Towards Efficient Private Distributed Computation on Unbounded Input Streams*

(Extended Abstract)

Shlomi Dolev[1], Juan Garay[2], Niv Gilboa[3],
Vladimir Kolesnikov[4], and Yelena Yuditsky[1]

[1] Department of Computer Science, Ben Gurion University of the Negev, Israel
{dolev,yuditsky}@cs.bgu.ac.il
[2] AT&T Labs – Research, Florham Park, NJ
garay@research.att.com
[3] Deptartment of Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel
niv.gilboa@gmail.com
[4] Bell Laboratories, Murray Hill, NJ
kolesnikov@research.bell-labs.com

Abstract. In the problem of private "swarm" computing, n agents wish to securely and distributively perform a computation on common inputs, in such a way that even if the entire memory contents of some of them are exposed, no information is revealed about the state of the computation. Recently, Dolev, Garay, Gilboa and Kolesnikov [ICS 2011] considered this problem in the setting of information-theoretic security, showing how to perform such computations on input streams of *unbounded length*. The cost of their solution, however, is exponential in the size of the Finite State Automaton (FSA) computing the function.

In this work we are interested in efficient (i.e., polynomial time) computation in the above model, at the expense of *minimal* additional assumptions. Relying on the existence of one-way functions, we show how to process unbounded inputs (but of course, polynomial in the security parameter) at a cost *linear* in m, the number of FSA states. In particular, our algorithms achieve the following:

- In the case of (n, n)-reconstruction (i.e., in which all n agents participate in the reconstruction of the distributed computation) and at most $n - 1$ agents are corrupted, the agent storage, the time required to process each input symbol, and the time complexity for reconstruction are all $O(mn)$.
- In the case of $(n - t, n)$-reconstruction (where only $n - t$ agents take part in the reconstruction) and at most t agents are corrupted, the agents' storage and time required to process each input symbol are $O(m\binom{n-1}{n-t})$. The complexity of reconstruction is $O(mt)$.

* This research has been supported by the Israeli Ministry of Science and Technology (MOST), the Institute for Future Defense Technologies Research named for the Medvedi, Shwartzman and Gensler Families, the Israel Internet Association (ISOC-IL), the Lynne and William Frankel Center for Computer Science at Ben-Gurion University, Rita Altura Trust Chair in Computer Science, *Israel Science Foundation* (grant number 428/11), Cabarnit Cyber Security MAGNET Consortium, MAFAT and Deutsche Telekom Labs at BGU.

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 69–83, 2013.
© Springer-Verlag Berlin Heidelberg 2013

We achieve the above through a carefully orchestrated use of pseudo-random generators and secret-sharing, and in particular a novel share re-randomization technique which might be of independent interest.

1 Introduction

Distributed computing has become an integral part of a variety of systems, including cloud computing and "swarm" computing, where n agents perform a computation on common inputs. In these emerging computing paradigms, security (i.e., privacy and correctness) of the computation is of a primary concern. Indeed, in swarm computing, often considered in military contexts (e.g., unmanned aerial vehicle (UAV) operation), security of the data and program state is of paramount importance; similarly, one persistent challenge in the field of cloud computing is ensuring the privacy of users' data, demanded by government, commercial, and even individual cloud users.

In this work, we revisit the notion of *perennial* private distributed computation, first considered by Dolev, Garay, Gilboa and Kolesnikov [8]. In such a computation, an unbounded sequence of commands (or inputs) are interpreted by several machines (agents) in a way that no information about the inputs as well as the state of the computation is revealed to an adversary who is able to "corrupt" the agents and examine their internal state, as long as up to a predetermined threshold of the machines are corrupted.

Dolev *et al.* were able to provide very strong (unconditional, or information-theoretic) security for computations performed by a finite-state machine (FSA), at the price however of the computation being efficient only for a small set of functions, as in general the complexity of the computation is exponential in the size (number of states) of the FSA computing the function.

In this work, we minimally weaken the original model by additionally assuming the existence of one-way functions (and hence consider polynomial-time adversaries—in the security parameter; more details below), and in return achieve very high efficiency in some cases as a function of the size of the FSA. We stress that we still consider computation on *a priori* unbounded number of inputs, and where the online (input-processing) phase incurs *no communication*. We now describe the model in more detail.

The setting. As in [8], we consider a distributed computation setting in which a party, whom we refer to as *the dealer*, has a finite state automaton (FSA) $\mathcal{A}$ which accepts an (*a priori* unbounded) stream of inputs $x_1, x_2, \ldots$ received from an external source. The dealer delegates the computation to agents $A_1, \ldots, A_n$, by furnishing them with an implementation of $\mathcal{A}$. The agents receive, in a synchronized manner, all the inputs for $\mathcal{A}$ during the online input-processing phase, where no communication whatsoever is allowed. Finally, given a signal from the dealer, the agents terminate the execution, submit their internal state to the dealer, who computes the state of $\mathcal{A}$ and returns it as output.

We consider an attack model where an entity, called the adversary, Adv, is able to adaptively "corrupt" agents (i.e., inspect their internal state) during the online execution phase, up to a threshold[1] $t < n$. We do not aim at maintaining the privacy of the

[1] We note that more general access structures may be naturally employed with our constructions.

automaton $\mathcal{A}$; however, we wish to protect the secrecy of the state of $\mathcal{A}$ and the inputs' history. We note that Adv may have external information about the computation, such as partial inputs or length of the input sequence, state information, etc. This auxiliary information, together with the knowledge of $\mathcal{A}$, may exclude the protection of certain configurations, or even fully determine $\mathcal{A}$'s state. We stress that this cannot be avoided in any implementation; thus, our goal is to prevent the leakage or derivation by Adv of any knowledge from seeing the execution traces that Adv did not already possess.

As mentioned above, our constructions relying on one-way functions dictates that the computational power of entities (adversary, agents), be polynomially bounded (in κ, the security parameter). Similarly, our protocols run on input streams of polynomial length. At the same time, we do not impose an *a priori* bound on its length; moreover, the size of the agents' state is independent of it. This allows to use agents of the same (small) complexity (storage and computational power) in all situations.

Our contributions. Our work is the first significant extension of the work of [8]. Towards the goal of making never-ending and private distributed computation practical, we introduce an additional (minimal) assumption of existence of one-way functions (and hence pseudo-random number generators [PRGs]), and propose the following constructions:

- A scheme with (n, n) reconstruction (where all n agents participate in reconstruction), where the storage and processing time per input symbol is $O(mn)$ for each agent. The reconstruction complexity is $O(mn)$.
- A scheme with $(n-t, n)$ reconstruction (where t corrupted agents do not take part in the reconstruction), where the above costs are $O(m\binom{n-1}{n-t})$.[2]

Regarding tools and techniques, the carefully orchestrated use of PRGs and secret-sharing techniques [17] allows our protocols to hide the state of the computation against an adaptive adversary by using share re-randomization. Typically, in the context of secret sharing, this is simply done by the addition of a suitable (i.e., passing through the origin) random polynomial. However, due to the no-communication requirement, share re-randomization is more challenging in our setting, particularly so in the more general case of the $(n-t, n)$-reconstruction protocol. We achieve share re-randomization by sharing PRG seeds among the players in a manner which allows players to achieve sufficient synchronization of their randomness, which is resilient to t corruptions.

Related work. Reflecting a well-known phenomenon in distributed computing, where a single point of failure needs to be avoided, a team of agents (e.g., UAVs) that collaborate in a mission is more robust than a single agent trying to complete a mission by itself (e.g., [1, 3]). Several techniques have been suggested for this purpose; another related line of work is that of automaton splitting and replication, yielding designs that can tolerate faults and as well as provide some form of privacy of the computation (see, e.g., [6–8, 10, 11]). As mentioned above, only [8] addresses the unbounded-input-stream scenario.

[2] For some values of t, e.g., $t = O(n)$, this quantity would be exponential in n. This however does not contradict our assumption on the computational power of the participants; rather, it simply means that, given κ, for some values of n and t this protocol cannot be executed in the allowed time.

Recall that in *secure multi-party computation* [2, 4, 13], n parties, some of which might be corrupted, are to compute an n-ary (public) function on their inputs, in such a way that no information is revealed about them beyond what is revealed by the function's output. At a high level, we similarly aim in our context to ensure the correctness and privacy of the distributed computation. However, as explained in [8], our setting is significantly different from that of MPC, and MPC definitions and solutions cannot be directly applied here. The reason is two-fold: MPC protects players *individual* inputs, whereas in our setting the inputs are common to all players. Secondly, and more importantly, MPC operates on inputs of fixed length, which would require an *a priori* estimate on the maximum input size s_{max} and agents' storage linear in s_{max}. While unbounded inputs could be processed, by for example processing them "in blocks," this would require communication during the online phase, which is not allowed in our setting. Refer to [8] for a more detailed discussion on the unbounded inputs setting *vis-à-vis* MPC's.

We note that using recently proposed fully-homomorphic encryption (FHE— [12] and follow-ups) trivially solves the problem we pose, as under FHE the agents can simply compute arbitrary functions. In fact, plain additively homomorphic encryption (e.g., [15]) can be used to encrypt the current state of the FSA and non-interactively update it as computation progresses, in a manner similar to what is described in our constructions (see the high-level intuition in Section 3). We note that, firstly, public-key encryption and, dramatically so, FHE, suffer from orders-of-magnitude computational overhead, as compared to the symmetric-key operations that we rely on. Perhaps more importantly, in this work we aim at minimizing the assumptions needed for efficient unbounded private distributed computation.

Finally, and as mentioned above, the problem of share re-randomization and conversion has been considered in the literature. Related to our setting, Cramer, Damgård and Ishai [5] for example consider the problem of locally converting a secret sharing of a value into another secret sharing of the same value.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2 we present in more detail the model, definitions and building blocks that we use throughout the paper. We dedicate Section 3 to a high-level description of our constructions, while in Section 4 we present them in detail. The full privacy analysis is presented in the full version of the paper [9].

2 Model and Definitions

A *finite-state automaton* (FSA) $\mathcal{A}$ has a finite set of states ST, a finite alphabet Σ, and a transition function $\mu : ST \times \Sigma \longrightarrow ST$. In this work we do not assume an initial state or a terminal state for the automaton, i.e., it may begin its execution from any state and does not necessarily stop.

We already described in the previous section the distributed computation setting—dealer, agents, adversary, and unbounded input stream—under which the FSA is to be executed. In more detail, we assume a *global clock* to which all agents are synchronized. We will assume that no more than one input symbol arrives during any clock tick. By *input stream*, we mean a sequence of input symbols arriving at a certain schedule of clock ticks. Abusing notation, we will sometimes refer to the input without explicit

reference to the schedule. (We note that the global clock requirement can in principle be removed if we allow the input schedule to be leaked to Adv.)

We also mentioned that Adv is allowed to corrupt agents as the execution of the protocol proceeds. We consider the so-called *passive* or *semi-honest* adversary model, where corrupted agents can combine their views in order to learn protected information, but are not allowed to deviate from the protocol. Furthermore, each agent can be corrupted only once during an execution. When it does, Adv can view the entire contents of a corrupted agent's memory, but does not obtain any of the global inputs.

Incidentally, we consider event processing by an agent as an *atomic operation*. That is, agents cannot be corrupted during an execution of state update. This is a natural and easily achievable assumption, which allows us to not worry about some tedious details. The computation is then considered to be secure, if any two executions (possibly on different inputs and initial states—defined more formally below) are "similarly" distributed.

This model of security for distributed computation on unbounded input streams was introduced by Dolev *et al.* [8] as the *progressive corruption* model (PCM), allowing Adv to be computationally unbounded, and in particular requiring that the distributions of the two executions (again, more formally defined below) be identical.

In this work we use a variant of PCM, applying the following two weakenings to the PCM definition:

1. Rather than requiring that the distributions of executions be identical, we require them to be *computationally* indistinguishable. This means that we guarantee security only against polynomial-time-bounded adversaries.

2. We require indistinguishability of executions for the *same* corruption timeline (and, of course, different input streams). This means that, for example, agent IDs are now allowed to be included in the agents' views. (We use agent IDs in one of our constructions.) We stress that this is not a significant security weakening, as essentially we only allow the adversary to differentiate among the agents' identities; the inputs and current state of the computation remain computationally hidden.

We now present our amended PCM definition. We first formalize the notion of *corruption timeline* and the view of the adversary.

Definition 1. *A* corruption timeline *ρ is a sequence $\rho = ((A_1, \tau_1), \ldots, (A_k, \tau_k))$, where $A_1, \ldots, A_k$ are the corrupted agents and $\tau_1, \ldots, \tau_k$ ($\tau_1 \leq \ldots \leq \tau_k$) denote the time when the corresponding corruption took place. The* length *of a corruption timeline is $|\rho| = k$.*

We denote by $\mathrm{VIEW}_\rho^\Pi(X, s)$ the probability distribution of the aggregated internal states of corrupted agents at the time of corruption, when executed on input X and initial state s.

Definition 2 (Computational Privacy in the Progressive Corruption Model). *We say that a distributed computation scheme Π is* t-private in the Progressive Corruption Model (PCM) *if for every two states $s_1, s_2 \in ST$, polynomial-length input streams*

X_1, X_2, and any corruption timeline ρ, $|\rho| \leq t$,

$$\text{VIEW}_{\rho}^{\Pi}(X_1, s_1) \stackrel{\text{c}}{\approx} \text{VIEW}_{\rho}^{\Pi}(X_2, s_2).$$

Here, '$\stackrel{\text{c}}{\approx}$' denotes the computational indistinguishability of two distributions.

Tools and Building Blocks. A pseudo-random generator (PRG) is a function of the form $G : X \rightarrow Y$, where X and Y are typically of the form $\{0,1\}^k$ and $\{0,1\}^{k+l}$, respectively, for some positive integers k, l. Recall that PRGs are known to exist based on the existence of one-way functions, and that the security property of a PRG guarantees that it is computationally infeasible to distinguish its output on a value chosen uniformly at random from X from a value chosen uniformly at random from Y (see, e.g., [14]). In our setting, we will further assume that the old values of the PRG seeds are securely erased by the agents upon use and hence are not included in the view of the adversary.

The other basic tool that our protocols make use of is (n, t)-*secret sharing* [17], where, essentially, a secret piece of information is "split" into shares and handed out to a set of n players by a distinguished player called *the dealer*, in such a way that up to a threshold $t < n$ of the players pulling together their shares are not able to learn anything about it, while $t + 1$ are able to reconstruct the secret. We present the specific instantiations of secret sharing as needed in the corresponding sections.

3 Overview of Our Approach

Let $\mathcal{A}$ be a publicly known automaton with m states. We assume that we have some ordering of the states of $\mathcal{A}$, which are denoted by corresponding labels. Every agent stores the description of the automaton. In addition, during the computation, for every state s_j of $\mathcal{A}$, every agent A_i computes and stores its current label ℓ_j^i. As mentioned above, all agents receive a global input stream $\Gamma = \gamma_1, \gamma_2, ...\gamma_i, ...$ and perform computation in synchronized time steps.

At a high level, the main idea behind our constructions is that the state labels will be shares (*à la* secret sharing [17]) of a secret which identifies the currently active state of $\mathcal{A}$. More specifically, for each of the m automaton states, the n state labels (held by the n agents) will be shares of value 1 if the state is currently active, and shares of 0 otherwise. We will show how the players' local computation on their shares will ensure that this property is maintained throughout the computation on the entire input stream Γ. When the input stream Γ is fully processed (or a stop signal is issued), the agents recover the current state by reconstructing the secrets corresponding to each automaton state. At the same time, shares of the secrets (when not taken all together) reveal no information on the current state of $\mathcal{A}$.

We now present additional high-level details on two variants of the approach above. Recall that we consider the semi-honest adversary model, where corrupted players are not allowed to deviate from the protocol, but combine their views in order to learn protected information.

$(\mathbf{n}, \mathbf{n})$-reconstruction. In this scenario, we require that all n agents participate in the reconstruction of the secret (corrupted players are considered semi-honest and hence honestly provide their computed shares).

At the onset of computation, the shares are initialized using an (n, n) additive secret-sharing scheme, such that the initial state labels are the sharing of 1, and labels of each of the other states are shares of 0. When processing a global input symbol γ, each agent computes a new label for a state s by summing the previous labels of all states s' such that $\mu(s', \gamma) = s$. It is easy to see that, due to the fact that we use additive secret sharing, the newly computed shares will maintain the desired secret-sharing property. Indeed, say that on input symbol γ, u states transition into state s. If all of them were inactive and their labels were shares of 0's, then the newly computed shares will encode a 0 (as the sum of u 0's). Similarly, if one of the u predecessor states was active and its label shared a 1, then the new active state s will also correspond to a share of 1.

A technical problem arises in the case of "empty" states, i.e., those that do not have incoming transitions for symbol γ, and hence their labels are undefined. Indeed, to hide the state of the automaton from the adversary who corrupts agent(s), we need to ensure that each label is a random share of the appropriate secret. Hence, we need to generate a random 0-share for each empty state without communication among the agents.

In the (n, n) sharing and reconstruction scenario, we will non-interactively generate these labels pseudo-randomly as follows. Each pair of agents (A_i, A_j) will be assigned a random PRG seed $seed_{ij}$ Then, at each event (e.g., processing input symbol γ), each agent A_i will pseudo-randomly generate a string r_j using each of the seeds $seed_{ij}$, and set the label of the empty state to be the sum of all strings r_j. This is done for each empty state independently. The PRG seeds are then (deterministically) "evolved" thereby erasing from the agent's view the knowledge of the labels' provenance, and making them all indistinguishable from random. As all agents are synchronized with respect to the input and the shared seeds, it is easy to see that the shares generated this way reconstruct a 0, since each string r_j will be included twice in the total sum, and hence will cancel out (we will use an appropriate [e.g., XOR-based] secret-sharing scheme such that this is ensured.).

Finally, and intuitively, we observe that PCM security will hold since the view of each corrupted agent only includes pseudo-randomly generated labels for each state and the current PRG seed value. As noted above, even when combined with the views of other corrupted players, the labels are still indistinguishable from random.

$(\mathbf{n - t}, \mathbf{n})$-reconstruction. In this scenario, up to t corrupted agents do not take part in the reconstruction (this is motivated by the possibility of agents (UAVs) being captured or destroyed by the adversary). Agents who submit their inputs are doing so correctly. Thus, here we require $n > 2t$.

We will take our (n, n)-reconstruction solution as the basis, and adapt and expand it as follows. First, in order to enable reconstruction with $n - t$ $(= t + 1)$ agents, we will use (n, t) additive secret-sharing (such as Shamir's [17]). Second, as before, we will use a PRG to generate labels, but now we will have a separate seed for each subset of agents of size $n - t + 1$. Then, at each event (e.g., processing of an input symbol), each agent A_i, for each of the groups he belongs to, will update its shares by generating a random (n, t)-secret sharing of a 0 using the randomness generated by applying G to

the group's seed. Then, agent A_i will use the share thus generated for the i-th agent as its own, and set the label of the empty state to be the sum of all such shares.

Here we note that, since agents are excluded from some of the groups, and that in this scenario up to t agents might not return their state during reconstruction, special care must be taken in the generation of the re-randomizing polynomials so that all agents have invariantly consistent shares, *even for groups they do not belong to*, and that any set of agents of size $t+1$ enable the reconstruction of the secrets. (See Section 4.2 for details.) The above is done for each empty state independently. As before, the PRG seeds are then (deterministically) "evolved," making them all indistinguishable from random.

Algorithm 1: Template algorithm for agent A_i, $1 \leq i \leq n$, for label and state update.

Input: An input symbol γ.
Output: New labels for every state.
1: **if** γ is initialized **then**
2: $\ell_j^i := \sum_{k,\mu(s_k,\gamma)=s_j} \ell_k^i$ (the sum is calculated over some field $\mathbb{F}$, depending on the scheme).
3: **end if**
4: **for** every $T \in \mathcal{T}$ s.t. $A_i \in T$ **do**
5: Compute $B^T S^T \leftarrow G(seed_r^T)$, where $B^T = b_1^T b_2^T ... b_m^T$, and $b_j^T \in \mathbb{F}$, $1 \leq j \leq m$.
6: $seed_{r+1}^T := S^T$.
7: **for** $j = 1$ to m **do**
8: $\ell_j^i := \ell_j^i + R_j$, where R_j is a scheme-specific pseudo-random quantity.
9: **end for**
10: **end for**

Remark 1. This approach reveals the length and schedule of the input Γ processed by the players. Indeed, the stored seeds (or more precisely, their evolution which is traceable by the adversary simply by corrupting at different times players who share a seed) do reveal to the adversary the number of times the update function has been invoked. We hide this information by requiring the agents to run updates at each clock tick.

Algorithm 1 summarizes the update operations performed by agent A_i ($1 \leq i \leq n$) during the r-th clock cycle. The key point is the generation of R_j, the label re-randomizing quantity. Notice also that in every clock cycle, there may or may not be an input symbol received by the agent; if the agent did not receive any input, we assume that the input symbol is not initialized.

4 The Constructions in Detail

4.1 The (n, n)-Reconstruction Protocol

We start our formalization of the intuition presented above with the case where all n out of the n agents participate in the state reconstruction. The protocol for this case, which we call $\Pi^{(n,n)}$, is presented below.

Protocol $\Pi^{(n,n)}$. The protocol consists of three phases:

Initialization. The dealer secret-shares among the agents a secret value for each state, such that the value for the initial state is 1 and for all the other states is 0. This is done as follows. Agent A_i ($1 \leq i \leq n$) is given a a random binary string $x_1^i x_2^i ... x_m^i$, with the constraints that

$$x_{init}^1 + x_{init}^2 + ... + x_{init}^n \equiv 1 \bmod 2,$$

where $init$ is the index of the initial state of the computation, and for every $1 \leq j \neq init \leq m$,

$$x_j^1 + x_j^2 + ... + x_j^n \equiv 0 \bmod 2.$$

Each agent then proceeds to assign its state labels as $\ell_j^i \leftarrow x_j^i$.

Event Processing. Each agent runs Algorithm 1, updating its labels and computing the new seeds for the PRG. Let $\mathcal{T}$ be the set of all possible agents' pairs. For line 8 of Algorithm 1, each agent A_i now computes

$$R_j = \sum_{T \in \mathcal{T}, A_i \in T} (b_j^T)_r.$$

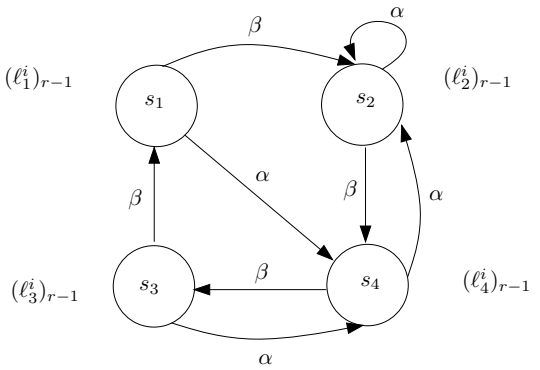

Fig. 1. *The internal state of agent A_i before a transition*

Reconstruction. All agents submit their internal states to the dealer, who reconstructs the secrets corresponding to each state, by adding (mod 2) the shares of each state, and determines and outputs the currently active state (the one whose reconstructed secret is 1).

Before proving the correctness and privacy achieved by the protocol, we illustrate the operation of the online (Event Processing) phase with the following example; refer to Figures 1 and 2. The two figures describe the execution of the protocol on an automaton with four states and two possible inputs. Figure 1 presents the internal state of agent A_i

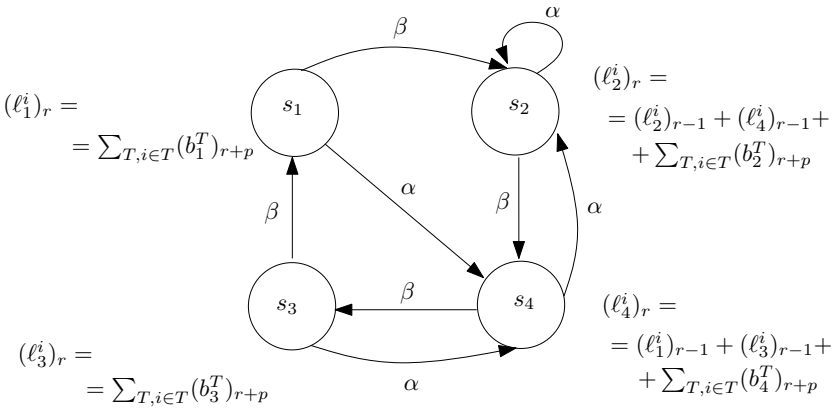

Fig. 2. *The internal state of agent A_i after an α transition*

after the $(r-1)$-th clock cycle. The agent holds the original automaton and has a label for each of the four states, $(\ell_1^i)_{r-1}$, $(\ell_2^i)_{r-1}$, $(\ell_3^i)_{r-1}$ and $(\ell_4^i)_{r-1}$.

Figure 2 shows the changes in the agent's internal state compared to Figure 1 after the r-th clock cycle. We also assume that in this clock cycle the agents receive an input symbol α. The new labels for each state are the sum of old labels and pseudo-random values. The labels in the sum are the old labels of all the states that transition to the current state given the input. Thus, the new $(\ell_2^i)_r$ includes a sum of the old $(\ell_2^i)_{r-1}$ and the old $(\ell_4^i)_{r-1}$, while the new $(\ell_3^i)_r$ doesn't include any labels in its sum because there is no state that transitions to s_3 after an α input. The pseudo-random addition to each state $j = 1, \ldots, 4$ is the sum $\sum_{T,i\in T}(b_j^T)_r$.

We start by proving the correctness of the construction.

Proposition 1. *At every Event Processing step of protocol $\Pi^{(n,n)}$, the secret corresponding to the current state in the computation is* 1 *and for all other states the secret is* 0*.*

Proof. The proof is by induction on the number of steps r that the automaton performs, i.e., the number of clock cycles.

For the base case, if we consider the state of the protocol after the initialization step and before the first clock cycle, i.e., at $r = 0$, then the statement is true by our definition of the label assignments. Let us first consider the case where at the r-th step an input symbol γ_r from Γ is received. Following the protocol, agent A_i's new label for state j becomes

$$\ell_j^i \longleftarrow \sum_{\substack{k\,:\\ \mu(s_k,\gamma_r)=s_j}} \ell_k^i + \sum_{A_i\in T} (b_j^T)_r.$$

Consider now the next state of the computation in the automaton; we wish to show that the secret corresponding to that state will be 1. Let $curr$ be the index of the current state of the automaton, and $next$ be the index corresponding to the next state; by definition,

$\mu(s_{curr}, \gamma_r) = s_{next}$. Then,

$$\ell^i_{next} \longleftarrow \sum_{\substack{k\,:\\ \mu(s_k,\gamma_r)=s_{next}}} \ell^i_k =$$

$$\ell^i_{curr} + \sum_{\substack{k\neq curr\,:\\ \mu(s_k,\gamma_r)=s_{next}}} \ell^i_k + \sum_{i\in T}(b^T_j)_r.$$

By the induction hypothesis, we know that

$$\sum_{i=0}^{n} \ell^i_{curr} \equiv 1 \pmod 2$$

and for $k \neq curr$,

$$\sum_{i=0}^{n} \ell^i_k \equiv 0 \pmod 2.$$

Thus, if we sum over all the agents:

$$\begin{aligned}
&\sum_{i=0}^{n}\left(\ell^i_{curr} + \sum_{\substack{k\neq curr\,:\\ \mu(s_k,\gamma_r)=s_{next}}} \ell^i_k + \sum_{i\in T}(b^T_j)_r\right)\\
&= \sum_{i=0}^{n}\ell^i_{curr} + \sum_{\substack{k\neq curr\,:\\ \mu(s_k,\gamma_r)=s_{next}}} \sum_{i=0}^{n}\ell^i_k\\
&+ \sum_{i=0}^{n}\sum_{i\in T}(b^T_j)_r \equiv 1+0 \equiv 1 \pmod 2.
\end{aligned}$$

This is because in $\sum_{i=1}^{n}\sum_{i\in T}(b^T_j)_r$, every $(b^T_j)_r$ appears exactly twice in this sum, once for every element in T. Using similar arguments one can see that all the other states will resolve to 0.

In the case that in the r-th step no input symbol is received, due to the fact that we just add the random strings in the same way as in the case above, we again get that the secret corresponding to the current state of the computation is 1, and for all others is 0. □

Proposition 2. *Protocol $\Pi^{(n,n)}$ is $(n-1)$-private in the PCM model according to Definition 2.*

Proof (sketch). Recall that the underlying observation is that when a corruption takes place (which cannot happen during the label-update procedure), the agent's state includes the current labels and PRG seeds which have already been evolved, and hence cannot be correlated with the label shares previously generated.

Without loss of generality, consider the case where Adv corrupts all but one agent according to an arbitrary corruption timeline, and assume, say, agent A_1 is not corrupted. We argue that the view of the adversary is indistinguishable from a view corresponding to (randomly) initialized agents $A_2, ..., A_n$ on the given automaton and any initial state. In other words, the view of the adversary is indistinguishable from the view he would obtain if he corrupted the agents simultaneously and before any input was processed. Once we prove that, the proposition follows.

The view of each corrupted agent includes $n-1$ seeds that he shares with other agents and the FSA labels which are secret shares of 0 or a 1. We argue that, from the point of view of the adversary, these labels are *random* shares of either 0 or 1. This follows from the PRG property that an evolved seed cannot be correlated with a prior output of the PRG, and from the fact that A_1 remains uncorrupted. Indeed, the newly generated "empty" states' labels look random since the adversary cannot link them to the PRG seeds in his view. The other states' labels look random to the adversary since they are XORed with A_1's label.

Thus, the total view of the adversary consists of random shares of 0 and 1, and is hence indistinguishable from the one corresponding to the initial state. □

We now calculate the time and storage complexity of $\Pi^{(n,n)}$. At every step of the computation, each agent pseudo-randomly generates and XORs $n-1$ strings. Further, each agent holds a small constant-length label for each automaton state, and $n-1$ PRG seeds, yielding an $O(m+n)$ memory requirement.

4.2 The $(n-t, n)$-Reconstruction Protocol

Recall that in this case, up to t of the agents might not take part in the reconstruction, and thus $n > 2t$.

A straightforward (albeit costly) solution to this scenario would be to execute $\Pi^{(n,n)}$ independently for every subset of agents of size $t+1$ (assuming for simplicity $n = 2t+1$). This would involve each agent A_i holding $\binom{n-1}{t}$ copies of the automaton $\mathcal{A}$, one copy for each such subset which includes A_i, and updating them all, as in $\Pi^{(n,n)}$, according to the same input symbol. Now, during the reconstruction, the dealer can recover the output from any subset of $t+1$ agents. The cost of this approach would be as follows. Every agent holds $\binom{n-1}{t}$ automata (one for every $t+1$ tuple that includes this agent), and executes $\Pi^{(n,n)}$, which requires $O(m+t)$ memory, resulting in a total cost of $O\left(\binom{n-1}{t} \cdot (m+t)\right)$, with the cost of computation per input symbol being proportional to storage's. In the sequel, we will refer to this approach as $\Pi^{(n-t,n)}_{\text{naive}}$.

We now present $\Pi^{(n-t,n)}$, an improved $(n-t, n)$ reconstruction scheme, whose intuition was already presented in Section 3. The protocol uses Shamir's secret-sharing scheme [17], which we now briefly review. Let $\mathbb{F}$ be a field of size greater than n, and $s \in \mathbb{F}$ be the secret. The dealer randomly generates coefficients $c_1, c_2, ..., c_t$ from $\mathbb{F}$ and construct the following polynomial of degree t, $f(x) = s + c_1 x + c_2 x^2 + ... + c_t x^t$. The dealer gives each participant A_i, $1 \leq i \leq n$, the value $f(i)$. It can be easily seen that one can reconstruct the secret from any subset of at least $t+1$ points, and no information about the secret is revealed by t points (or less).

Protocol $\Pi^{(n-t,n)}$. As before, the protocol consists of three phases:

Initialization. Using Shamir's secret sharing as described above, the dealer shares a secret 1 for the initial state and 0 for all other states. In addition, the dealer generates a random seed for every set of $n-(t-1)=n-t+1$ agents, and gives each agent the seeds for the sets it belongs to. Let $\mathcal{T}$ be the set of all possible subsets of $n-t+1$ agents.

Event Processing. Each agent runs Algorithm 1 updating its labels, as follows.

Let $T \in \mathcal{T}$ and j, $1 \leq j \leq m$, be a state of the automaton. Upon obtaining value b_j^T (refer to Algorithm 1), the agents in T (individually) construct a degree-t polynomial, P_j^T, by defining its value on the following $t+1$ field points: 0, all the points i such that $A_i \notin T$, and k such that k is the minimal agent's index in T (the choice of which point in T is arbitrary). Now define $P_j^T(0)=0$, $P_j^T(i)=0\ \forall A_i \notin T$, and $P_j^T(k)=b_j^T$.

Observe that by this definition, *every* agent $A_i \in T$ can use polynomial interpolation to compute $P_j^T(i)$, since the only required information is b_j^T (and the knowledge of set membership).

Let polynomial P_j be defined as $P_j = \sum_{T\in\mathcal{T}} P_j^T$. Each agent A_i now computes $P_j(i)$ (note that this is possible since the values corresponding to sets the agent does not belong to is set to 0), and updates the j-th label, $1 \leq j \leq m$, in Algorithm 1 by setting $R_j = P_j(i)$ in line 8.

Reconstruction. At least $t+1$ agents submit their internal state to the dealer, who, for every $j=1,\ldots,m$, views the j-th labels of $t+1$ agents as shares in a Shamir secret-sharing scheme. The dealer reconstructs all the m secrets using the scheme's reconstruction procedure, and determines and outputs the currently active state (whose recovered secret is equal to 1).

Proposition 3. *At every Event Processing step of protocol $\Pi^{(n-t,n)}$, the shared secret for the current state in the computation is 1 and for all the other (inactive) states, the shared secret is 0. Furthermore, $t+1$ agents can jointly reconstruct all secrets.*

Proof. We prove the proposition by induction on the number of clock cycles r. We show that at each clock cycle r, for every state s_j, the n labels $\ell_j^1,\ldots,\ell_j^n$ are points on a degree t polynomial Q_j whose free coefficient is 1 if j is the current state and 0 otherwise.

At initialization, the claim is true by our definition of the label assignments.

Assume that the induction hypothesis is correct after $r-1$. We prove the hypothesis for the r-th step. Assume first that in this step the agents receive an input letter γ_r, and denote the current state by s_{curr}. By our definition, the new label of the state j of agent i is

$$\ell_j^i \longleftarrow \sum_{\substack{k:\\ \mu(s_k,\gamma_r)=s_j}} \ell_k^i + P_j(i),$$

or, equivalently,

$$\ell_j^i \longleftarrow \sum_{\substack{k:\\ \mu(s_k,\gamma_r)=s_j}} Q_k(i) + P_j(i).$$

For every $j, 1 \leq j \leq m$, define polynomial Q'_j as

$$Q'_j = \sum_{\substack{k\ :\\ \mu(s_k,\gamma_r)=s_j}} Q_k + P_j.$$

Therefore, $Q'_j(i) = \ell^i_j$ for every j and every i. In addition, since every Q_k is of degree t and so is P_j, we deduce that Q'_j is also of degree t. We finish proving the induction step by showing that $Q'_j(0) = 1$ only for the correct state.

Let $\mu(s_{curr}, \gamma_r) = s_{next}$. By induction, $Q_{curr}(0) = 1$ and $Q_j(0) = 0$ for any $j \neq curr$. Furthermore, by construction $P_j(0) = 0$, and therefore $Q'_{curr}(0) = 1$. Since $Q_j(0) = 0$ for any $j \neq curr$, we have that $Q'_j(0) = 0$ for any $j \neq next$.

If the agents do not receive any input symbol in the r-th clock cycle, then the claim follows by similar arguments as above. □

Proposition 4. $\Pi^{(n-t,n)}$ *is* t*-private in the PCM model according to Definition 2.*

At a high level, the proof follows the steps of the proof of Proposition 2. The full details of the privacy analysis are presented in the full version of the paper [9].

We now calculate the costs incurred by the protocol. The space complexity of each agent is as follows. An agent holds a label for every state, i.e. $m \cdot (\lceil log|\mathbb{F}| \rceil + 1)$ bits. Additionally every agent holds $\binom{n-1}{n-t} = \binom{n-1}{t-1}$ seeds, where every seed is of size len. Thus, in total we have $\binom{n-1}{t-1} \cdot len + m \cdot (\lceil log|\mathbb{F}| \rceil + 1)$ bits. Each step of the Event Processing phase requires $O(m\binom{n-1}{t-1})$ time for seed manipulation and field operations. Reconstruction (by the dealer) is just interpolation of m polynomials of degree t.

References

1. Ben-Shahar, O., Dolev, S., Dolgin, A., Segal, M.: Direction Election in Flocking Swarms. In: Proc. of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing, pp. 73–80 (2010)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proc. 20th STOC, pp. 1–10 (1988)
3. Bamberger Jr., R., Watson, D., Scheidt, D., Moore, K.: Flight Demonstrations of Unmanned Aerial Vehicle Swarming Concepts. Johns Hopkins APL Technical Digest 27(1), 41–55 (2006)
4. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: Proc. 20th STOC, pp. 11–19 (1988)
5. Cramer, R., Damgård, I., Ishai, Y.: Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (2005)
6. Dolev, S., Gilboa, N., Kopeetsky, M., Persiano, G., Spirakis, P.: Information Security for Sensors by Overwhelming Random Sequences and Permutations. In: Proc. of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (2010)
7. Dolev, S., Garay, J., Gilboa, N., Kolesnikov, V.: Swarming Secrets. In: 47th Annual Allerton Conference on Communication, Control, and Computing (2009)
8. Dolev, S., Garay, J., Gilboa, N., Kolesnikov, V.: Secret Sharing Krohn-Rhodes: Private and Perennial Distributed Computation. In: Innovations in Computer Science (ICS), pp. 32–44 (2011)

9. Dolev, S., Garay, J., Gilboa, N., Kolesnikov, V., Yuditsky, Y.: Towards Efficient Private Distributed Computation on Unbounded Input Streams, Cryptology ePrint Archive, Report 2013/220
10. Dolev, S., Kopeetsky, M., Shamir, A.: RFID Authentication Efficient Proactive Information Security within Computational Security. Theory Comput. Syst. 48(1), 132–149 (2011)
11. Dolev, S., Lahiani, L., Yung, M.: Secret Swarm Unit Reactive k-Secret Sharing. Ad Hoc Networks 10(7), 1291–1305 (2012)
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. 41st STOC, pp. 169–178 (2009)
13. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Proc. 19th STOC, pp. 218–229 (1987)
14. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press (2000)
15. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
16. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: Proc. of the 7th ACM conference on Computer and Communications Security (CCS), pp. 245–254 (2000)
17. Shamir, A.: How to Share a Secret. Communications of the ACM 22(11), 612–613 (1979)