Secretive Birds: Privacy in Population Protocols

Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Eric Ruppert

Secretive Birds: Privacy in Population Protocols

Carole Delporte-Gallet[*] Hugues Fauconnier[*] Rachid Guerraoui[†] Eric Ruppert[‡]

September 28, 2007

Abstract

We study private computations in a system of tiny mobile agents. We consider the mobile population protocol model of Angluin *et al.* [2] and ask what can be computed without ever revealing any input to a curious adversary. We show that any predicate that can be computed in the original population model can be made private through an obfuscation procedure that exploits the inherent non-determinism of the mobility pattern. In short, the idea is for every mobile agent to generate, besides its actual input value, a set of wrong input values to confuse the curious adversary. To converge to the correct result, the procedure has the agents eventually eliminate the wrong values; however, the moment when this happens is hidden from the adversary. This is achieved without jeopardizing the tiny nature of the agents: they still have very small storage size that is independent of the cardinality of the system. We present three variants of this obfuscation procedure that help compute respectively, *remainder*, *threshold*, and *or* predicates which, when composed, cover all those that can be computed in the population protocol model.

Keywords: population protocols, mobile computing, privacy, anonymity, secure multi-party computation, automata.

A little bird has whispered a secret to me. [9]

1 Introduction

Despite the large amount of recent work on mobile systems, very little theoretical research has been devoted to modelling such systems. A notable exception is the work of Angluin *et al.* [2]: they introduced the population protocol model to describe systems of very simple mobile agents. The model has totally asynchronous agents, only a constant amount of memory per agent, no system infrastructure, and no assumptions about the mobility patterns of the agents, except for a fairness guarantee that ensures (for example) that agents cannot be forever disconnected from the others. The model was illustrated with a set of sensors, each strapped to a bird. Pairs of sensors could communicate when their host birds were close together, and the sensor network would provide aggregated information about the flock.

The population protocol model, along with some variations, has been studied in a series of papers [1, 2, 3, 4, 5, 6, 7]. In particular, the class of decision problems that can be solved by the population protocol model has been characterized precisely. Angluin *et al.* [2] gave several examples of predicates that can be computed in the population protocol model, and it was later shown that

[*]LIAFA, Université Paris VII
[†]School of Computer and Communication Sciences, EPFL
[‡]Department of Computer Science and Engineering, York University

no others are computable [4]. This provided a characterization of computable predicates in the model: those that can be expressed in Presburger arithmetic [10]. (This is essentially first-order arithmetic, using the symbols $+, 0, 1, \wedge, \vee, \neg, \forall, \exists, =, <, (,)$ and variables.)

Computability within the population protocol model is defined in terms of eventually stabilizing to the correct output value. This is an essential property of the model, since there are no assumptions about the mobility pattern of the agents, beyond the rather weak fairness guarantee. In particular, an individual agent may have no interactions at all for an arbitrarily long prefix of a computation, so in general, one can never be certain that the final output value has been computed.

A key aspect of the population model is anonymity: there is no way to distinguish any two agents. One motivation for such an assumption is the lack of infrastructure and the mass production that might render it difficult to assign unique identifiers to agents or to programme them individually. Another motivation is for the agents to preserve their privacy. An agent might simply not want to reveal who it is, when it met which other agent or where it was. The first motivation underlying anonymity is sometimes questionable. Indeed, it takes only a small number of bits to store a huge collection of agent identifiers and a simple randomized procedure can generate distinct identifiers with very high probability. The second motivation seems generally more relevant, for there are many reasons a mobile agent might not like to leave its identifier wherever it goes or share it with whomever it meets.

In this work, we explore the privacy aspect of these anonymous mobile systems. That is, not only do we consider algorithms where agents never reveal their identifiers but we also seek for them to hide their input values from one another while computing some function of those inputs. In general, we say an algorithm is private if an honest but curious agent cannot learn any information about the inputs to the system (including even the number of inputs) beyond what can be deduced from its own input and the output value that must be computed. (This requirement would enforce anonymity, even if the agents had identifiers: otherwise one could deduce a lower bound on the number of participating agents.) Here, we focus on ensuring privacy in any finite prefix of a computation. This, together with the fact that population protocols are only required to eventually stabilize to the correct output value, allows us to strengthen the notion of privacy: we require that an honest but curious agent cannot definitively learn *anything* about the inputs of agents at any point in the computation, yet the algorithm must still correctly stabilize to the correct output value.

Consider a simple example of determining which of two candidates is the winner of an election by the agents. Assume each agent has input value 1 if it votes for the first candidate and 2 if it votes for the second candidate. There is a simple protocol to achieve this computation [2]: when two agents with different votes meet, they cancel each other. Once an agent has had its vote cancelled, it remembers the last non-cancelled vote that it has seen to determine its output. (Some extra care must be taken to deal with the possibility of a tie vote.) This protocol, however, provides no privacy. In fact, the unpredictability of the mobility pattern might allow a single curious agent to meet *all* others in their initial state and deduce the exact input vector of the entire population, discovering exactly how many agents voted for each candidate. In this paper, we ask what predicates can be computed without letting any curious agent, at any point of its computation, determine any information about the input (or output) values of any other agent. Following the specific example above, this means we would like a curious agent to be unable to determine at any point of its computation how many agents voted for a candidate, the width of the margin by which one candidate won, whether the number of voters was even or odd, and so on.

In a sense, we study a variant of secure multi-party computations [8] in the context of population protocols. We consider a passive adversary that can read the state of one agent but cannot corrupt it. However, there are several ways in which our work differs from the usual notion of secure multi-party computation. The tiny nature of the devices we consider precludes the use of expensive cryptographic protocols. The anonymity of the system means that signature schemes cannot be used. Our algorithms do not use randomization, instead relying on the inherent non-determinism of the mobility pattern. Interestingly, in our model, the curious agent can see the entire state of any other agent it interacts with, so no secret keys can be used to achieve privacy.

This paper proves that any predicate that can be computed in the original population protocol model, namely any predicate that can be expressed in Presburger arithmetic, can be computed in a private way. Our result holds even if the curious agent can store an unbounded amount of information, namely the states of all agents it has interacted with from the beginning of the computation to the present. At the heart of our result lies the idea of an *obfuscation procedure* which heavily relies on the non-determinism of the mobility pattern. We use this procedure in different forms, according to whether we compute a *remainder*, a *threshold* or an *or* predicate. (The composition of these covers all predicates that can be computed in the original population protocol model.) Basically, we make agents change their input values without changing the overall output, in a way that is carefully designed to confuse any curious agent. In the context of the voting example, this would, roughly speaking, mean that every agent would generate, besides its own vote several votes that cancel each other. The procedure is devised such that (a) the confusing values are eventually cancelled, without any curious agent knowing when that happens, and (b) the correct result is indeed computed, while making sure that the size of the memory of every agent is fixed, independent of the size of the system.

The rest of the paper is organized as follows. We first recall the original population protocol model and introduce our definition of private computation in this context. Then we show how to compute any *remainder* or *threshold* predicate, and then how to compute any Boolean combination of such predicates, deriving our general result about what can be computed in a private way. We conclude the paper by discussing several research directions for private mobile computing.

2 Private Population Protocols

Our formalization of the population protocol model is based on the work of Angluin *et al.* [2]. For a population of n agents, $\mathcal{P}_n = \{p_0, \ldots, p_{n-1}\}$ denotes the set of agents. (The subscripts are for convenience only, and are not visible to the agents themselves: they do not have any effect on an execution.) Each agent in the system is modelled as a finite state machine, and algorithms must be *uniform*: each finite state machine is "programmed" identically and the programming does not depend on the number of agents in the system. This makes the model strongly *anonymous*, since there is not enough space in the state to give each agent a unique identifier.

Let Σ be a finite input alphabet and Y be a finite output alphabet. Each agent p_i has an input drawn from Σ. The input for a population protocol for n agents is a vector $I = (\sigma_0, \ldots, \sigma_{n-1})$ of elements of Σ, where σ_i is the input of agent p_i. Let $\mathcal{D}$ be the set of all vectors on Σ of length at least two. The goal of an algorithm is to compute a function $f : \mathcal{D} \to Y$. Each agent must eventually output the value of this function for the input that was initially provided to the agents. Here we restrict ourselves to compute only *predicates*: the output alphabet is the set $\{0, 1\}$.

We now describe how to specify a population protocol. A population protocol is defined by a finite set Q of possible agent states, an input assignment $\iota : \Sigma \to Q$, a transition relation $\delta \subseteq Q \times Q \times Q \times Q$, and an output assignment $\omega : Q \to Y$. If two agents in states q_1 and q_2 encounter each other, they can change into states q_1' and q_2', respectively, where $(q_1, q_2, q_1', q_2') \in \delta$. We sometimes use the notation $q_1, q_2 \to q_1', q_2'$ to describe the elements of δ.

A *configuration* is a mapping $C : \mathcal{P}_n \to Q$ specifying the state of each agent. Let C and C' be configurations, u, v be distinct agents and t be a transition. We say that C goes to C' with *interaction* $e = ((u, v), t)$, denoted $C \xrightarrow{e} C'$, if $t = (C(u), C(v), C'(u), C'(v))$ belongs to δ and $C'(w) = C(w)$ for all $w \in \mathcal{P}_n - \{u, v\}$. We say that C goes to C' in one step, denoted $C \to C'$, if

$C \xrightarrow{e} C'$ for some interaction $e = ((u, v), t)$; in this case e is called the interaction associated with this step, t is the transition of this step and agent u and agent v are *involved* in this step.

An *execution* of the protocol on input $I \in \mathcal{D}$ is an infinite sequence of configurations, $C_0, C_1, C_2, \ldots$ such that (1) C_0 is the initial assignment for I: if $I = (\sigma_0, \ldots, \sigma_{n-1})$ then, for all i such that $0 \leq i \leq n-1$, $C_0(p_i) = \iota(\sigma_i)$ and (2) $C_i \to C_{i+1}$ for all i. An *execution fragment* is a contiguous portion of an execution. The output of an agent in state q is $\omega(q)$. We say that the execution *stably outputs* $v \in Y$ if every agent eventually outputs v and never changes its output thereafter. Formally, this means that there is an i such that for all agents p and for all $j > i$, $\omega(C_j(p)) = v$.

If every sequence of interactions were considered to be a possible execution in the model, it would be possible to have isolated agents that never interact with one another. So the model must incorporate a fairness guarantee. In a *fair* execution, if a configuration C occurs infinitely often and $C \to C'$, then C' occurs infinitely often. If, for example, we associate probabilities with different interactions, then an execution will be fair with probability 1. A protocol *stably computes* a function $f : \mathcal{D} \to Y$ if, for every input $I \in \mathcal{D}$, every fair execution on input I stably outputs $f(I)$. In the following, all executions are assumed to be fair.

Given an execution $E = C_0, C_1, C_2, \ldots$ and an agent u, the *history of interactions* for agent u in E, denoted $H_u(E)$ is the sequence of states and transitions of interactions associated with each step of E in which u is involved. More precisely $H_u(E) = (q_0, t_0), \ldots (q_i, t_i) \ldots$ where t_i is the transition of the i-th interaction in which p_i is involved and q_i is the state of agent u when this interaction occurs. The history of interactions for agent u in E up to T is the initial segment of length T of $H_u(E)$ if T is greater than the length of $H_u(E)$ and $H_u(E)$ otherwise.

We now define the notion of privacy for a population protocol. If some agent encounters another agent, we assume that it learns both the current state of the other one and what transition is chosen. Then, intuitively, the protocol has the privacy property if no agent can learn anything about the current input from any initial sequence of the history of interactions in which it is involved. Let I_1 and I_2 be two inputs in $\mathcal{D}$ where some agent p gets the same input value. The agent p is able to distinguish I_1 from I_2 if, for at least one execution E_1 on input I_1, the history of interactions for p in E_1 up to some T cannot be an initial segment of a history of interactions of agent p for any execution on input I_2. A population protocol has the *output-independent privacy* property if no agent is able to distinguish any pair of sufficiently large input vectors in which it has the same input value. More formally, 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. Thus, if the protocol is private, at no time in the execution E_1 on input I_1 can an agent p deduce with certainty that the input vector of the execution was not I_2. In other words, there is no time when p can rule out any possible input vector (of size at least n_0).

3 Computing Predicates Privately

Our goal is to show that all predicates computable in the population protocol model are computable privately. We shall show that all computable predicates can be computed by a protocol satisfying several properties, and that those properties are sufficient to guarantee that the protocol has output-independent privacy. We label the curious agent p_0. (Since the identities of agents cannot be used in the protocols themselves, the arguments below are not affected by this convention.)

Consider a population protocol with state set Q. Fix some collection $\mathcal{G}$ of system configurations, which we shall call *good* configurations. A transition $q, r \to s, t$ of the protocol is called *$\mathcal{G}$-imitable* if, from any configuration $C_0 \in \mathcal{G}$ with p_0 in state q or r, there exists an execution fragment $C_0 \to C_1 \to \cdots \to C_m$ such that $C_m \in \mathcal{G}$ and agent p_0 participates in exactly one interaction during the fragment and that interaction's transition is $q, r \rightarrow s, t$. (This property should hold both for the case where p_0 is playing the role of the agent that changes from state q to s and for the case where p_0 changes from r to t.)

The following theorem, which will be proved in Sections 3.1, 3.2 and 3.3, will yield protocols that have output-independent privacy.

Theorem 1 *Let P be any predicate that is computable in the population protocol model (without privacy). Then there exist a protocol A that computes P, a constant n_0 and a set $\mathcal{G}$ of configurations of A such that*

1. *for any input configuration of size at least n_0, there is an execution fragment of A that contains no interactions involving p_0 and ends in a configuration of $\mathcal{G}$,*
2. *every transition of A is $\mathcal{G}$-imitable,*
3. *for any states q_1 and q_2, there is a sequence of interactions between two agents that start in states q_1 and q_2 and end in states q_2 and q_1, respectively, and*
4. *for any states q_1 and q_2, the "null" transition $q_1, q_2 \rightarrow q_1, q_2$ is permitted.*

We now show that the first two properties of the preceding theorem are sufficient for privacy.

Theorem 2 *Any population protocol that satisfies Properties 1 and 2 of Theorem 1 has output-independent privacy.*

Proof: Consider any execution prefix E, starting from an initial configuration C_0. Let C'_0 be any initial configuration that has at least n_0 agents. We must construct an execution prefix E', starting from C'_0, such that p_0 undergoes the same sequence of interactions in E and E'. We begin E' with the execution fragment that satisfies Property 1, which does not include any interactions involving p_0 and leaves the system in a good configuration.

Then, for each interaction involving p_0 in E, we append an execution fragment to the end of the constructed execution using the definition of $\mathcal{G}$-imitability. Each fragment includes exactly one interaction that involves p_0, and that interaction's transition is the same as in p_0's next interaction in E, and the fragment leaves the system in a good configuration. The existence of such a fragment is guaranteed by Property 2.

When all of these fragments have been appended, we obtain the required execution E'. The history of interactions for p_0 is the same in E and E', by construction. ■

The following corollary follows immediately from Theorems 1 and 2.

Corollary 3 *Every predicate that can be computed in the population protocol model (without privacy) can be computed with output-independent privacy.*

Although Properties 3 and 4 of Theorem 1 are not required for privacy, they are crucial for our proof, in Section 3.3, that Boolean combinations of privately computable predicates are also privately computable.

3.1 Computing Remainder Predicates

Let Σ be an input alphabet. Let c_σ be an integer constant for each $\sigma \in \Sigma$ and let m and r be integer constants such that $0 \le r < m$. The predicate $P(I)$ that is 1 on input $I = (\sigma_0, \ldots, \sigma_{n-1})$ if and only if $\sum_{i=0}^{n-1} c_{\sigma_i} \equiv r (\text{mod } m)$ is called a *remainder predicate*. In this section, we show that any remainder predicate can be computed in a way that satisfies the properties of Theorem 1.

There is a fairly straightforward way to compute the predicate $P(I)$ if there is no need for privacy [2]. Each agent stores a value, initially c_σ, where σ is the input symbol of the agent. When two agents with values v_1 and v_2 meet, one agent gives its value to the other: they change their values to 0 and $v_1 + v_2$. All arithmetic is done modulo m. The algorithm maintains the sum of the values of all agents as an invariant. Eventually, the sum is stored in a single agent, which can then determine the output value and disseminate it to all other agents.

To ensure privacy, we must add transitions that allow agents to disguise their input values. When agents in states v_1 and v_2 meet, one can give the other *part* of its value: the agents change their values to $v_1 + 1$ and $v_2 - 1$. This preserves the sum of the agents' values as an invariant. However, this modification, by itself, would prevent the protocol from converging to the correct output. To avoid this problem, we introduce a mechanism that ensures that this transition is only applied a finite (but unbounded) number of times. This will be sufficient to obscure the inputs from the adversary, while still ensuring that the sum is eventually gathered into a single agent to produce the output value. This mechanism is implemented by giving each agent a flag that is initially 1 and is eventually changed to 0. The transitions in which one agent shifts part of its value to the other are enabled only while the flags are 1. The algorithm is described more precisely in the following proof.

Proposition 4 *Any remainder predicate can be computed by a protocol satisfying the properties of Theorem 1.*

Proof: We describe the protocol that computes the predicate $\sum_{i=0}^{n-1} c_{\sigma_i} \equiv r (\text{mod } m)$. The state of each agent is a pair (v, f) comprised of a value $v \in \{\perp_0, \perp_1, 0, 1, \ldots, m-1\}$ and a Boolean flag f. Let Q denote the set of all such pairs (v, f). The values $\perp_0$ and $\perp_1$ are used to indicate that the agent has given its value to another agent and is no longer active in exchanging values; the subscript indicates the agent's output value. The initial state of an agent with input σ is $(c_\sigma \bmod m, 1)$. The output for states $(r, 0)$ and $(\perp_1, 0)$ is 1. The output for all other states is 0. The transitions M1 to M10 are given below, where v_1 and v_2 are any values in $\{0, 1, \ldots, m-1\}$, i is any value in $\{0, 1\}$ and q_1 and q_2 are any states. All arithmetic is done modulo m. An asterisk $(*)$ is used as a wildcard to match any value, and indicates that part of the state is not changed by the transition.

$$\begin{aligned}
(v_1, 1), (v_2, 1) &\rightarrow (v_1 + 1, 1), (v_2 - 1, 1) && \text{(M1)}\\
(*, 1), (*, *) &\rightarrow (*, 0), (*, *) && \text{(M2)}\\
(*, 0), (*, 1) &\rightarrow (*, 1), (*, 1) && \text{(M3)}\\
(v_1, 0), (v_2, 0) &\rightarrow (v_1 + v_2, 0), (0, 0) && \text{(M4)}\\
(v_1, 0), (0, 0) &\rightarrow (v_1, 0), (\perp_0, 0) && \text{(M5)}\\
(\perp_i, *), (*, 1) &\rightarrow (0, 0), (*, 1) && \text{(M6)}\\
(r, 0), (\perp_i, 0) &\rightarrow (r, 0), (\perp_1, 0) && \text{(M7)}\\
(v_1, 0), (\perp_i, 0) &\rightarrow (v_1, 0), (\perp_0, 0), \text{if } v_1 \neq r && \text{(M8)}\\
q_1, q_2 &\rightarrow q_2, q_1 && \text{(M9)}\\
q_1, q_2 &\rightarrow q_1, q_2 && \text{(M10)}
\end{aligned}$$

Transition M1 is the one that is crucial for privacy: it conceals inputs by shifting part of an agent's value to another agent, and can be invoked as long as the agents' flags are 1. Transitions M2 and M3 control the flags. Transition M4 gathers the sum into a single agent once the flags are 0, and Transition M5 ensures that exactly one agent ends up with a non-$\perp$ value. Transition M6 allows the $\perp$ values to be turned back to 0, reversing the effect of Transition M5 as long as flags are 1. Transitions M7 and M8 spread the output value from the (eventually unique) agent with a non-$\perp$ value to all other agents. Finally, Transitions M9 and M10 are included to satisfy the properties 3 and 4 of Theorem 1.

We first argue that this protocol correctly computes the predicate $P(I)$. Transition M2 ensures that, from any configuration, there is always a reachable configuration in which all flags are 0. Thus, any fair execution will eventually enter a configuration in which all flags are 0. After that point, all flags will remain 0 forever. Then, Transition M4 ensures that every agent except one will have a value that is either 0 or $\perp$. Transition M5 ensures that, eventually, exactly one agent will have a value different from $\perp$. (Transition M6 cannot be applied since all flags are 0.) Since the sum of the non-$\perp$ values stored in all agents (modulo m) is left invariant by all of the transitions, the one remaining non-$\perp$ value will be $\left(\sum_{i=0}^{n-1} c_{\sigma_i}\right) \bmod m$, so it will have the correct output value. Transitions M7 and M8 ensure that all other agents eventually stabilize with output $P(I)$ also.

We now show that the protocol satisfies the properties of Theorem 1. We choose $n_0 = 5$ and we define a configuration to be in $\mathcal{G}$ if and only if it has at least four agents, and agents $p_1, \ldots, p_4$ each have flag 1 and non-$\perp$ values. Note that any initial configuration with at least n_0 agents is good, so Property 1 of Theorem 1 is trivially satisfied. Property 3 is trivially satisfied, since the protocol includes Transition M9, which allows any pair of agents to swap states in a single interaction. Property 4 is also trivially satisfied, since the protocol includes Transition M10. It remains to show that every transition of the protocol is $\mathcal{G}$-imitable.

Consider any transition to be imitated. Suppose the curious agent interacts with an agent in state (v, f) in this transition. Let C_0 be any good configuration. We show how to drive agent p_1 into state (v, f), starting from configuration C_0. We consider two cases.

If v is a non-$\perp$ value, p_1 and p_2 meet repeatedly using Transition M1 until p_1 has value v. Then, if $f = 0$, p_1 sets its flag to 0 using Transition M2. At this point, p_1 has state (v, f).

If v is $\perp_0$ or $\perp_1$, p_1 and p_2 meet repeatedly using Transition M1 until p_1 has value 0. Then agents p_1 and p_2 set their flags to 0 using Transition M2, and meet once more using Transition M5 to set p_1's state to $(\perp_0, 0)$. If $v = \perp_1$, p_3 and p_4 meet using Transition M1 until p_3's state is $(r, 1)$, p_3 sets its flag to 0 using Transition M2, and then p_3 meets p_1 using Transition M7 to set p_1's state to $(\perp_1, 0)$. At this point, p_1's state is $(v, 0)$. If $f = 1$, then p_1 meets p_4 using Transition M3 to set its flag to 1. Then, p_1 will be in state (v, f).

Once the agent p_1 has been driven into state (v, f), it has the necessary interaction with p_0. After that, we must describe how to drive the system back into a good configuration. The above procedure leaves p_4 with a non-$\perp$ value and flag 1. Thus any of p_1, p_2, p_3 that have values $\perp_0$ or $\perp_1$ can meet p_4 using Transition M6 to get state $(0, 1)$. Then any of p_1, p_2, p_3 that have flag 0 can meet p_4 using Transition M3 to set their flags to 1. The resulting configuration is good. ■

3.2 Computing Threshold Predicates

Let Σ be an input alphabet. Let c_σ be an integer constant for each $\sigma \in \Sigma$ and let k be an integer constant. The predicate $P(I)$ that is 1 on input $I = (\sigma_0, \ldots, \sigma_{n-1})$ if and only if $\sum_{i=0}^{n-1} c_{\sigma_i} \geq k$ is called a *threshold predicate*. In this section, we show that any threshold predicate can be computed privately. We begin with the special case where the threshold k is positive.

Proposition 5 *Any threshold predicate with a positive threshold k can be computed by a protocol satisfying the properties of Theorem 1.*

Proof: Let $m = 2 \cdot \max(\{|c_\sigma| : \sigma \in \Sigma\} \cup \{k\})$. Each agent will store a value between $-m$ and m.

The general approach used to construct this algorithm is similar to the one used in Section 3.1 to compute remainder predicates privately. Each agent stores a value and a flag bit, and while flags are 1, the agents can shift parts of their values to each other. Eventually, the flags will all be set to 0, and the algorithm will compute the sum.

For remainder predicates, the sum (modulo m) could be stored in a single agent. For threshold predicates, we cannot use modular arithmetic, so the sum may end up spread across several agents. If the sum is positive, eventually, some number of agents (possibly 0) have the value m, at most one other agent has a positive value, and the remaining agents have value 0. On the other hand, if the sum is negative, all agents will eventually have non-positive values.

Because the sum is not collected into a single agent, distributing the output value to all agents is more complicated than in Section 3.1. Each agent stores an output bit. As long as the agent's flag is 1, its output bit is meaningless, so by convention we require it to be 0. Once an agent's flag is 0, the value of the output bit behaves as follows. If an agent's value is at least k, its output bit must be 1. If an agent's value is negative, its output bit must be 0. Otherwise, an agent's output bit can be either 0 or 1: in this case, the agent will determine its output bit from its interactions.

We now give a full description of the algorithm. The state of each agent is a triple (v, o, f) where $-m \le v \le m$, and o and f are Boolean values representing the output bit and flag bit, respectively. As described in the previous paragraph, not all triples are legal states: the output bit can take values 0 and 1 only when $f = 0$ and $0 \le v < k$. Initially, the state of an agent with input symbol σ is $(c_\sigma, 0, 1)$. The transitions T1 to T8 are given below, where v_1 and v_2 are any values between $-m$ and m and q_1 and q_2 are any states. (The notation $[v_1 \ge k]$ in Transition T2 indicates that the output bit should be set to 1 if and only if $v_1 \ge k$.)

$$(v_1,0,1),(v_2,0,1) \rightarrow (v_1+1,0,1),(v_2-1,0,1), \text{ if } v_1 < m \text{ and } v_2 > -m \tag{T1}$$
$$(v_1,0,1),(*,*,*) \rightarrow (v_1,[v_1 \ge k],0),(*,*,*) \tag{T2}$$
$$(*,*,0),(*,0,1) \rightarrow (*,0,1),(*,0,1) \tag{T3}$$
$$(v_1,*,0),(v_2,*,0) \rightarrow \left\{ \begin{array}{ll} (m,1,0),(v_1+v_2-m,1,0) & \text{if } m \le v_1+v_2 \le 2m \\ (v_1+v_2,1,0),(0,1,0) & \text{if } k \le v_1+v_2 < m \\ (v_1+v_2,0,0),(0,0,0) & \text{if } -m \le v_1+v_2 < k \end{array} \right\}, \text{ if } v_1 v_2 \ne 0 \tag{T4}$$
$$(v_1,1,0),(v_2,0,0) \rightarrow (v_1,1,0),(v_2,1,0), \text{if } v_1 \ge k \text{ and } 0 \le v_2 < k \tag{T5}$$
$$(v_1,0,0),(v_2,1,0) \rightarrow (v_1,0,0),(v_2,0,0), \text{if } v_1 < k \text{ and } 0 \le v_2 < k \tag{T6}$$
$$q_1,q_2 \rightarrow q_2,q_1 \tag{T7}$$
$$q_1,q_2 \rightarrow q_1,q_2 \tag{T8}$$

Transitions T1, T2 and T3 play the same role as Transitions M1, M2 and M3 in Section 3.1. Values are collected into a smaller number of agents using Transition T4. The output value is distributed using Transitions T5 and T6. Transitions T7 and T8 are included to satisfy Properties 3 and 4 of Theorem 1.

We first argue that this protocol correctly computes $P(I)$. Transition T2 ensures that, eventually, all flags are 0. After this point, they will always be 0. We focus on what happens after all flags become 0. Notice that all transitions preserve the sum of values. We consider several cases.

First, consider the case where the sum of all inputs is negative. We now argue that Transition T4 ensures that, beyond some time, all agents will have non-positive values. Once all flags have become 0, only Transition T4 alters the multiset of values stored in agents' states. It is easy to check that an application of this transition cannot increase the sum of all positive values. Furthermore, the sum of all positive values decreases whenever agents with oppositely signed values meet. Because the sum of all agents is invariant, there must always be an agent whose value is negative. Thus, the sum of all positive values will eventually be 0, so all values will be either 0 or negative beyond that time. Since there is always an agent with a strictly negative value (and therefore a 0 output bit), Transition T6 ensures that a configuration in which all agents have output bit 0 is always reachable. Fairness guarantees that such a configuration will eventually be reached. At this point the algorithm has stabilized with output 0.

Now, consider the case where the sum input values is non-negative. Once all flags have become 0, the sum of negative values is non-decreasing, and there is an increase in this sum whenever two agents with oppositely signed values meet. Since the sum of values is invariant, there is always an agent with positive value, so the sum of negative values is eventually 0. This means all agents have non-negative values beyond that time. We now consider two subcases.

If the sum of values is less than k, then all agents must have values between 0 and $k-1$ (inclusive). Transition T4 will ensure that, eventually, only one agent has a positive value, and all others have value 0. After this time, only Transitions T4, T6, T7 and T8 can be applied. Agents meeting according to Transition T4 will both get output bit 0 and applications of Transition T6 can only increase the number of agents with output bit 0, so eventually, all agents will have output bit 0 forever, as required.

If the sum of values is greater than or equal to k, we consider the time when all agents' values have become non-negative and remain so forever. Then, Transition T4 ensures that, eventually, each agent, except possibly one, has value 0 or m. Beyond that point, any application of Transition T4 will have $v_1 + v_2 \geq k$, so output bits of both agents will be set to 1. Also, at least one agent will always have value at least k (and therefore output bit 1), so Transition T5 will ensure that all agents eventually stabilize with output bit 1.

This completes the proof that the protocol computes $P(I)$. We now show that the protocol satisfies the properties of Theorem 1. We choose n_0 to be 12. We define a configuration to be in $\mathcal{G}$ if and only if it has at least 6 agents, the flags of agents $p_1, \ldots, p_5$ are all 1, and the sum of the values of agents $p_0, \ldots, p_5$ is equal to 0.

First we establish Property 1 of Theorem 1. Consider any initial configuration that has at least 12 agents. We describe how to drive the system into a good configuration without using any interactions involving p_0. For $i = 1, 2, 3, 4, 5$, agents p_i and p_{i+5} interact using Transition T1 until each of the agents $p_1, \ldots, p_5$ have value 0. Then, p_5 interacts with p_{11} using Transition T1 until its value is the negation of p_0's value. The resulting configuration is good.

Next, we show that the protocol satisfies Property 2 of Theorem 1. Consider any transition to be imitated. Suppose the curious agent interacts with an agent in state (v, o, f) in this transition. Let C_0 be any good configuration. We show how to drive agent p_1 into state (v, o, f), starting from C_0. First, $p_1, \ldots, p_5$ meet using Transition T1 until p_1, p_2, p_3 and p_4 each have value 0. (This is possible, since the sum of values of $p_1, \ldots, p_5$ in configuration C_0 is equal to the opposite of the value of p_0, so the sum is between $-m$ and m.) Next, p_1 and p_2 meet repeatedly, using Transition T1 until p_1 has value v. If $f = 0$, p_1 and p_2 meet again, this time using Transition T2, to change p_1's flag to 0. If, at this point, p_1's output bit differs from o, we must have $0 \leq v < k$ and $o = 1$. In this case, p_3 and p_4 meet repeatedly using Transition T1 until p_3's value is k, then once more to change p_3's flag to 0 using Transition T2, and then p_3 and p_1 meet using Transition T5 to change p_1's output bit to 1. When all of these interactions have occurred, p_1 is in state (v, o, f).

Next, p_0 and p_1 have their interaction. Now, we must restore the system to a good configuration. In C_0, the sum of the values of $p_0, \ldots, p_5$ was 0, since $C_0 \in \mathcal{G}$. All interactions since C_0 have been among agents $p_0, \ldots, p_5$ and every transition preserves the sum of the values of the two interacting agents. Thus, the sum of the values of agents $p_0, \ldots, p_5$ is still 0. The interactions since C_0 may have changed at most three agents' flags from 1 to 0. Since $p_1, \ldots, p_5$ all had flag 1 in C_0, there is at least one agent whose flag is still 1. If any of $p_1, \ldots, p_5$ have flag 0, those agents meet an agent whose flag is 1 using Transition T3 to set their flags back to 1. The resulting configuration is good.

Properties 3 and 4 of Theorem 1 are trivial, since the protocol has Transitions T7 and T8. ■

Corollary 6 *Any threshold predicate can be computed by a protocol satisfying the properties of Theorem 1.*

Proof: We have already described how to compute any threshold predicate with a positive threshold k. To compute a threshold predicate with a threshold $k \leq 0$, notice that $\sum_{i=0}^{n-1} c_{\sigma_i} \geq k$ if and only if $\sum_{i=0}^{n-1} (-c_{\sigma_i}) \not\geq -k+1$. Since $-k+1 > 0$, we can compute the threshold predicate $\sum_{i=0}^{n-1} (-c_{\sigma_i}) \geq -k+1$ as described in the proof of Proposition 5 and negate the result. ■

3.3 Computing All Semilinear Predicates

To complete the proof of Theorem 1, we show that the properties of the theorem can be preserved when computing Boolean combinations of predicates.

Theorem 7 *If predicates P^1 and P^2 can be computed by population protocols which satisfy the properties of Theorem 1, then there are population protocols that compute $\neg P^1$ and $P^1 \vee P^2$, also satisfying the properties of Theorem 1.*

Proof: The required population protocol for $\neg P^1$ is obtained by simply negating the output map of the protocol for P^1.

We now construct the required population protocol for computing $P^1 \vee P^2$. Let $A^1 = (Q^1, \delta^1, \iota^1, \omega^1)$ and $A^2 = (Q^2, \delta^2, \iota^2, \omega^2)$ be the population protocols for P^1 and P^2, respectively. The protocol for $P^1 \vee P^2$ is quite straightforward: it simply runs the algorithms A^1 and A^2 in parallel. Each agent's state will contain two components, one representing the state of the agent in each of the two algorithms. Whenever two agents meet, they have an interaction from the first algorithm, using the first components of their states, and an interaction from the second algorithm, using the second components of their states.

More formally, this protocol has the form $A = (Q, \delta, \iota, \omega)$, where

$$\begin{aligned}
Q &= Q^1 \times Q^2, \\
\iota(\sigma) &= (\iota^1(\sigma), \iota^2(\sigma)), \\
\omega(q) &= \omega^1(q) \vee \omega^2(q), \text{ and} \\
\delta &= \{((q^1, q^2), (r^1, r^2), (s^1, s^2), (t^1, t^2)) : (q^1, r^1, s^1, t^1) \in \delta^1 \text{ and } (q^2, r^2, s^2, t^2) \in \delta^2\}.
\end{aligned}$$

Since A^1 and A^2 satisfy Property 4 of Theorem 1, this definition of δ allows two agents who have an interaction to update the first or second halves of their states according to the transition relation of A^1 or A^2, respectively, while leaving the other halves of their states unchanged. Similarly, because A^1 and A^2 satisfy Property 3, this definition of δ allows two agents to swap the first or second half of their states while leaving the other halves unchanged. These facts are useful in some of the constructions we give below. If $C = ((q_1^1, q_1^2), (q_2^1, q_2^2), \ldots, (q_n^1, q_n^2))$ is a configuration of algorithm A, we use the notation C^1 for $(q_1^1, q_2^1, \ldots, q_n^1)$ and C^2 for $(q_1^2, q_2^2, \ldots, q_n^2)$. Also, we write $C = (C^1, C^2)$.

We first argue that this algorithm A stably computes the predicate $P^1 \vee P^2$. Consider any fair execution $E = (C_0^1, C_0^2), (C_1^1, C_1^2), (C_2^1, C_2^2), \ldots$ of A on some input I of size n. We show that $E^1 = C_0^1, C_1^1, C_2^1, \ldots$ is a fair execution of A^1. By the definition of A, C_0^1 is the initial configuration of A^1 on input I, and for all i, $C^1_i \to C^1_{i+1}$, according to the transition relation of A^1. To see that E^1 is fair, suppose some configuration C^1 appears infinitely often in the execution and $C^1 \to D^1$ is a possible transition of A^1. Since there are only a finite number of possible configurations of A^2 with n agents, some configuration (C^1, C^2) must appear infinitely often in E. Because A^2 satisfies Property 4 of Theorem 1, $(C^1, C^2) \to (D^1, C^2)$ is a possible transition of A. Since E is fair, (D^1, C^2) must appear infinitely often in E. Thus, D^1 appears infinitely often in E^1, as required. A symmetric argument proves that $C^2_0, C^2_1, C^2_2, \ldots$ is a fair execution of A^2. Thus, after some point, if any agent is in state (q^1, q^2), we must have $\omega^1(q^1) = P^1(I)$ and $\omega^2(q^2) = P^2(I)$, so $\omega(q^1, q^2) = \omega^1(q^1) \vee \omega^2(q^2) = P^1(I) \vee P^2(I)$.

In the remainder of this proof, we show that the algorithm A satisfies the properties of Theorem 1. Choose n^1_0 and $\mathcal{G}^1$ to satisfy the properties of Theorem 1 for A^1. Choose n^2_0 and $\mathcal{G}^2$ to satisfy the properties of Theorem 1 for A^2. Let $n_0 = \max(n^1_0, n^2_0)$. Let $\mathcal{G}$ be the set of configurations C where the first components of the elements of C form a configuration in $\mathcal{G}^1$ and the second components of elements of C form a configuration in $\mathcal{G}^2$. (*I.e.*, $\mathcal{G} = \{(C^1, C^2) : C^1 \in \mathcal{G}^1 \text{ and } C^2 \in \mathcal{G}^2\}$.) We shall show that n_0 and $\mathcal{G}$ satisfy the properties of Theorem 1 for A.

First, we show that A has Property 1. Consider any input configuration $C_0 = (C^1_0, C^2_0)$ for algorithm A that has size at least n_0. Then, C^1_0 is an input configuration of A^1 with at least $n_0 \geq n^1_0$ agents. There exists an execution fragment of A^1 that starts from C^1_0 and leads to a configuration $C^1 \in \mathcal{G}^1$. Thus, there is an execution fragment of A that starts from (C^1_0, C^2_0) and leads to (C^1, C^2_0). Since C^2_0 is an input configuration of A^2 with at least $n_0 \geq n^2_0$ agents, there is also an execution fragment of A^2 that starts from C^2_0 and leads to a configuration $C^2 \in \mathcal{G}^2$. Thus, there is an execution fragment of A that starts from (C^1, C^2_0) and leads to $(C^1, C^2) \in \mathcal{G}$. Concatenating the two execution fragments of A establishes Property 1 of Theorem 1 for protocol A.

Next, we show that A has Property 2. Consider any transition $(q^1, q^2), (r^1, r^2) \to (s^1, s^2), (t^1, t^2)$. Let $C = (C^1, C^2)$ be any good configuration of A in which p_0 has state (q^1, q^2). Then, $C^1 \in \mathcal{G}^1$ and $C^2 \in \mathcal{G}^2$.

Since p_0 is in state q^1 in C^1, there is an execution fragment of A^1 starting from C^1 and ending in a good configuration G^1 during which p_0 has a single interaction, which has transition $q^1, r^1 \to s^1, t^1$. Let p_i be the agent that p_0 has its interaction with. Let D^1 and F^1 be the configurations immediately before and after p_0's interaction in this fragment. Then, there is an execution fragment α_1 of A starting from (C^1, C^2) and ending in (D^1, C^2) during which p_0 has no interactions. (The interactions in α_1 only affect the first components of agents' states.)

Since p_0 is in state q^2 in C^2, there is an execution fragment of A^2 starting from C^2 and ending in a good configuration G^2 during which p_0 has a single interaction of the form $q^2, r^2 \to s^2, t^2$. Let p_j be the agent that p_0 has its interaction with. Let D^2 and F^2 be the configurations immediately before and after p_0's interaction in this fragment. Then, there is an execution fragment α_2 of A starting from (D^1, C^2) and ending in (D^1, D^2) during which p_0 has no interactions. (The interactions in α_2 only affect the second components of agents' states.)

If $i \neq j$, let β_1 be an execution fragment starting from (D^1, D^2) in which p_i and p_j swap the second components of their states. (Otherwise, let β_1 be an empty execution fragment.) At the end of β_1, agent p_i is in state (r^1, r^2). Let β_2 be an execution fragment starting from the end of β_1 consisting of a single interaction between p_0 and p_i, applying the transition $(q^1, q^2), (r^1, r^2) \to (s^1, s^2), (t^1, t^2)$. If $i \neq j$, let β_3 be an execution fragment starting from the final configuration of β_2 in which p_i and p_j swap the second components of their states. (Otherwise, let β_3 be an empty execution fragment.) Then, at the end of $\beta_1 \cdot \beta_2 \cdot \beta_3$, the configuration of the system is (F^1, F^2).

There is an execution fragment γ_1 of A starting from (F^1, F^2) and ending in (G^1, F^2) during which p_0 has no interactions. (The interactions in γ_1 affect only the first halves of agents' states.) There is also an execution fragment γ_2 of A starting from (G^1, F^2) and ending in (G^1, G^2) during which p_0 has no interactions. (The interactions in γ_2 affect only the second halves of agents' states.)

Putting these fragments together, we obtain the fragment $\alpha_1 \cdot \alpha_2 \cdot \beta_1 \cdot \beta_2 \cdot \beta_3 \cdot \gamma_1 \cdot \gamma_2$ of A, which starts from configuration (C^1, C^2), ends in $(G^1, G^2) \in \mathcal{G}$, and during which p_0 has exactly one interaction, which has transition $(q^1, q^2), (r^1, r^2) \to (s^1, s^2), (t^1, t^2)$. Thus, this transition is $\mathcal{G}$-imitable. This completes the proof of Property 2 for A.

Next, we establish Property 3 for A. Let (q_1^1, q_1^2) and (q_2^1, q_2^2) be any two states of Q. There is a sequence of interactions of A^1 between two agents that start in states q_1^1 and q_2^1 and end in states q_2^1 and q_1^1, respectively. Thus, there is a sequence of interactions of A between two agents that start in states (q_1^1, q_1^2) and (q_2^1, q_2^2) and end in states (q_2^1, q_1^2) and (q_1^1, q_2^2), respectively. Also, there is a sequence of interactions of A^2 between two agents that start in states q_1^2 and q_2^2 and end in states q_2^2 and q_1^2, respectively. So there is a sequence of interactions of A between two agents that start in states (q_2^1, q_1^2) and (q_1^1, q_2^2) and end in states (q_2^1, q_2^2) and (q_1^1, q_1^2), respectively. Concatenating the two sequences of interactions of A yields the required sequence that starts with two agents in states (q_1^1, q_1^2) and (q_2^1, q_2^2) and ends with the agents in states (q_2^1, q_2^2) and (q_1^1, q_1^2), respectively. Thus, A satisfies Property 3 of Theorem 1.

Finally, Property 4 of Theorem 1 for A follows trivially from the definition of δ and the fact that both A^1 and A^2 have this property. ■

Putting together all of the preceding results yields a proof of Theorem 1. It is known that every predicate computable in the population protocol model can be expressed as a Boolean combination of remainder and threshold predicates [4]. It follows from Proposition 4, Corollary 6 and Theorem 7 that all such predicates can be computed by a protocol that satisfies the properties of Theorem 1. (Notice that in no case do we ever choose a value of n_0 that is greater than 12, so the choice of n_0 does not depend on the predicate to be computed.)

4 Concluding Remarks

Although we restricted attention to computing predicates, the techniques can be applied to any function. Let $f : \mathcal{D} \to Y$ be any function that is computable by a population protocol without privacy. Then, for each $y \in Y$, define a predicate $P_y(x)$ to be 1 if and only if $f(x) = y$. This predicate can be computed, and can therefore be computed privately. All of the (finitely many) predicates P_y can be computed in parallel using the same approach as in Section 3.3 to yield a private protocol for computing f.

This work is a first step towards studying private mobile computing. Several directions for future research are appealing. Some seem fairly accessible. For instance, one could show that our obfuscation procedure can also be effective against a dynamic adversary that can control several agents on the fly. None of these agents will be able to determine the input values of the other agents, either individually or collectively. Other problems appear more difficult. It is not clear whether it is possible to devise an obfuscation procedure that would work if the adversary need only eventually converge towards obtaining knowledge of the inputs of other agents, without necessarily knowing when the correct input values have been discovered. We have restricted attention to problems where all agents produce the same output, but one could also consider problems that require agents to output different values. Some papers have altered the basic model of population protocols by putting a probability distribution on the possible transitions. Can we design protocols that would protect privacy with high probability, even if the adversary knows the probability distribution? It would also be intriguing to figure out how the agents should be strengthened to hide their inputs from an active adversary, who can cause agents to diverge from their protocol.

References

[1] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In *Proc. 1st IEEE International Conference on Distributed Computing in Sensor Systems*, volume 3560 of *LNCS*, pages 63–74, 2005.

[2] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, Mar. 2006.

[3] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. In *Proc. 20th International Symposium on Distributed Computing*, volume 4167 of *LNCS*, pages 61–75, 2006.

[4] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing.* To appear.

[5] D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing behavior in networks of nondeterministically interacting sensors. In *Proc. 9th International Conference on Principles of Distributed Systems*, 2005.

[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *Proc. 2nd IEEE International Conference on Distributed Computing in Sensor Systems*, volume 4026 of *LNCS*, pages 51–66, 2006.

[7] M. Fischer and H. Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Proc. 10th International Conference on Principles of Distributed Systems*, number 4305 in LNCS, pages 395–409, 2006.

[8] O. Goldreich. *Foundations of Cryptography*, volume 2, chapter 7. Cambridge University Press, 2004.

[9] F. Marryat. *Peter Simple*, volume 3, chapter I. Saunders and Otley, 1834.

[10] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes-Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, Warszawa, 1929.