing protocol, and let

$$\mathcal{V} = \mathbb{Z}_k \times \{0,1\} \times \mathbb{Z}_k \times \{\{\mathsf{Initiator}, \mathsf{Responder}\} \times \mathbb{Z}_k \times \{\mathfrak{S}, \mathfrak{S}', \mathfrak{u}, \overline{\mathfrak{u}}\} \times \{0,1\} \times \mathcal{Z}\}^{\alpha_j(\tau)}$$

be the space of all possible views of A_j until step τ of the computation.

Additionally, let $\boldsymbol{\theta_t} = \boldsymbol{(\rho_t, (r_t, L_t, \ell_t, Z_t))}$ be a random variable representing the t-th observation of A_j. Among the sequence $\{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)}$, there are four special observations that must appear in any correct execution of the protocol:

$$\theta_a = (\mathsf{Initiator}, (r_a, \mathfrak{S}, \ell_a, Z_a)), \text{ } \mathsf{Sender} \text{ initiates secure transfer}$$

$$\theta_b = (\mathsf{Initiator}, (r_b, \mathfrak{S}', \ell_b, Z_b)), \text{ } \mathsf{Sender} \text{ transmits secret value}$$

$$\theta_c = (\mathsf{Responder}, (r_c, \mathfrak{u}, \ell_c, Z_c)), \text{ eligible } \mathsf{Receiver} \text{ is selected}$$

$$\theta_d = (\mathsf{Responder}, (r_d, \mathfrak{R}, \ell_d, Z_d)), \text{ secret sent to selected } \mathsf{Receiver}$$

These observations occur in the above order for everyone except the leader (who instead views: $\theta_c, \theta_d, \theta_a, \theta_b$). The remaining observations either cause the Sender to refresh its randomness value, the leader to relabel itself with $\overline{\mathfrak{u}}$, or do not result in a state change.

We want to show that the vector of values $\langle \boldsymbol{q_0^j}, \{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)} \rangle$ is conditionally independent of the input vector $\boldsymbol{I}$ given the fixed input value (i^*, ℓ^*) at A_j and the output λ^*

observed by A_j, which would imply

$$\begin{aligned}
&\Pr(\boldsymbol{I} = I^* \mid \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{A_j}}(\boldsymbol{(i^*, \ell^*), \lambda^*})^{\tau} = V) \\
=\ &\Pr(\boldsymbol{I} = I^* \mid \boldsymbol{i_j} = (i^*, \ell^*), \langle \boldsymbol{q_0^j}, \{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)} \rangle = \langle q^*, \{\theta_t^*\}_{t=1}^{\alpha_j(\tau)} \rangle, \boldsymbol{\lambda_j} = \lambda^*) \\
=\ &\Pr(\boldsymbol{I} = I^* \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*)
\end{aligned}$$

In order to show this, we first restrict our attention to a fixed schedule $s \in S$ (i.e. a specific choice of agent pairs selected to interact for τ steps of the protocol), where

$$S = \{\{(x_t, y_t)\}_{t=1}^{\tau} : (x_t, y_t) \in \mathbb{Z}_n^2 \wedge x_t \neq y_t\}$$

Let us assume that we only consider schedules s wherein the protocol succeeds (i.e. all but some polynomially small fraction of the schedules).

Because $\boldsymbol{q_0^j}$ is a function of (i^*, ℓ^*), we can say that the event $\boldsymbol{i_j} = (i^*, \ell^*) \wedge \boldsymbol{q_0^j} = q^* = \langle i^*, (r^*, \_, \ell^*, Z_0) \rangle$ is exactly the same as the event $\boldsymbol{i_j} = (i^*, \ell^*) \wedge \boldsymbol{r^j} = r^*$, where $\boldsymbol{r^j}$ is a random variable representing the initial randomness used by agent A_j in its computation of the input function $\mathcal{I}$. Recall that this $\boldsymbol{r^j}$ is uniform over $\mathbb{Z}_k$ and independent of all other variables in the protocol (including $\boldsymbol{I}, \boldsymbol{i_j}, \boldsymbol{\lambda_j}$, and s).

Furthermore, when a schedule s is fixed, the order of agent pairs selected to interact determines the order in which the Sender and Receiver tokens are passed among the agents. This fully determines the $\boldsymbol{L_t}, \boldsymbol{\ell_t}$, and $\boldsymbol{Z_t}$ components of the $\boldsymbol{\theta_t}$. The only remaining values to consider are the external randomness value $\boldsymbol{r_t}$ and the internal secret state $\boldsymbol{\mu_t}$ (which is correlated with the value of $\boldsymbol{r_t}$ due to (R2)). However, by the assumption that the protocol successfully computes the `Remainder`, we can see that (R2) is only executed once at each agent when the agent observes θ_a followed sometime afterwards by θ_b (which invoke transitions (R2) and (R3), respectively). For fixed s, the number of interactions between these two observations is fixed, and therefore the only correlation between these two values is that they are separated by the secret being transferred. For a fixed schedule (where the order in which agent values are aggregated is fixed), recall that this secret is equal to $r^0 + \sum_{x \in X \subseteq \mathbb{Z}_n} i_x$. All other observations have $\boldsymbol{r_t}$ that is drawn uniformly and independently of all other values. Therefore, the event $\{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)} \wedge s$ is the same as $\boldsymbol{r^0} = r^{**} \wedge \{\boldsymbol{r_t} = r_t^*\}_{t=1}^{\alpha_j(\tau)} \wedge s$, where $r^{**} = r_t^* - \sum_{x \in X \subseteq \mathbb{Z}_n} i_x \pmod k$ and each r_t^* is the exact observed randomness value in each observation made by A_j. This is a consequence of the fact that given a fixed schedule, the only variation in the values of each of the $\boldsymbol{\theta_t}$ is from the randomness values drawn by the agents at each step, which are also uniformly random and independent of all other variables of the protocol.

Let $\boldsymbol{S}$ be a random variable representing the τ ordered pairs of agents chosen to interact by the scheduler. Conditioned on some fixed schedule s and some fixed view V at A_j, and by the mutual independence of the uniformly drawn randomness values with all other variables in the protocol, for any $I \in \Sigma^n$ that is consistent with input (i^*, ℓ^*) at A_j and output λ^*,

$$\begin{aligned}
&\Pr(\boldsymbol{I} = I \mid \boldsymbol{\mathsf{view}}_{\boldsymbol{A_j}}^{\boldsymbol{\mathcal{P}}}((\boldsymbol{i^*}, \boldsymbol{\ell^*}), \boldsymbol{\lambda^*})^{\boldsymbol{\tau}} = V, \boldsymbol{S} = s) \\
= \ &\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \langle \boldsymbol{q_0^j}, \{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)} \rangle = \langle q^*, \{\theta_t^*\}_{t=1}^{\alpha_j(\tau)} \rangle, \boldsymbol{\lambda_j} = \lambda^*, \boldsymbol{S} = s) \\
= \ &\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{r^j} = r^*, \{\boldsymbol{r_t} = r_t^*\}_{t=1}^{\alpha_j(\tau)}, \boldsymbol{\lambda_j} = \lambda^*, \boldsymbol{S} = s) \\
= \ &\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*, \boldsymbol{S} = s)
\end{aligned}$$

Thus $\boldsymbol{I}$ is conditionally independent of $\langle \boldsymbol{q_0^j}, \{\boldsymbol{\theta_t}\}_{t=1}^{\alpha_j(\tau)} \rangle$ given $\boldsymbol{S}$, $\boldsymbol{i_j}$, and $\boldsymbol{\lambda_j}$. By the law of total probability, summing over all possible schedules $s \in S$ gives

$$\begin{aligned}
&\Pr(\boldsymbol{I} = I \mid \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{A_j}}((\boldsymbol{i^*}, \boldsymbol{\ell^*}), \boldsymbol{\lambda^*})^{\boldsymbol{\tau}} = V) \\
= &\sum_{s \in S} \Pr(\boldsymbol{I} = I \mid \boldsymbol{\mathsf{view}}^{\mathcal{P}}_{\boldsymbol{A_j}}((\boldsymbol{i^*}, \boldsymbol{\ell^*}), \boldsymbol{\lambda^*})^{\boldsymbol{\tau}} = V, \boldsymbol{S} = s) \Pr(\boldsymbol{S} = s) \\
= &\sum_{s \in S} \Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*, \boldsymbol{S} = s) \Pr(\boldsymbol{S} = s) \\
= &\sum_{s \in S} \Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*) \Pr(\boldsymbol{S} = s), \text{ by independence of the scheduler} \\
= &\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*) \sum_{s \in S} \Pr(\boldsymbol{S} = s) \\
= &\Pr(\boldsymbol{I} = I \mid \boldsymbol{i_j} = (i^*, \ell^*), \boldsymbol{\lambda_j} = \lambda^*)
\end{aligned}$$

as needed. □

Thus we have shown that Algorithm 4.4 is information-theoretically private whenever the protocol successfully computes the `Remainder` (which happens with probability $1 - n^{-c}$ for any $c > 0$). When the protocol fails due to a phase clock error, we actually do not know how much information is leaked by the protocol. Though we suspect that the information is limited, we designate this as outside of the scope of this work and only make claims about privacy in the case that the protocol is successful.

In order to show that we can achieve information-theoretic input privacy (with high probability) for all semilinear predicates, as in [DGFGR07], similar algorithms for computing `Threshold` and `Or` are also needed. We leave these problems as open for future work.

Chapter 5

Catalytic Inputs

This chapter is motivated by applications of population protocols in chemical reaction networks. Here, we introduce and study a variation of the population protocol model wherein certain states are persistent. We formally define this model and lower bounds on computing various problems therein: In Theorem 5.1, we show that `Parity` cannot be computed in this model in fewer than $\Omega(N^2)$ steps, and in Theorem 5.2 we show that a lower bound of $\Omega(\sqrt{N})$ on the initial input margin is needed to compute `Majority` in $O(N \log N)$ steps.

Then, we study the effect of spontaneous state changes on the computation of the `Majority` problem in the presence of persistent states. In Theorems 5.3, 5.4, and 5.5, we show that an adaptation of third-state dynamics can be used to solve approximate majority in the presence of catalysts and leaks, both separately and together. Finally, in Theorem 5.6 we form a comparison between the effects of leaks and that of Byzantine processes on population protocols.

The results of this chapter were produced in a joint work with James Aspnes and John Lazarsfeld (see [AAL20]).

5.1 Introduction

Recall that population protocols are a special case of chemical reaction networks (CRNs), which are systems of transition rules describing how a set of chemical reactants stochastically transform into a set of chemical products. Specifically, population protocols are chemical reaction networks with exactly two reactants which form two products, where each transition rule for a pair of reactants is weighted with probability 1. Given their suitability for modeling chemical processes, population protocols have been used to study computation not only by chemical reaction networks [CDS12], but also DNA strand displacement [CDS$^+$13, TWS15] and biochemical networks [CCN12].

These applications of population protocols in chemistry have inspired various adaptations of the model. In this chapter, we focus on two such variations on population protocols and then study them in the context of solving `Majority`, the problem of determining which of two states is initially more prevalent in a population.

The first modification to the model we consider, which was introduced to the literature in previous works studying source detection and bit-broadcast [DK18, ADK$^+$17] and later studied in the context of the majority problem [ATU21, dCN20], is the presence of **persistent-state agents**, or agents whose state never changes. While some works use persistent-state agents to model authoritative sources of information [DK18] or "stubborn" nodes that are unwilling to change state [dCN20], others describe these entities as an embodiment of chemical catalysts because they induce a state transition in another agent without themselves changing state [ADK$^+$17, ATU21]. Using the latter perspective, we refer to these persistent-state agents as **catalysts** (formally defined in Section 5.4.2).

In this work, we call the class of population protocols with catalysts the **catalytic input (CI) model**. We formally define the model to consist of n **catalytic input agents**, which in accordance with their name do not ever change state, and m **worker agents** that can change state and wish to compute some function on the states of the catalysts. While the CI model is similar to the standard population protocol model, we show that there exists a strong separation between the two in terms of their computational power.

The next variation on the model we consider is the introduction of transient leak events, studied previously in the contexts of solving source detection and comparison [ADK$^+$17, ATU21]. In brief, a "leak" simulates the low-probability event that a molecule undergoes a reaction that would typically take place in the presence of a catalyst. In population protocols, this is modeled by a spontaneous change of state at a single agent, and note that catalytic agents in the CI model are not susceptible to leaks because they never change state. A leak replaces an interaction between two agents at any given step with some fixed probability, known as the **leak rate** [ADK$^+$17]. Although leaks have typically been studied in the presence of catalysts, we consider leaks to more generally model unpredictable or adversarial behavior which may occur in the absence of catalysts as well.

We explore the impact of leaks on third-state dynamics [AAE08, PVV09] solving `Majority`. Our work demonstrates that the well-studied third-state dynamics can solve `ApproxMajority`, or `Majority` with a lower-bounded initial difference between the counts of the two input states, with upper-bounded leak rate both in the standard and CI population models. This indicates that the well-studied third-state dynamics process is robust to some degree of both faulty and adversarial behavior.

5.2 Related Work

The third-state dynamics protocol in the original population model (sometimes called *undecided-state dynamics*) was introduced in [AAE08] and independently in [PVV09]. An agent is either in a state X or Y, or in a *blank* state B (sometimes called an *undecided* state). The transition rules are shown in Figure 5.1 and we refer to this

Algorithm 5.1a: `DBAM`	**Algorithm 5.1b:** `DBAM-C`	
$X, B \to X, X$	$X, B \to X, X$	$I_X, B \to I_X, X$
$Y, B \to Y, Y$	$Y, B \to Y, Y$	$I_Y, B \to I_Y, Y$
$X, Y \to B, B$	$X, Y \to B, B$	

Transition rules for the `DBAM` protocol [AAFJ08] in the original population model, and our `DBAM-C` protocol in the CI model.

protocol as `DBAM`, which stands for *double-B approximate majority*. The terminology "*double-B*" captures the fact that following an $X + Y$ interaction, both agents transition to the B state. This protocol is the two-way variant of the original protocol from [AAE08], which uses one-way communication and where only one agent updates its state per pairwise interaction. In contrast to the original protocol from [AAE08], `DBAM` is **symmetric**, meaning that an agent's state update is independent of its role (i.e. Initiator versus Responder) in the interaction.

Assuming an initial X majority, a simplified analysis from [CHKM19] showed that all n agents in the population transition to the X state within $O(n \log n)$ total interactions with high probability, so long as the input margin $\big||X| - |Y|\big|$ at the start of the protocol is at least $\Omega(\sqrt{n \log n})$. The `DBAM` protocol is also robust to a small subset of faulty *Byzantine* agents [AAE08, CHKM19], meaning that all but a $O(\sqrt{n \log n}/n)$ fraction of the population still reaches the X state within $O(n \log n)$ interactions with high probability, despite the presence of these dishonest agents.

The `DBAM` protocol and similar variants of third-state dynamics have been shown to more generally compute `Consensus` (where all agents converge to either X or Y, but where this need not be the initial majority value), both in the original population protocols model [AAE08, CHKM19] and in other similar distributed models [BCN20, dCN20]. The closely related results of [dCN20] analyze an analogous version of the `DBAM` protocol in the synchronous PULL model, where the authors considered systems with *stubborn* agents (as in [YOA+13]) which are similar to the persistent-state catalytic agents we consider in the present work. However, the parallel synchronous scheduling model considered in [dCN20] is fundamentally distinct from the sequential pairwise scheduling used in population protocols.

The notion of a persistent *source* state in population protocols originated from [DK18], where sources are used to solve `Detection` (the detection of a source in the population) and `BitBroadcast` (the broadcast of a 0 or 1 message from a set of source agents). An accompanying work [ADK$^+$17] introduces the concept of leaks, or spontaneous state changes, and investigates the `Detection` problem in their presence. Generally, leaks can be dealt with using error-correcting codes [WTE$^+$18]; however, for certain problems there are more efficient specialized solutions. For example, `Detection` in the presence of leaks (up to rate $\beta = O(1/n)$) can be solved with high probability using $\log \frac{n}{k} + O(\log \log n)$ states, where $k \leq n$ is the number of sources in the population, as demonstrated by [ADK$^+$17].

More recently, [ATU21] examines leaks in the context of the `Comparison` problem. `Comparison` is a generalization of the `Majority` problem, where some possibly small subset of the population is in input state X_0 or Y_0 and the task of the population is to determine which of the two states is more prevalent. In [ATU21], `Comparison` is solved in $O(n \log n)$ interactions with high probability using $O(\log n)$ states per agent, assuming $|X_0| \geq C|Y_0|$ for some constant C, and $X_0, Y_0 \geq \Omega(\log n)$. The protocol is self-stabilizing, meaning that it dynamically responds to changes in the counts of input states.

5.3 Contribution

Based on models considered in recent protocols for populations with persistent-state agents [DK18, ADK$^+$17, ATU21], we assume a population with n catalytic input agents and m worker agents, and the goal of the worker agents is to compute some predicate over the states of the catalytic inputs. We call this model the Catalytic Input (CI) model. While conceptually similar to other models considering these types of catalytic agents [ADK$^+$17, ATU21, dCN20], introducing the distinction between the two (possibly unrelated) population sizes provides a new level of generality for designing and analyzing protocols in this setting, both with and without leaks.

For $m = \Theta(n)$, we show (Theorem 5.1) that computing the `Parity` of the input population with high probability[1] requires at least $\Omega(n^2)$ total interactions, demonstrating a strong separation between the CI model and the standard population protocol model. On the other hand, we show (Theorem 5.3) that the simple third-state dynamics [AAE08, PVV09] for `ApproxMajority` in the standard model can be naturally adapted to the CI model: we present such a constant-state protocol for the CI model that solves `ApproxMajority` in $O(n \log n)$ total steps with high probability when the input margin is $\Omega(\sqrt{n \log n})$.

In the `ApproxMajority` problem in the CI model, each catalytic input agent holds a persistent value of I_X or I_Y and each worker agent holds either an undecided, or blank value B, or an X or Y value corresponding to a belief in an I_X or I_Y input majority, respectively. The worker agents seek to correctly determine the larger of $|I_X|$ and $|I_Y|$ so long as the input margin $\big||I_X| - |I_Y|\big|$ is sufficiently large. By adapting the third-state dynamics process [AAE08], we present a constant-state protocol for approximate majority with catalytic inputs called `DBAM-C` (see Algorithms 5.1a and 5.1b). The protocol converges with high probability in $O(N \log N)$ total steps when the initial input margin is $\Omega(\sqrt{N \log N})$ and $m = \Theta(n)$. We then show (Theorem 5.2) that this input margin is optimal in the CI model up to a $O(\sqrt{\log N})$ factor when $m = \Theta(n)$.

Moreover, in the presence of transient leak events (as introduced in [ADK$^+$17, ATU21]), we show (Theorems 5.4 and 5.5) that both the third-state dynamics pro-

[1] We define "high probability" to mean with probability at least $1 - n^{-c}$ where n is the total number of agents and $c \geq 1$.

tocol in the original model and our adapted protocol in the CI model exhibit a strong robustness to leaks. When the probability β of a leak event is bounded below $O\left(\sqrt{n \log n}/n\right)$, we show that with high probability both protocols still quickly reach a configuration where nearly all agents share the correct input majority value.

Notice that the `ApproxMajority` problem in the CI model is equivalent to the comparison problem considered by [ATU21], so we demonstrate how our protocol compares to the results of this work. We show that our `DBAM-C` protocol converges correctly within the same time complexity of $O(n \log n)$ total steps, while only using *constant* state space (compared to the logarithmic state used by the protocols in their work). Moreover, in populations where $m = \Theta(n)$, our protocol tolerates a less restrictive bound on the input margin compared to [ATU21] ($\Omega(\sqrt{n \log n})$ compared to $\Omega(n)$). In the presence of transient leaks, our protocol also shows robustness to a higher leak rate of $\beta \leq O(\sqrt{n \log n}/n)$. However, unlike [ATU21], our protocol is not self-stabilizing and requires that the number of inputs be at least a constant fraction of the total population for our main results. In order to achieve these results, we leverage the random walk analysis techniques and analysis structure introduced by [CHKM19].

The resilience of these dynamics to leaks exhibits similarities to previous work involving Byzantine agents, and we define and prove a notion of equivalence between the two. We compare the impact of leaks on population protocols with that of faulty Byzantine processes. While the fast robust `ApproxMajority` protocol of [AAE08] is proven to be robust to a number of Byzantine agents that is bounded by the input margin [AAE08, CHKM19], we show (Theorem 5.6) that `DBAM` is robust to a similarly bounded leak rate and has sampling error matching the result from [CHKM19].

5.4 Extensions to the Standard Model

Population protocols [AAD$^+$06] are a class of algorithms for modeling distributed computation in networks of finite-state agents communicating through pairwise interactions. Their suitability for analyzing numerous chemical processes has motivated the adaptation of the original population protocol framework to better model these chemical systems. Here, we further the study of two such adaptations in the context of solving approximate majority: persistent-state agents (or *catalysts*) and spontaneous state changes (or *leaks*).

5.4.1 Sampling

In order to determine the success or failure of an execution of $\mathcal{P}$, we will consider a sample of the population to signify the outcome of the protocol [ADK$^+$17]. After the expected time to converge, one agent is selected at random and its state is observed. The output associated with the agent's state is considered the output of the protocol. The probability of sampling an agent whose state does not reflect the desired output of the protocol is called the **sample error rate**. Multiple samples can be aggregated to improve the rate of success.

5.4.2 Catalysts and Leaks

Following [ADK$^+$17], in a (nontrivial) interaction of the form $A, B \to A, D$, we say A *catalyzes* the transformation of the agent in state B to be in state D. If A catalyzes every interaction it participates in, A is referred to as a **catalyst**.

In chemistry, a reaction that occurs in the presence of a catalyst also occurs at a lower rate in the absence of that catalyst. For this reason, recent work in DNA strand displacement, chemical reactions networks, and population protocols [TWS15, ADK$^+$17, ATU21] have studied the notion of *leakage*: When a catalytic reaction $A, B \to A, D$ is possible, then there is some probability that a transition $B \to D$ can occur without interacting with A at all. This type of event, called a **leak**, was introduced in [TWS15].

The probability with which the non-catalyzed variation of a reaction takes place is the **leak rate**, which we denote by β. We simulate a leak as follows: At each step in time, with probability $1 - \beta$, the scheduler samples an ordered pair of agents to interact with one another as described in Section 2.1; the rest of the time (i.e. with probability β) one agent is chosen uniformly at random from all possible agents and the **leak function** $\ell : Q \to Q$ is applied to update this agent's state. Note that we only consider *non-catalytic* agents to be susceptible to these faulty events.

5.4.3 Catalytic Inputs

In this work, we formalize a **catalytic input** (CI) model consisting of n catalytic agents that supply the input and m worker agents that perform the computation and produce output. We define $N = m + n$ to be the total number of agents in the population. At each time step, the scheduler samples any two agents in the population to interact with one another. If two catalysts are chosen to interact, then the interaction is considered to be **null** as no nontrivial state transition occurs. When $n = O(m)$, the probability that two catalysts are chosen to interact is upper bounded by a constant, and so the total running time of the protocol is asymptotically equivalent to the number of non-null interactions needed to reach convergence. In the CI model, we consider convergence to be a term that refers to the states of the worker agents only, as the catalytic agents never change state. Namely, for the approximate majority problem, successful convergence equates to the *worker agents* being in the majority-accepting state. In general, we wish to obtain results that hold with high probability with respect to the *total* number of agents N.

5.5 Catalytic Input Model: Lower Bounds

In this section, we characterize the computational power of the CI population protocol model. Using information-theoretic arguments, we prove two lower bounds over the catalytic model when the number of input agents is a constant fraction of the total population:

Theorem 5.1. *In the catalytic input model with n input agents and $m = \Theta(n)$ worker agents, any protocol that computes the `Parity` of the inputs with probability at least $1 - N^{-\gamma}$ requires at least $\Omega(N^2)$ total steps for any $\gamma \geq 1$.*

Theorem 5.2. *In the catalytic input model with n input agents and $m = \Theta(n)$ worker agents, any protocol that computes the `Majority` of inputs within $O(N \log N)$ total steps requires an input margin of at least $\Omega(\sqrt{N})$ to be correct with probability at least $1 - N^{-\gamma}$ for any $\gamma \geq 1$.*

The first result can be viewed as a separation between the CI and original population models: since it is shown in [KU18] that the `Parity` of agents can be computed in the original model within $O(\operatorname{polylog} n)$ parallel time with high probability, our result indicates that not all semi-linear predicates over the input population in the CI model can be computed in sub-linear parallel time with high probability. Additionally, this rules out the possibility of designing fast protocols for *exact* majority in the CI model when the input size is a constant fraction of the entire population because computing exact majority is as hard as computing `Parity` for odd n when the initial margin of the majority opinion is exactly 1 (see Subsection 5.5.1). On the other hand, the second result indicates the existence of a predicate — `ApproxMajority` — that does not require a large increase in convergence time to be computed with high probability in this new model.

One key characteristic of a CI population is the inability for worker agents to distinguish which inputs have previously interacted with a worker. Instead, every worker-input interaction acts like a random sample with replacement from the input population. For proving lower bounds in this model, this characteristic of a CI population leads to the following natural argument: consider a population of n catalytic input agents and a worker population consisting of a single **super-agent**. Here, we assume the super-agent has unbounded state and computational power, and it is thus able to simulate the entire worker population of any protocol with more workers. In this simulation, any interaction between a worker and an input agent is equivalent to the super-agent interacting with an input chosen uniformly at random: in other words, as a sample with replacement from the input population. Thus we view the super-agent as running a central randomized algorithm to simulate the random interactions that occur in population protocols. If the super-agent needs s samples to compute some predicate over the inputs with high probability, then so does any multi-worker protocol in the CI model. We denote this information-theoretic model as the **Super CI model**, and restate the above argument more formally in the following lemma.

Lemma 5.1. *Consider a population with n catalytic input agents and a worker population consisting of a single super-agent W. Let P be a predicate over the input population that requires s total interactions between W and the input population in order for W to correctly compute P with probability ϵ. Then for a CI population with n catalytic inputs and m worker agents, computing P correctly with probability ϵ requires at least s total interactions.*

5.5.1 Proof of Theorem 5.1

In a CI model population with n input agents and m worker agents where $m = \Theta(n)$, Theorem 5.1 shows that computing the parity or exact majority of the inputs requires at least $\Omega(n^2) = \Omega(N^2)$ total interactions to be correct with high probability. We prove this by showing that in the Super CI model described in the previous subsection, a computationally unbounded super-agent W requires at least $\Omega(n^2)$ samples of the input population to correctly compute the input parity with high probability. Applying Lemma 5.1 then gives Theorem 5.1.

Relationship Between Majority and Parity

Formally, for an input population P of n agents, each with input value 0 or 1, the `Parity` of P is said to be 1 if an odd number of agents have input value 1, and 0 otherwise. Now, consider the `Majority` predicate over P, which is simply the majority value of the input population. Letting X denote the 1-inputs, and Y the 0-inputs, we refer to the *input margin* of the population P as the quantity $\big||X|-|Y|\big|$. Suppose that n is odd and the input margin of P is 1. Then X and Y are either $\left\lfloor\frac{n}{2}\right\rfloor$ and $\left\lceil\frac{n}{2}\right\rceil$ or vice versa. These two cases can be distinguished either by computing the `Majority` predicate or the `Parity` predicate, making both of these problems equivalent to distinguishing the two cases under this constraint on the input. We will now argue that distinguishing these cases in the Super CI model requires $\Omega(n^2)$ samples.

Optimality of the Sample Majority Map

Recall that in the Super CI model, a predicate over the input population P is computed by a single super agent worker W with unbounded computational power. Thus, the output of W can be viewed as a mapping between a string of input values obtained from interactions between W and the input population and the output set $\{0, 1\}$. We refer to interactions between W and the input population as **samples** of the input, and for a fixed number of samples s, we refer to W's output as its **strategy**.

First, we show that for some fixed distribution over the input values of P, the strategy that maximizes W's probability of correctly outputting the majority value of P is simply to output the majority value of its samples. Let $\boldsymbol{S} \in \{0, 1\}^s$ be the

sample string representing the s independent samples with replacement taken by W, and let $\mathbb{S}$ denote the set of all 2^s possible sample strings. We model the population of input agents as being generated by an adversary. Specifically, let $\boldsymbol{M}$ denote the majority value (0 or 1) of the input population, where we treat $\boldsymbol{M}$ as a random variable whose distribution is unknown. In any realization of $\boldsymbol{M}$, we assume a fixed fraction $p > 1/2$ of the inputs hold the majority value.

Given an input population, the objective of the worker agent is to correctly determine the value of $\boldsymbol{M}$ through its input sample string $\boldsymbol{S}$. By Yao's principle [Yao77], the error of any randomized algorithm (i.e., the randomized simulation run by the super-agent) on the worst case value of $\boldsymbol{M}$ is no smaller than the error of the best deterministic algorithm on some fixed distribution over $\boldsymbol{M}$. So our strategy is to pick a distribution over $\boldsymbol{M}$, and to use the the error of the best deterministic strategy with respect to this distribution as a lower bound on the worst-case error of any randomized algorithm used by the super-agent.

Thus, assuming $\boldsymbol{M}$ is chosen according to some fixed distribution, we model the worker's strategy as a fixed map $f : \{0,1\}^s \to \{0,1\}$. Letting $\mathcal{F}_s$ denote the set of all such maps, W then faces the following optimization problem: $\max_{f \in \mathcal{F}_s} \Pr(f(\boldsymbol{S}) = \boldsymbol{M})$. For a given $f \in \mathcal{F}_s$, let $p_f = \Pr(f(\boldsymbol{S}) = \boldsymbol{M})$, and let $\Phi \in \mathcal{F}_s$ denote the map that outputs the majority value of the input sample string $\boldsymbol{S}$. In the following lemma, we show that when the distribution over $\boldsymbol{M}$ is uniform, setting $f := \Phi$ maximizes p_f. In other words, to maximize the probability of correctly guessing the input population majority value, the worker's optimal strategy is to simply guess the majority value of its s independent samples. The proof of the lemma simply uses the definitions of conditional probability and the Law of Total Probability to obtain the result.

Lemma 5.2. *Let $\boldsymbol{S} \in \{0,1\}^s$ be a sample string of size s drawn from an input population with majority value $\boldsymbol{M}$ and majority ratio p, and assume* $\Pr(\boldsymbol{M} = 1) = \Pr(\boldsymbol{M} = 0) = 1/2$. *Then* $\Pr(\Phi(\boldsymbol{S}) = \boldsymbol{M}) \geq \Pr(f(\boldsymbol{S}) = \boldsymbol{M})$ *for all maps $f \in \mathcal{F}_s$,* *where Φ is the map that outputs the majority value of the sample string $\boldsymbol{S}$.*

Proof. Recall that we model the majority value of the input population $\boldsymbol{M}$ as a 0-1 random variable. Assume here that the distribution of $\boldsymbol{M}$ is fixed, and that $\Pr(\boldsymbol{M} = 0) = \Pr(\boldsymbol{M} = 1) = 1/2$.

For any map $f \in \mathcal{F}_s$, we can compute $p_f = \Pr(f(\boldsymbol{S}) = \boldsymbol{M})$ by

$$\begin{aligned} p_f &= \Pr(f(\boldsymbol{S}) = \boldsymbol{M}) \\ &= \Pr(f(\boldsymbol{S}) = \boldsymbol{M}, \boldsymbol{M} = 0) + \Pr(f(\boldsymbol{S}) = \boldsymbol{M}, \boldsymbol{M} = 1) \\ &= \Pr(\boldsymbol{M} = 0) \cdot \Pr(f(\boldsymbol{S}) = \boldsymbol{M} \mid \boldsymbol{M} = 0) + \Pr(\boldsymbol{M} = 1) \cdot \Pr(f(\boldsymbol{S}) = \boldsymbol{M} \mid \boldsymbol{M} = 1) \\ &= \frac{1}{2} \cdot \Pr(f(\boldsymbol{S}) = \boldsymbol{M} \mid \boldsymbol{M} = 0) + \frac{1}{2} \cdot \Pr(f(\boldsymbol{S}) = \boldsymbol{M} \mid \boldsymbol{M} = 1) \end{aligned} \tag{5.1}$$

where the last inequality follows from assuming $\Pr(\boldsymbol{M} = 0) = \Pr(\boldsymbol{M} = 1) = 1/2$. Recall that $\boldsymbol{S} \in \{0,1\}^s$ is the input string of s independent samples from the input population, and $\mathbb{S}$ is the set of all possible values of $\boldsymbol{S}$. Thus for any $f \in \mathcal{F}_s$, the law of total probability gives

$$\begin{aligned} \Pr(f(\boldsymbol{S}) = \boldsymbol{M} \mid \boldsymbol{M} = 0) &= \sum_{t \in \mathbb{S}} \Pr(f(\boldsymbol{S}) = \boldsymbol{M}, \boldsymbol{S} = t \mid \boldsymbol{M} = 0) \\ &= \sum_{t \in \mathbb{S}} \Pr(f(t) = \boldsymbol{M}, \boldsymbol{S} = t \mid \boldsymbol{M} = 0) \\ &= \sum_{t \in \mathbb{S}} \Pr(f(t) = \boldsymbol{M} | \boldsymbol{M} = 0) \cdot \Pr(\boldsymbol{S} = t | f(t) = \boldsymbol{M}, \boldsymbol{M} = 0) \end{aligned}$$

Since the events $\boldsymbol{S} = t$ and $f(t) = \boldsymbol{M}$ are independent, $\Pr(\boldsymbol{S} = t \mid f(t) = \boldsymbol{M}, \boldsymbol{M} = 0) = \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 0)$ for every $t \in \mathbb{S}$. Furthermore, given that every $f \in \mathcal{F}_s$ is a deterministic map, we can rewrite

$$\Pr(f(t) = \boldsymbol{M} \mid \boldsymbol{M} = 0) = \Pr(f(t) = 0) = \mathbf{1}_{\{f(t)=0\}},$$

where $\mathbf{1}_{\{f(t)=0\}}$ is the indicator random variable of the event $f(t) = 0$. Thus for any

$f \in \mathcal{F}_s$ and every $t \in \mathbb{S}$ we have

$$\Pr(f(t) = \boldsymbol{M} \mid \boldsymbol{M} = 0) = \sum_{t \in \mathbb{S}} \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 0) \cdot \mathbf{1}_{\{f(\boldsymbol{S})=0\}}$$

It can be similarly shown that

$$\Pr(f(t) = \boldsymbol{M} \mid \boldsymbol{M} = 1) = \sum_{t \in \mathbb{S}} \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 1) \cdot \mathbf{1}_{\{f(t)=1\}}$$

for every $t \in \mathbb{S}$ and a fixed $f \in \mathcal{F}_s$. Thus substituting back into (5.1) gives

$$\begin{aligned} p_f &= \Pr(f(\boldsymbol{S}) = \boldsymbol{M}) \\ &= \frac{1}{2} \sum_{t \in \mathbb{S}} \big(\Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 0) \cdot \mathbf{1}_{\{f(\boldsymbol{S})=0\}} + \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 1) \cdot \mathbf{1}_{\{f(\boldsymbol{S})=1\}} \big) \end{aligned}$$

Now, let $\mathbb{S}_0 \subset \mathbb{S}$ denote the set of of input sample strings $\{0,1\}^s$ with a 0-majority, and let $\mathbb{S}_1 \subset \mathbb{S}$ denote the set of sample strings with a 1-majority. Thus $\mathbb{S}_0$ and $\mathbb{S}_1$ are disjoint and $\mathbb{S}_0 \cup \mathbb{S}_1 = \mathbb{S}$. Additionally, for a fixed $f \in \mathcal{F}_s$ and any $t \in \mathbb{S}$ define $\alpha(f,t)$ by

$$\alpha(f,t) = \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 0) \cdot \mathbf{1}_{\{f(t)=0\}} + \Pr(\boldsymbol{S} = t \mid \boldsymbol{M} = 1) \cdot \mathbf{1}_{\{f(t)=1\}}$$

Thus for a fixed $f \in \mathcal{F}_s$ we can again rewrite

$$\Pr(f(\boldsymbol{S}) = \boldsymbol{M}) = \frac{1}{2} \left(\sum_{t \in \mathbb{S}_0} \alpha(f,t) + \sum_{t \in \mathbb{S}_1} \alpha(f,t) \right) \tag{5.2}$$

Recall that $\Phi \in \mathcal{F}_s$ is the map that outputs the majority value of the input sample string $\boldsymbol{S} \in \{0,1\}^s$. Fix any other map $f \neq \Phi \in \mathcal{F}_s$. Since $f \neq \Phi$, there exists at least one string $t \in \mathbb{S}$ such that $f(t) \neq \Phi(t)$, and assume without loss of generality that $t \in \mathbb{S}_0$. By definition, this means $\Phi(t) = 0$ and $f(t) = 1$. Using the definition of $\alpha(f,t)$, and recalling that $\Pr(t_i = 0 \mid \boldsymbol{M} = 1) = 1-p < 0.5$ and $\Pr(t_i = 1 \mid \boldsymbol{M} = 1) = p > 0.5$

are the probabilities that a single sample of t is 0 or 1 respectively, we have

$$\begin{aligned}
\alpha(f,t) &= \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=0)\cdot \mathbf{1}_{\{f(t)=0\}} + \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=1)\cdot \mathbf{1}_{\{f(t)=1\}} \\
&= \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=0)\cdot 0 + \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=1)\cdot \mathbf{1}_{\{f(t)=1\}} \\
&= \Pr(t_i=0 \mid \boldsymbol{M}=1)^k \cdot \Pr(t_i=1 \mid \boldsymbol{M}=1)^{s-k} \\
&= (1-p)^k \cdot p^{s-k}
\end{aligned}$$

where $s/2 \le k < s$ since $t \in \mathbb{S}_0$. Meanwhile, for the same $t \in \mathbb{S}_0$, using the majority sample map Φ gives

$$\begin{aligned}
\alpha(\Phi,t) &= \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=0)\cdot \mathbf{1}_{\{\Phi(t)=0\}} + \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=1)\cdot \mathbf{1}_{\{\Phi(t)=1\}} \\
&= \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=0)\cdot \mathbf{1}_{\{\Phi(t)=0\}} + \Pr(\boldsymbol{S}=t \mid \boldsymbol{M}=1)\cdot 0 \\
&= \Pr(t_i=0 \mid \boldsymbol{M}=0)^k \cdot \Pr(t_i=1 \mid m=0)^{s-k} \\
&= p^k \cdot (1-p)^{s-k}
\end{aligned}$$

where again $s/2 \le k < s$ since $t \in \mathbb{S}_0$. Since by definition $p > 1/2$, it follows that $\alpha(\Phi,t) > \alpha(f,t)$ for any $f \in \mathcal{F}_s$ where $f \neq \Phi$, and for any $t \in \mathbb{S}_0$ where $f(t) \neq \Phi(t)$. It can similarly be shown that $\alpha(\Phi,t) > \alpha(f,t)$ for any $t \in \mathbb{S}_1$ with $f(t) \neq \Phi(t)$. By the definition of $\Pr(f(\boldsymbol{S}) = \boldsymbol{M})$ from (5.2), it follows that $\Pr(f(\boldsymbol{S}) = \boldsymbol{M}) < \Pr(\Phi(\boldsymbol{S}) = \boldsymbol{M})$ for any $f \neq \Phi \in \mathcal{F}_s$, thus proving the claim. □

Sample Lower Bound for Majority With Input Margin 1

We have established by Lemma 5.2 that to correctly output the input population majority, the super worker agent's error-minimizing strategy is to output the majority of its s samples when the distribution over $\boldsymbol{M}$ is uniform. Now the following lemma shows that when the input margin of the population is 1, this strategy requires at least $\Omega(n^2)$ samples in order to output the input majority with probability at least $1 - n^{-c}$ for some constant $c \ge 1$. The proof uses a tail bound on the Binomial distribution to show the desired trade-off between the error probability and the requisite number of samples needed to achieve this error.

Lemma 5.3. *Let P be a Super CI population of n agents with majority value $\boldsymbol{M}$ and input margin 1, and consider an input sample string $\boldsymbol{S} \in \{0,1\}^s$ obtained by a super worker agent W. Then for any $c \geq 1$, letting $\Phi(\boldsymbol{S})$ denote the sample majority of $\boldsymbol{S}$, $\Pr(\Phi(\boldsymbol{S}) \neq \boldsymbol{M}) \leq n^{-c}$ only holds when $s \geq \Omega(n^2)$.*

Proof. Let $\boldsymbol{X_s}$ be a random variable representing the sum of a sequence of s Binomial trials with parameter p. This $\boldsymbol{X_s}$ denotes the number of samples matching the majority input value. Without loss of generality, we will assume $\boldsymbol{M} = 1$, meaning that $p = 1/2 + 1/2n$. Since $\Pr(\Phi(\boldsymbol{S}) \neq \boldsymbol{M}) = \Pr(\boldsymbol{X_s} \leq s/2)$, we will prove that $s \geq \Omega(n^2)$ is a necessary constraint to satisfy $\Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$.

Here $\Pr(\boldsymbol{X_s} \leq s/2)$ is just the lower tail of the cumulative distribution function (CDF) of a Binomial distribution with parameter p. Thus when $p = 1/2 + \delta$ for $\delta > 0$, we have the following lower bound on $\Pr(\boldsymbol{X_s} \leq s/2)$ (see [Ash90]):

$$\Pr(\boldsymbol{X_s} \leq s/2) \geq \frac{1}{\sqrt{2s}} \cdot \exp\left(-s \cdot D_{KL}(\tfrac{1}{2} \mid\mid p)\right) \tag{5.3}$$

Here, D_{KL} denotes the Kullback-Leibler (KL) divergence between a fair coin and a Bernoulli random variable with bias p. This can be rewritten as

$$\begin{aligned} D_{KL}(\tfrac{1}{2} \mid\mid p) &= \frac{1}{2} \cdot \log \frac{1/2}{p} + \frac{1}{2} \cdot \log \frac{1/2}{1-p} \\ &= \frac{1}{2} \cdot \log \frac{1}{4p \cdot (1-p)} \\ &\leq 4\delta^2 \end{aligned} \tag{5.4}$$

where the last inequality holds for $0 < \delta \leq 1/3$.

Substituting (5.4) into (5.3) then gives

$$\Pr(\boldsymbol{X_s} \leq s/2) \geq \frac{1}{\sqrt{2s}} \cdot \exp\left(-4s\delta^2\right)$$

and since we are assuming $p = 1/2 + 1/2n$, we have

$$\Pr(\boldsymbol{X_s} \leq s/2) \geq \frac{1}{\sqrt{2s}} \cdot \exp\left(-\frac{s}{n^2}\right)$$

where $\delta = 1/2n \leq 1/3$ for all $n \geq 2$. Thus to ensure $\Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$, it is necessary to have $(1/\sqrt{2s}) \cdot \exp\left(-s/n^2\right) \leq n^{-c}$.

Taking natural logarithms then yields the following constraint on s:

$$\frac{s}{n^2} + 0.5 \log s + 0.5 \geq c \cdot \log n \tag{5.5}$$

We now want to show $s \geq \Omega(n^2)$ is needed to satisfy (5.5). To do this, consider any $s' = o(n^2)$. Observe then that $s'/n^2 = o(1)$ and $0.5 \log s' < \log n$. It follows that

$$\begin{aligned}\frac{s'}{n^2} + 0.5 \log s' + 0.5 &< o(1) + \log n \\ &\leq c \log n\end{aligned}$$

where the final inequality necessarily holds for all $c \geq 1$ for large enough n.

Thus no value $s = o(n^2)$ can satisfy the necessary condition of (5.5), which means that we must have $s \geq \Omega(n^2)$ in order to ensure $\Pr(\boldsymbol{X_s} \leq s/2) = \Pr(\Phi(\boldsymbol{X_s}) \neq \boldsymbol{M}) \leq n^{-c}$ holds for any $c \geq 1$. □

The proof of Theorem 5.1 (which is restated for convenience) follows from Lemmas 5.1, 5.2, and 5.3 by invoking Yao's principle.

Theorem 5.1. *In the catalytic input model with n input agents and $m = \Theta(n)$ worker agents, any protocol that computes the `Parity` of the inputs with probability at least $1 - N^{-\gamma}$ requires at least $\Omega(N^2)$ total steps for any $\gamma \geq 1$.*

Proof. By Lemmas 5.2 and 5.3 and using Yao's principle, in the Super CI model with an input population P of n agents and input margin 1, a single super-agent worker W can only compute the majority value of P with high probability by taking at least $s = \Omega(n^2)$ input samples in the worst case. By Lemma 5.1, this means that in the regular CI model with an input population of n agents, any protocol for Majority with input margin 1 requires at least $\Omega(n^2)$ total steps to be computed correctly with probability at least $1-n^{-c}$. Because computing Majority on a population with input margin 1 reduces to computing Parity over this input, we have that any protocol for Parity in the CI model requires at least $\Omega(n^2)$ total steps in the worst-case to be correct with probability at least $1-n^{-c}$. When the size of the worker population is $m = \Theta(n)$, this means that $N = m + n = \Theta(n)$. Thus for an appropriate choice of c, computing Parity on such populations requires at least $\Omega(N^2)$ samples to be correct with probability at least $1 - N^{-\gamma}$ for any $\gamma \geq 1$. □

5.5.2 Proof of Theorem 5.2

As mentioned, Theorem 5.1 implies a strong separation between the CI model and original population model, as [KU18] and [AAE$^+$17, AAG18b] have shown that both Parity and Majority are computable with high probability within $O(n \operatorname{polylog} n)$ total steps in the original model, respectively. Thus, the persistent-state nature of input agents in the CI model may seem to pose greater challenges than in the original model for computing predicates quickly with high probability. However, using the same sampling-based lower bound techniques developed in the preceding section, Theorem 5.2 shows that when $m = \Theta(n)$, and when restricted only to $O(n \log n)$ total steps, any protocol computing majority in the CI model requires an input margin of at least $\Omega(\sqrt{n}) = \Omega(\sqrt{N})$ to be correct with high probability in N.

Moreover, in Section 5.6 we present a protocol for approximate majority in the CI model that converges correctly with high probability within $s = O(N \log N)$ total steps, so long as the initial input margin is $\Omega(\sqrt{N \log N})$. Thus, the existence of such a protocol indicates that the $\Omega(\sqrt{N})$ lower bound on the input margin is nearly tight (up to a $\sqrt{\log N}$ factor) for protocols limited to $O(N \log N)$ total steps when $m = \Theta(n)$.

Input Margin Lower Bound for Majority

We return to the Super CI model and use the same notation developed in Section 5.5.1. We want to show that when $s = O(n \log n)$, we must have $p \geq 1/2 + \Omega(1/\sqrt{n})$ in order to ensure $\Pr(\Phi(\boldsymbol{S}) \neq \boldsymbol{M}) = \Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$. This lower bound on p (the proportion of 1-agents, without loss of generality, in the input population) corresponds to an input margin lower bound of $\Omega(\sqrt{n})$.

Lemma 5.4. *Assume a 0-1 population of n agents with majority value $\boldsymbol{M}$ and majority proportion $p = 1/2 + \delta$, and consider an input sample string $\boldsymbol{S} \in \{0,1\}^s$ where $s = O(n \log n)$. Then for any $c \geq 1$, $\Pr(\Phi(\boldsymbol{S}) \neq \boldsymbol{M}) \leq n^{-c}$ only holds when $\delta \geq \Omega(1/\sqrt{n})$, where Φ is the map that outputs the majority value of $\boldsymbol{S}$.*

Proof. Again without loss of generality, assume $\boldsymbol{M} = 1$. Since $\Pr(\Phi(\boldsymbol{S}) \neq \boldsymbol{M}) = \Pr(\boldsymbol{X_s} \leq s/2)$, we will show that $\delta \geq \Omega(1/\sqrt{n})$ is a necessary condition to have $\Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$ when $s = O(n \log n)$.

Recall the lower bound on $\Pr(\boldsymbol{X_s} \leq s/2)$ from Lemma 5.3:

$$\Pr(\boldsymbol{X_s} \leq s/2) \geq \frac{1}{\sqrt{2s}} \cdot \exp\left(-4s\delta^2\right) \tag{5.6}$$

which holds for $0 < \delta \leq 1/3$. (Note that when $\delta = 1/\sqrt{n}$, $\delta \leq 1/3$ for all $n \geq 9$).

Setting $s = an \log n$ for some $a > 0$ lets us write (5.6) as

$$\Pr(\boldsymbol{X_s} \leq s/2) \geq \frac{1}{\sqrt{2an \log n}} \cdot \exp\left(-4an \log n \cdot \delta^2\right)$$

which means it is necessary to have $(1/\sqrt{2an \log n}) \cdot \exp(-4an \log n \cdot \delta^2) \leq n^{-c}$ to ensure that $\Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$.

Again by taking natural logarithms, we find that we require

$$4an \log n \cdot \delta^2 + 0.5 \log n + 0.5 \log \log n + 0.5 \log a + 0.5 \geq c \log n \tag{5.7}$$

To show that $\delta \geq \Omega(1/\sqrt{n})$ is needed to satisfy (5.7), we use a similar strategy as in Lemma 5.3 and consider any $\delta' = o(1/\sqrt{n})$. This would imply $4an\log n \cdot (\delta')^2 = o(4a\log n) = o(\log n)$, and since $0.5\log\log n = o(\log n)$ and $a > 0$ is a constant, we have

$$\begin{aligned} 4an\log n \cdot (\delta')^2 + 0.5\log n + 0.5\log\log n + 0.5\log a + 0.5 &= 0.5\log n + o(\log n) \\ &< c\log n \end{aligned}$$

where the final equality will hold for all $c \geq 1$ and large enough n.

Thus if $\delta = o(1/\sqrt{n})$, then the necessary condition (5.7) will be violated, meaning that that we must have $\delta = \Omega(1/\sqrt{n})$ to ensure $\Pr(\boldsymbol{X_s} \leq s/2) \leq n^{-c}$ holds for any $c \geq 1$ when $s = O(n\log n)$. Since we defined $p = 1/2+\delta$, this corresponds to requiring an input margin of at least $\Omega(\sqrt{n})$ when $s = O(n\log n)$. □

We now formally prove Theorem 5.2, which is restated for convenience.

Theorem 5.2. *In the catalytic input model with n input agents and $m = \Theta(n)$ worker agents, any protocol that computes the* `Majority` *of inputs within $O(N\log N)$ total steps requires an input margin of at least $\Omega(\sqrt{N})$ to be correct with probability at least $1 - N^{-\gamma}$ for any $\gamma \geq 1$.*

Proof. By Lemmas 5.2 and 5.4, in the Super CI model with an input population P of size n, computing the majority of inputs correctly in $s = O(n\log n)$ samples with probability at least $1 - n^{-c}$ requires an input margin of at least $\Omega(\sqrt{n})$. By an argument similar to Lemma 5.1 and Theorem 5.1, note that this implies that any protocol that computes `Majority` in the regular CI model within $O(n\log n)$ total steps also requires an input margin of $\Omega(\sqrt{n})$ to be correct with probability at least $1 - n^{-c}$. Now consider that the size of the worker population is $m = \Theta(n)$, which means that $N = m + n = \Theta(n)$. This implies that for an appropriate choice of constant c and taking only $O(N\log N)$ total steps, the input margin must be at least $\Omega(\sqrt{N})$ in order for `Majority` to be computed correctly with probability at least

$1 - N^{-\gamma}$ for any $\gamma \geq 1$. □

5.6 Approximate Majority with Catalytic Inputs

We now present and analyze the DBAM-C protocol for computing approximate majority in the CI model. The protocol is a natural adaptation of the third-state dynamics from the original model, where we now account for the behavior of n catalytic input agents and m worker agents. Using the CI model notation introduced in 5.4.3, we consider a population with $N = n + m$ total agents. Each input agent begins (and remains) in state I_X or I_Y, and we assume each worker agent begins in a *blank* state B, but may transition to states X or Y according to the transition rules found in Figure 5.1. Letting i_X and i_Y (and similarly x, y and b) be random variables denoting the number of agents in states I_X and I_Y (and respectively X, Y, and B), we denote the *input margin* of the population by $\epsilon = |i_X - i_Y|$. Throughout the section, we assume without loss of generality that $i_X \geq i_Y$.

Intuitively, an undecided (blank) worker agent adopts the state of a decided agent (either an input or worker), but decided workers only revert back to a blank state upon interactions with other workers of the opposite opinion. Thus the protocol shares the opinion-spreading behavior of the original DBAM protocol, but note that the inability for decided worker agents to revert back to the blank state upon subsequent interactions with an input allows the protocol to converge to a configuration where all workers share the same X or Y opinion.

The main result of the section characterizes the convergence behavior of the DBAM-C protocol when the input margin ϵ is sufficiently large. Recall that we say the protocol *correctly computes* the Majority of the inputs if we reach a configuration where $x = m$. The following theorem shows that, subject to mild constraints on the population sizes, when the input margin is $\Omega(\sqrt{N \log N})$, the protocol correctly computes the majority value of the inputs in roughly logarithmic parallel time with high probability.

Theorem 5.3. *There exists some constant $\alpha \geq 1$ such that, for a population of n inputs, m workers, and initial input margin $\epsilon \geq \alpha\sqrt{N \log N}$, the `DBAM-C` protocol correctly computes the majority value of the inputs within $O\left(\frac{N^4}{m^3} \log N\right)$ total interactions with probability at least $1 - N^{-c}$ for any $c \geq 1$ when $m \geq n/10$ and N is sufficiently large.*

Because the CI model allows for distinct (and possibly unrelated) input and worker population sizes, we aim to characterize all error and success probabilities with respect to the *total* population size N. The analysis in the proof of Theorem 5.3 characterizes the convergence behavior of the protocol in terms of both population sizes m and n, and thus the convergence time of $O((N^4/m^3) \log N)$ is not always equivalent to $O(N \log N)$. On the other hand, in the case when $m = \Theta(n)$ — which is an assumption used to provide lower bounds over the CI model from Section 5.5 — we have as a corollary that the protocol correctly computes the majority of the inputs within $O(N \log N)$ total steps with probability at least $1 - N^{-\alpha}$.

5.6.1 Analysis Overview

The proof of the main result leverages and applies the random walk tools from [CHKM19] (in their analysis of the original `DBAM` protocol) to the `DBAM-C` protocol. Given the uniformly-random behavior of the interaction scheduler, the random variables x, y and b (which represent the count of X, Y, and B worker agents in the population) each behave according to some one-dimensional random walk, where the biases in the walks change dynamically as the values of these random variables fluctuate. Based on the coupling principle that an upper bound on the number of steps for a random walk with success probability p to reach a certain position is an *upper bound* on the step requirement for a second random walk with probability $\hat{p} \geq p$ to reach the same position, we make use of several *progress measures* that give the behavior of the protocol a natural structure. As used in the analysis in [CHKM19], we define $\hat{x} = x + b/2$, $\hat{y} = y + b/2$, and $\Delta = \epsilon + \hat{x} - \hat{y}$. It can be easily seen that $\hat{x} + \hat{y} = m$ will hold throughout the protocol. On the other hand, the progress measure Δ captures the collective gap between the majority and non-majority opinions in the population. Observe that the protocol has correctly computed the input majority value when $\Delta = \epsilon + m$ and $\hat{y} = 0$.

Our analysis uses a structure of **phases** and **stages** to prove the correctness and efficiency of the protocol. Every correctly-completed stage of Phase 1 results in the progress measure Δ doubling, and the phase completes correctly once Δ is at least ϵ plus some large constant fraction of m. Then, every correctly-completed stage of Phase 2 results in the progress measure $\hat{y}$ decreasing by a factor of two, and the phase completes correctly once $\hat{y}$ drops to $O(\log m)$. Finally, Phase 3 of the protocol ends correctly once $\hat{y}$ drops to 0. The details of this structure are stated formally in [AAL20] (wherein the progress measure is referred to as P).

Note that among the protocol's non-null transitions (see Algorithm 5.1b), only the interactions $I_X + B$, $I_Y + B$, $X + B$, and $Y + B$ change the value of either progress measure. For this reason, we refer to the set of non-null transitions (which includes $X + Y$ interactions) as **productive** steps, and the subset of interactions that change our progress measures as the set of **blank-consuming** productive steps. The analysis strategy for every phase and stage is to employ a combination of standard Chernoff bounds and martingale techniques (in general, see [GS01] and [Fel68]) to obtain high-probability estimates of (1) the number of productive steps needed to complete each phase/stage correctly, and (2) the number of total steps needed to obtain the productive step requirements. Given an input margin that is sufficiently large, and also assuming a population where the number of worker agents is at least a small constant fraction of the input size, we can then sum over the error probabilities of each phase/stage and apply a union bound to yield the final result of Theorem 5.3.

While the `DBAM-C` protocol is conceptually similar to the original `DBAM` protocol, the presence of persistent-state catalysts whose opinions never change requires a careful analysis of the convergence behavior. Moreover, simulation results presented in Figure 5.1 show interesting differences in the evolution of the protocol for varying population sizes. As a simple corollary of Theorem 5.3, we also state the following result, which simplifies the convergence guarantees of the `DBAM-C` protocol in the case when $m = \Theta(n)$.

Corollary 5.1. *There exists some constant $\alpha \geq 1$ such that, for a population of n inputs, $m = cn$ workers where $c \geq 1$, and an initial input margin $\epsilon \geq \alpha\sqrt{N \log N}$, the `DBAM-C` protocol correctly computes the majority of the inputs within $O(N \log N)$ total interactions with probability at least $1 - N^{-a}$ for any $a \geq 1$ when N is sufficiently large.*

We note that the result of Corollary 5.1 implies that the CI model input margin lower bound from Theorem 5.2 is tight up to a multiplicative $O(\sqrt{\log N})$ factor.

5.7 Approximate Majority with Transient Leaks

We now consider the behavior of the `DBAM` and `DBAM-C` protocols in the presence of transient leak faults. Even in the presence of these adversarial events (which occur up to some bounded rate β), both the `DBAM` and `DBAM-C` protocols will, with high probability, reach configurations where nearly all agents share the input majority opinion. In the presence of leaks, we consider the approximate majority predicate to be computed correctly upon reaching these low sample error configurations.

Recall that a transient leak is an event where an agent spuriously changes its state according to some leak function ℓ. For example, we denote by $U \to V$ the event that an agent in state U transitions to state V due to a leak event, where the timing of such events are dictated by the random scheduler and occur with probability β at each step. In both the DBAM and DBAM-C protocols, the only state changes that could possibly take place due to leaks are $X \to B$, $Y \to B$, $B \to X$, and $B \to Y$ because these describe all possible state changes that could take place in the presence of an interacting partner. However, our analysis considers an *adversarial* leak event $X \to Y$, which maximally decreases our progress measures and can be considered the "worst" possible leak. Though this leak event is not chemically sound (because no normal interaction can cause an X agent to transition to the Y state), our results demonstrate that both DBAM and DBAM-C protocols are robust to this strong adversarial leak event. Thus in a more realistic, chemically sound setting, our results will also hold, as the set of transitions working against our progress measures are weaker.

5.7.1 Leak Robustness of the DBAM Protocol

We start by showing the leak-robustness of the DBAM protocol for approximate majority in the original population protocol model. Recall that in the standard model, *all* agents are susceptible to leaks. Our main result shows that when the leak rate β is sufficiently small, the protocol still reaches a configuration with bounded *sample error* (the proportion of agents in the non-initial-majority state) within $O(n \log n)$ total interactions with high probability. Unlike the scenario without leak events, the protocol will never be able to fully converge to a configuration where all agents remain in the majority opinion. However, reaching a configuration where despite leaks, *nearly* all agents hold the input majority value matches similar results of [AAE08, CHKM19, ADK+17, ATU21]. Formally, we have the following theorem which characterizes the eventual sample error of the protocol with respect to the magnitude of the leak rate β.

Theorem 5.4. *There exists some constant $\alpha \geq 1$ such that, for a population with* *initial input margin $\epsilon \geq \alpha\sqrt{n \log n}$ and adversarial leak rate $\beta \leq (\alpha\sqrt{n \log n})/12672n$, an execution of the* `DBAM` *protocol will reach a configuration with*

1. *sample error $O(\log n/n)$ when $\beta \leq O(\log n/n)$*
2. *sample error $O(\beta)$ when $\omega(\log n/n) \leq \beta \leq (\alpha\sqrt{n \log n})/12672n$*

within $O(n \log n)$ total interactions with probability at least $1 - n^{-c}$ for any $c \geq 1$ when n is sufficiently large.

To prove Theorem 5.4, we again make modified use of the random walk tools from [CHKM19]. Using the progress measures $\hat{y} = y + b/2$ and $\Delta = \hat{x} - \hat{y}$, observe that an $X \rightarrow Y$ leak event incurs twice as much negative progress to both measures as opposed to $X + B$ events. Compared to the analysis from the non-leak setting, the analysis *with* adversarial leaks must account for the stagnation (or potentially the reversal) of the protocol's progress toward reaching a high-sample-error configuration. Note that since the sample error of a configuration is defined to be $(y + b)/n$ (since we assume an initial x majority, without loss of generality), we will use the value $\hat{y}/n$ to approximate a configuration's sample error.

We use a similar structure of phases and stages as in the previous section, which can be referenced in [AAL20]. In this leak-prone setting, we refer to *productive interactions* as any of the non-null transitions found in Figure 5.1 in addition to a leak event. The set of three non-null and non-leak transitions are referred to as *non-leak productive steps*. For each phase and stage, we obtain high-probability estimates on the number of productive and total steps needed to complete the phase/stage correctly in two steps: first, we bound the number of leak events that can occur during a fixed interval of productive events, and then we show that a smaller subsequence of non-leak productive steps is sufficient to ensure that enough progress is made to offset the negative progress of the leaks. We again rely on a combination of Chernoff concentration bounds and martingale inequalities in order to show this progress at every phase and stage.

Theorem 5.4 also separates the behavior of the protocol into two classes: when $\beta \leq O(\log n/n)$ (*small* leak rate), and when $\omega(\log n/n) \leq \beta \leq O(\sqrt{n \log n}/n)$ (*large* leak rate). When the leak rate is large, the probability of a leak event conditioned on a productive step becomes roughly equal to the conditional probability of a non-leak productive step when $\hat{y} = O(\beta n)$. Thus, we cannot expect the protocol to make further "progress" toward a lower sample-error configuration with high probability beyond $\hat{y} = O(\beta n)$. The same holds for small leak rate when $\beta = O(\log n/n)$, and for even smaller values of β, our analysis tools only allow for the high-probability guarantee that $\hat{y}$ eventually drops to $O(\log n)$. In [AAL20], we give additional arguments showing that the protocol remains in a configuration with sample error $O(\log n/n)$ for small leak rate, and with sample error $O(\sqrt{n \log n}/n)$ for large leak rate, for at least a polynomial number of interactions with high probability.

5.7.2 Leak Robustness of the DBAM-C Protocol

The analysis of the previous subsection is adapted to show that the DBAM-C protocol also exhibits a similar form of leak-robustness in the CI model, and in the following theorem we prove the case where $m = \Theta(n)$. Recall that in the CI model, only the non-catalytic worker agents are susceptible to leak events.

Theorem 5.5. *There exist constants $\alpha, d \geq 1$ such that, for a population with $m = cn$ for $c \geq 1$ and input margin $\epsilon \geq \alpha\sqrt{N \log N}$, the DBAM-C protocol will reach a configuration with*

1. *sample error $O(\log N/N)$ when $\beta \leq O(\log N/N)$*

2. *sample error $O(\beta)$ when $\omega(\log N/N) \leq \beta \leq (\alpha\sqrt{N \log N})/dN$*

within $O(N \log N)$ total interactions with probability at least $1 - N^{-a}$ for $a \geq 1$ when N is sufficiently large.

The proof of the theorem uses the same progress measures and phase and stage structure introduced in the non-leak setting in Section 5.6, and the final sample-error guarantee of the protocol is again defined with respect to the magnitude of the leak rate. The behavior of the protocol between the two classes of leak rate is similar as in the DBAM analysis, and the formal details can be found in [AAL20]. Note that as the upper bound on the leak rate β is a decreasing function in N, the sample error guarantees of both protocols increase with population size. This relationship is shown across various simulations of the DBAM-C protocol in Figure 5.1. Moreover, Figure 5.2a depicts aggregate sample data over many executions of the DBAM-C protocol for varying values of N, and Figure 5.2b illustrates the logarithmic parallel time needed to reach convergence in the non-leak setting.

5.8 Leaks Versus Byzantine agents

The original third-state dynamics approximate majority protocol [AAE08] is robust to a bounded number of Byzantine agents, and as shown in the previous sections, both the DBAM protocol and the DBAM-C protocol in the CI model are robust to a bounded leak rate. In this section, we consider the connection between these two types of faulty behavior.

While leaks can occur at any agent with fixed probability throughout an execution, Byzantine agents are a fixed subset of the population, and while a leak event does not change the subsequent behavior of an agent, Byzantine agents may continue to misbehave forever. However, there are parallels between these two models of adversarial behavior. A leak at one agent can cause additional agents to deviate from a convergent configuration; similarly, interactions among non-Byzantine agents,

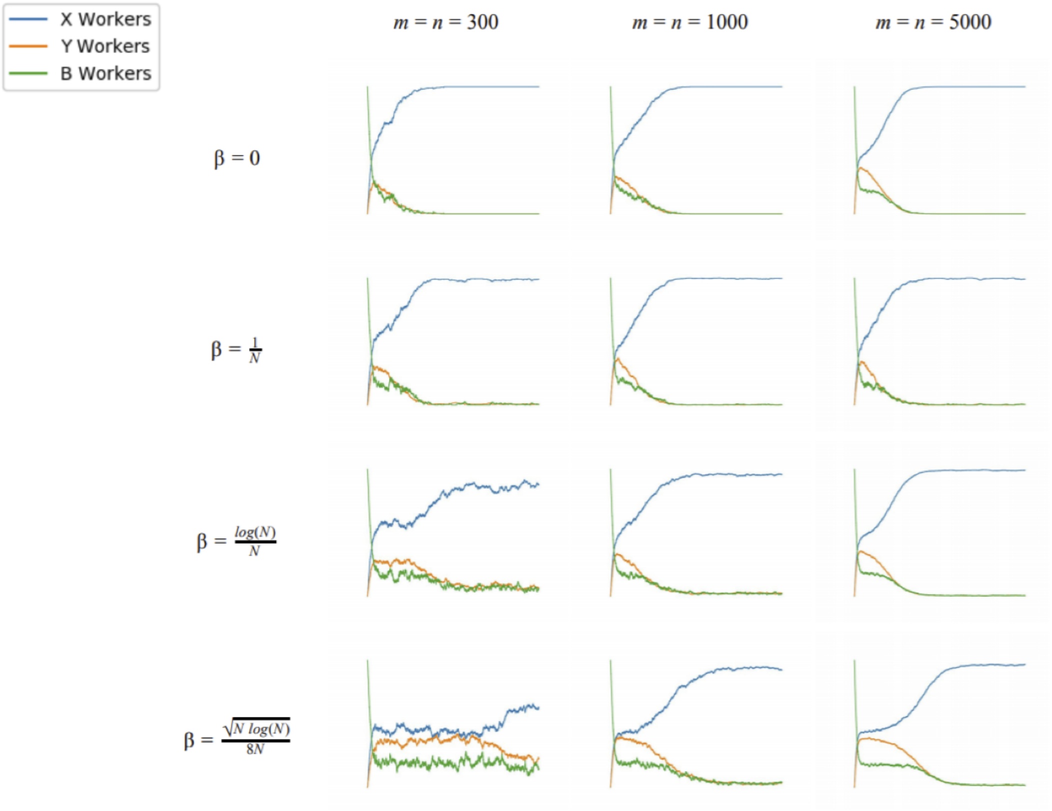
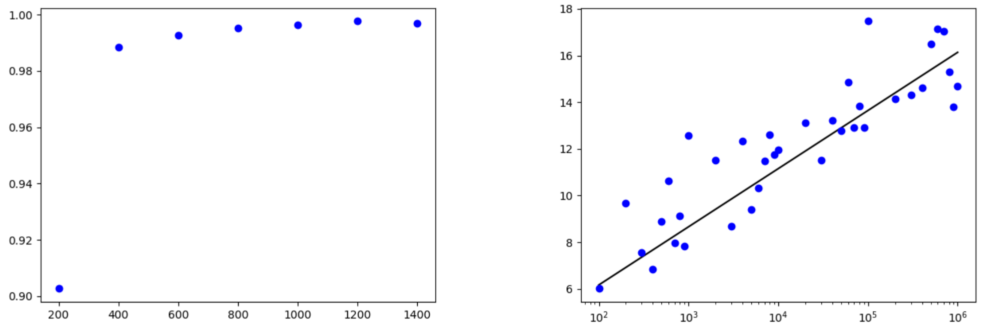


Figure 5.1: Simulations of the `DBAM-C` protocol, where each subplot shows how the proportion of X, Y and B worker agents evolves over the course of an execution. All simulations are for $m = n$, input margin $\epsilon = \sqrt{N \log N}$ (with an I_X majority), and varying values of leak rate β over $4N \log N$ total interactions. Note that these plots are of single executions and thus provide a qualitative illustration of behavior, rather than statistically significant data. However, we can see that for larger values of $m = n$ and smaller values of β, the number of X worker agents reaches a larger count more quickly (in an asymptotic sense).

some of which have deviated from a convergent configuration by interacting with a Byzantine agent, can cause additional non-Byzantine agents to diverge.

We prove that for the `DBAM` and `DBAM-C` protocols, introducing a leak rate of β has the same asymptotic effect as introducing $O(\beta N)$ Byzantine agents to the population, which demonstrates an equivalence between these two notions of adversarial behavior among the class of third-state dynamics protocols. Although the results of the previous section assumed leaks that do not follow the laws of chemistry, the following result considers **weak leaks**, which cause the selected agent to decrease its confidence in the majority value by one degree (i.e. a leak causes an agent in state X



(a) Sample success rate (y-axis) averaged over 3000 executions of `DBAM-C` for $n = 600$, $\Delta_0 = \sqrt{N \log N}$, and $\beta = 1/N$, and varying values of m (x-axis). Samples were drawn uniformly from the worker population after $4N \log N$ total interactions.

(b) Parallel time (y-axis) for `DBAM-C` without leaks to reach consensus for varying population sizes (x-axis) where $m = 2n$. Data points represent a single execution. The solid black line is $\frac{3}{4}\log_2(N)$, showing that convergence takes $O(N \log N)$ interactions.

Figure 5.2: Success rate and running time of `DBAM-C` over various executions of the protocol.

to transition to B and an agent in state B to transition to Y, matching the $X + Y$ and $Y + B$ transitions). For our purposes, we define two adversarial models $\mathcal{M}_1$ and $\mathcal{M}_2$ to be **equivalent** for some protocol $\mathcal{P}$ if $\mathcal{P}$ converges to the same asymptotic sample error rate in the same asymptotic running time in both models. We then have the following equivalence result:

Theorem 5.6. *A population of N agents running `DBAM` (or `DBAM-C`) with weak leak rate $O(\beta)$ is equivalent to a population of $N + \mathcal{B}$ agents, where $\mathcal{B} = O(N\beta)$ agents are Byzantine, running `DBAM` (or `DBAM-C`) without leaks, where in either setting the protocol converges in $O(N \log N)$ interactions with error probability $O(\beta)$.*

Proof. Below we will refer to the running protocol as $\mathcal{P}$, where it is implicit that $\mathcal{P}$ is either `DBAM` or `DBAM-C`. Though different from the notation used in previous sections, we will refer to the total number of agents in the population (catalytic or otherwise) as N. We will prove the equivalence of these adversarial behaviors in two parts.

First we must show that running $\mathcal{P}$ among N agents with weak leak rate β has the same asymptotic error rate and run-time as running $\mathcal{P}$ in a population of $N + \mathcal{B}$ agents *without leaks*, where $\mathcal{B} = O(N\beta)$ is the number of Byzantine agents. We can do this by imagining that the N honest agents are the entire population executing $\mathcal{P}$. At each step in time, the scheduler selects two agents from the population to interact. When any two of the N honest agents interact, the interaction is indistinguishable from the case where there are N total agents in the non-Byzantine setting. However, with some probability p, one Byzantine and one non-Byzantine agent are selected to interact with one another, potentially causing the non-Byzantine agent to diverge from a convergent state. In the `DBAM` and `DBAM-C` protocols, this would mean that the Byzantine agent could just stay in state Y, causing catalysts and Y-agents to stay the same (as needed), B-agents to become Y-agents, and X-agents to become B-agents. The probability of this cross-interaction is

$$\begin{aligned} p = \frac{N\mathcal{B}}{\binom{N+\mathcal{B}}{2}} \leq \frac{N(cN\beta)}{\binom{N+cN\beta}{2}} &\leq \frac{2N(cN\beta)}{N^2} \\ &= 2c\beta \\ &= O(\beta) \end{aligned}$$

We define an **effective interaction** to be any interaction that produces a non-null state transition. Here, an effective interaction is any interaction between honest agents and any simulated leaks, i.e. interactions between honest and Byzantine agents. The running time of the simulation is the number of total steps it takes to have sufficiently many effective interactions to successfully complete the protocol among the N honest agents. The probability of an effective interaction is

$$\begin{aligned} \frac{N(N-1)/2 + N\mathcal{B}}{\binom{N+\mathcal{B}}{2}} &= \frac{N(N-1) + 2N\mathcal{B}}{(N+\mathcal{B})(N+\mathcal{B}-1)} \\ &= \frac{N(N+2\mathcal{B}-1)}{(N+\mathcal{B})(N+\mathcal{B}-1)} \end{aligned}$$

So if the running time of $\mathcal{P}$ on N honest agents with leaks is $O(N \log N)$ total interactions, then the simulation must run for $O\left((N+\mathcal{B}) \log N\right)$ steps so that the expected fraction of “productive” interactions is

$$O\left(\frac{N(N+2\mathcal{B}-1)}{(N+\mathcal{B})(N+\mathcal{B}-1)}(N+\mathcal{B})\log N\right) \leq O\left(\frac{N(2N+2\mathcal{B}-2)}{N+\mathcal{B}-1}\log N\right)$$
$$= O(N\log N)$$

For $\mathcal{B} = O(N\beta)$ and $\beta \leq 1$, $(N+\mathcal{B})\log N = N\log N + O(N\beta\log N) = O(N\log N)$.

Next we must show that running $\mathcal{P}$ with $N+\mathcal{B}$ agents, where $\mathcal{B} = O(N\beta)$ agents are Byzantine, has the same asymptotic error rate and running time as running $\mathcal{P}$ in a population of N agents with leak rate β. In the Byzantine setting, we only care about the behavior of the honest nodes. Therefore, as before, we can imagine that the N agents in the population are equivalent to the N honest agents in the population we are trying to simulate and now suppose that there are an additional $O(N\beta)$ Byzantine agents. Assuming that the Byzantine agents are optimally adversarial (always in the Y state), then weak leaks among the honest agents directly simulate interactions with Byzantine agents. Also as before, the total number of steps of the simulation will be $O(N\log N) = O\left((N+\mathcal{B})\log(N+\mathcal{B})\right)$ because we are omitting interactions among Byzantine agents entirely, yielding fewer total interactions to simulate the same behavior in the population of $N+\mathcal{B}$. □

5.8.1 Super-Adversarial Byzantine Agents

In Theorem 5.6, we assumed that leaks are not fully adversarial (converting X to Y), but rather only decrease the confidence in the majority value by one degree. However, our analysis in previous sections assumes fully adversarial leaks in order to demonstrate that in the worst possible case (i.e. for the maximum decrease in the progress measure), we still succeed in computing `ApproxMajority` up to leak rate β. The equivalent to this in the Byzantine model would be to add a transition to `DBAM` or `DBAM-C` of the form $T + \_ \rightarrow T + Y$, where T is a special state held only by **super-adversarial Byzantine agents**, and _ is a wildcard representing any non-catalytic state. This new state transition indicates that interacting with a Byzantine agent causes any non-catalytic agent to shift into the Y state, exactly modeling the fully adversarial leaks described in earlier sections.

We observe that the proof of Theorem 5.6 can be repurposed to demonstrate an equivalence between the stronger notion of fully adversarial leaks used in earlier sections and the modified protocols defined above for super-adversarial Byzantine agents. This allows us to then apply the results from Theorem 5.6 to the actual `DBAM` and `DBAM-C` protocols as they are described in previous sections, where the adversarial process is strictly weaker.

Chapter 6

Conclusion & Future Work

This dissertation examines various extensions to the original population protocol model that was introduced in [AAD$^+$06]. The modifications to the original model improve the adaptability of these abstract algorithms to the real-world systems they aim to model, whether they be wireless sensor networks, IoT devices, wearable medical trackers, or molecules in a well-mixed solution.

In this work, we have shown that the separation of an agent's state into distinct components for communication and local computation permits arbitrary Turing machine computation even when an agent can obtain only 1-bit of information from each interaction. We have examined the issue of privacy in population protocols and provided new definitions and algorithms for achieving privacy with probabilistic scheduling. We also demonstrated that immutable states impact computation in the population protocol model, and that third-state dynamics are resilient to both these immutable states and spontaneous state changes.

The contribution of this work to the study of population protocols is also significant in that it offers a foundation for further research in these key variations to the original model. While this work provides preliminary results investigating the topics of message complexity, input privacy, and catalytic inputs in the population protocol model, there remains much left to explore in each of these areas. Below, we discuss possible avenues for future work.

6.1 Message Complexity

While our work solves several problems in the population protocol message model (like `BitBroadcast` and stable computation of arbitrary symmetric functions), there is a multitude of other problems that would be interesting to solve in this model with improved running time efficiency. There remains much left to do on the task of further stratifying computable problems in this model into classes of time complexity based on tradeoffs between internal and external state complexity.

Furthermore, it is compelling to explore the impact of waning internal memory on the computational power of this model. In many devices, battery life is a concern and is often times dealt with by limiting system resources when energy is nearly exhausted. If we need to conserve internal memory over time, it would be interesting to determine what information is safest to delete, what minimum amount of *internal storage* (i.e. computational capacity or RAM) must we hang onto, and whether we achieve some conservation of internal state space when computing the same well-studied problems in this model?

6.2 Input Privacy

As mentioned earlier, the present work does not solve the problems of computing `Threshold` and `Or` privately in population protocols with probabilistic scheduling. Designing algorithms for these predicates would allow for private input computation of arbitrary semilinear predicates in the population protocol model. While additional predicates are stably computable in the population protocol model with probabilistic scheduling, we chose to focus on the semilinear predicates so as to directly compare our results to those in [DGFGR07]. However, it is certainly of interest to study private computation of arbitrary functions in the model. Moreover, we expect that more efficient algorithms for privately computing `Remainder` exist — future works may determine a running time lower bound for computing `Remainder` with $O(1)$ state and matching algorithm for solving this problem.

Additional work is also needed to determine the relationship between weak privacy and computational indistinguishability of same-output input vectors. Alternative definitions of privacy may be better suited for different threat models and computational assumptions, and we are interested in further exploring how such definitions relate to one another.

6.3 Catalytic Inputs

Much of the results in this work presume that the sizes of the catalytic and worker populations are asymptotically equal; however, further work in this area should consider varying relative sizes of these two sectors of the population. In addition, while we showed that third-state dynamics can compute `ApproxMajority` with high probability, it remains open whether this can be done stably in the catalytic model. We suspect that the leak rate tolerance of third-state dynamics for solving `ApproxMajority` varies based on the initial margin of the population. We offer this as a suggested problem for future work.

Our work demonstrates a relationship between Byzantine processes and spontaneous leak events in third-state dynamics solving `ApproxMajority`. It is an interesting question to ask whether or not such a relationship can be cast between these two adversarial processes among general computations by population protocols.

We conclude with the remark that many alternative variations to the population protocol model may make this theoretical framework more adaptable to real-world systems.

Bibliography

[AAC16] Dana Angluin, James Aspnes, and Dongqu Chen. A population protocol for binary signaling consensus. Technical Report YALEU/DCS/TR-1527, Yale University Department of Computer Science, 2016.

[AAD^{+}06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 18:235–253, 03 2006.

[AAD^{+}20] Talley Amir, James Aspnes, David Doty, Mahsa Eftekhari, and Eric Severson. Message Complexity of Population Protocols. In *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[AAE06a] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21:183–199, 01 2006.

[AAE06b] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, 2006.

[AAE08] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, July 2008.

[AAE^{+}17] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald Rivest. Time-space trade-offs in population protocols. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2560–2579. SIAM, 2017.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[AAFJ05] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing behavior in networks of nondeterministically interacting sensors. In *9th International Conference on Principles of Distributed Systems*, 2005.

[AAFJ08] Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. *ACM Trans. Auton. Adapt. Syst.*, 3(4):13:1–13:28, December 2008.

[AAG18a] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *SODA*, 2018.

[AAG18b] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2221–2239. SIAM, 2018.

[AAL20] Talley Amir, James Aspnes, and John Lazarsfeld. Approximate majority with catalytic inputs. In *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[ABBS16] James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. Time and Space Optimal Counting in Population Protocols. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[ADG$^+$05] James Aspnes, Zoë Diamadi, Kristian Gjøsteen, René Peralta, and Aleksandr Yampolskiy. Spreading alerts quietly and the subgroup escape problem. In *Advances in Cryptology - ASIACRYPT 2005*, pages 253–272, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[ADK$^+$17] Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *DNA*, 2017.

[AG15] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proceedings, Part II, of the 42Nd International Colloquium on Automata, Languages, and Programming - Volume 9135*, ICALP 2015, pages 479–491, Berlin, Heidelberg, 2015. Springer-Verlag.

[AGV15] Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 47–56, 2015.

[Ash90] R.B. Ash. *Information Theory*. Dover books on advanced mathematics. Dover Publications, 1990.

[ATU21] Dan Alistarh, Martin Töpfer, and Przemysław Uznański. Comparison dynamics in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 55–65, New York, NY, USA, 2021. Association for Computing Machinery.

[BC18] Olivier Blazy and Céline Chevalier. Spreading alerts quietly: New insights from theory and practice. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. Association for Computing Machinery.

[BCC$^+$21] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 33–44, New York, NY, USA, 2021. Association for Computing Machinery.

[BCER17] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 451–453, 2017.

[BCN20] Luca Becchetti, Andrea Clementi, and Emanuele Natale. Consensus dynamics: An overview. *ACM SIGACT News*, 51(1):58–104, 2020.

[BEF$^+$18] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and $\Theta(\log n)$ states. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[BEF$^+$20] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Time-space trade-offs in population protocols for the majority problem. *Distributed Computing*, 2020.

[BGK20] Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 119–129, New York, NY, USA, 2020. Association for Computing Machinery.

[BKKO18] Petra Berenbrink, Dominik Kaaser, Peter Kling, and Lena Otterbach. Simple and Efficient Leader Election. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 9:1–9:11, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BKR19] Petra Berenbrink, Dominik Kaaser, and Tomasz Radzik. On counting the population size. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 43–52. ACM, 2019.

[CCN12] Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2:656, 09 2012.

[CDS12] Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. In *Natural Computing*, volume 13, 04 2012.

[CDS$^+$13] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from dna. *Nature nanotechnology*, 8, 09 2013.

[CFG$^+$20] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing, 2020.

[CHKM19] Anne Condon, Monir Hajiaghayi, David Kirkpatrick, and Ján Maňuch. Approximate majority analyses using tri-molecular chemical reaction networks. *Natural Computing*, pages 1–22, 2019.

[CKL$^+$20] Ran Canetti, Yael Tauman Kalai, Anna Lysyanskaya, Ronald Rivest, Adi Shamir, Emily Shen, Ari Trachtenberg, Mayank Varia, and Daniel J. Weitzner. Privacy-preserving automated exposure notification. *IACR Cryptol. ePrint Arch.*, 2020:863, 2020.

[CMN$^+$11] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. In *Proceedings of the 7th ACM ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing*, FOMC '11, page 6–15, New York, NY, USA, 2011. Association for Computing Machinery.

[CMT05] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 109–117, 2005.

[dCN20] Francesco d'Amore, Andrea Clementi, and Emanuele Natale. Phase transition of a non-linear opinion dynamics with noisy interactions - (extended abstract). In *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2020.

[DE19] David Doty and Mahsa Eftekhari. Efficient size estimation and impossibility of termination in uniform dense population protocols. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 34–42. ACM, 2019.

[DGFGR06] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *Distributed Computing in Sensor Systems*, pages 51–66, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DGFGR07] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. Secretive birds: Privacy in population protocols. In *Principles of Distributed Systems*, pages 329–342, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[DGGK09] Shlomi Dolev, Juan Garay, Niv Gilboa, and Vladimir Kolesnikov. Swarming secrets. In *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1438–1445, 2009.

[DK18] Bartlomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 87–99. ACM, 2018.

[DS15] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *Distributed Computing*, pages 602–616, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[DY13] Laye Hadji Diakite and Li Yu. Energy and bandwidth efficient wireless sensor communications for improving the energy efficiency of the air interface for wireless sensor networks. In *2013 IEEE Third International Conference on Information Science and Technology (ICIST)*, pages 1426–1429, March 2013.

[Fel68] William Feller. *An introduction to probability theory and its applications. Vol. I.* Third edition. John Wiley & Sons Inc., New York, 1968.

[GDE$^+$21] Leszek Gasieniec, David Doty, Mahsa Eftekhari, Eric Severson, Grzegorz Stachowiak, and Przemysław Uznański. A time and space optimal stable population protocol solving exact majority. *IEEE*, 2021.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[GS01] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

[GS18] Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2653–2667. SIAM, 2018.

[GSU19] Leszek Gąsieniec, Grzegorz Stachowiak, and Przemysław Uznański. Almost logarithmic-time space optimal leader election in population protocols. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, page 93–102, New York, NY, USA, 2019. Association for Computing Machinery.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[KU18] Adrian Kosowski and Przemysław Uznański. Population protocols are fast. *arXiv preprint arXiv:1802.06872*, 2018.

[Lin17] Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.

[LLZ$^+$12] Rongxing Lu, Xiaodong Lin, Haojin Zhu, Xiaohui Liang, and Xuemin Shen. Becan: A bandwidth-efficient cooperative authentication scheme for filtering injected false data in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(1):32–43, Jan 2012.

[LLZC13] Chen-Xu Liu, Yun Liu, Zhen-Jiang Zhang, and Zi-Yao Cheng. High energy-efficient and privacy-preserving secure data aggregation for wireless sensor networks. *International Journal of Communication Systems*, 26(3):380–394, 2013.

[MAA$^+$15] Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *14th IEEE International Symposium on Network Computing and Applications*, pages 35–42, 2015.

[MAS16] Yves Mocquard, Emmanuelle Anceaume, and Bruno Sericola. Optimal proportion computation with population protocols. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 216–223, Oct 2016.

[MT19] Nima Monshizadeh and Paulo Tabuada. Plausible deniability as a notion of privacy. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 1710–1715, 2019.

[PVV09] Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *IEEE INFOCOM 2009*, pages 2527–2535. IEEE, 2009.

[QSW10] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient Turing-universal computation with DNA polymers. In *DNA 2010: Proceedings of The Sixteenth International Meeting on DNA Computing and Molecular Programming*, volume 6518 of *Lecture Notes in Computer Science*. Springer, 2010.

[SCWB08] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7:615–633, 2008.

[SOI$^+$19] Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Logarithmic expected-time leader election in population protocol model. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 323–337. Springer, 2019.

[SPS$^+$17] Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358(6369):eaal2052, 2017.

[SSW08] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. In *DNA14*, pages 57–69, 2008.

[STE19] Prahesa K. Setia, Gamze Tillem, and Zekeriya Erkin. Private data aggregation in decentralized networks. In *2019 7th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*, pages 76–80, 2019.

[TG09] Gelareh Taban and Virgil D. Gligor. Privacy-preserving integrity-assured data aggregation in sensor networks. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 03*, CSE '09, page 168–175, USA, 2009. IEEE Computer Society.

[TWS15] Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In *DNA Computing and Molecular Programming - 21st International Conference, DNA 21, Boston and Cam-*

bridge, MA, USA, August 17-21, 2015. Proceedings, volume 9211 of *Lecture Notes in Computer Science*, pages 133–153. Springer, 2015.

[UVV13] P. Udayakumar, Ranjana Vyas, and O. P. Vyas. Energy efficient election protocol for wireless sensor networks. In *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*, pages 1028–1033, March 2013.

[WDM+19] Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567:366–372, 2019.

[WTE+18] Boya Wang, Chris Thachuk, Andrew D. Ellington, Erik Winfree, and David Soloveichik. Effective design principles for leakless strand displacement systems. *Proceedings of the National Academy of Sciences*, 115(52):E12182–E12191, 2018.

[Yao77] Andrew C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[YOA+13] Ercan Yildiz, Asuman Ozdaglar, Daron Acemoglu, Amin Saberi, and Anna Scaglione. Binary opinion dynamics with stubborn agents. *ACM Transactions on Economics and Computation (TEAC)*, 1(4):1–30, 2013.

[YTM+00] Bernard Yurke, Andrew J. Turberfield, Allen P. Mills, Jr., Friedrich C. Simmel, and Jennifer L. Nuemann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.