SWARMING SECRETS*

(EXTENDED ABSTRACT)

Shlomi Dolev, Juan Garay, Niv Gilboa, Vladimir Kolesnikov

***Abstract*—Information-theoretically secure schemes for sharing and modifying a secret among a dynamic swarm of computing devices are presented. The schemes securely and distributively maintain a global state for the swarm, and support an unlimited number of changes to the state according to (global) inputs received. We present proactively secure schemes supporting players *joining* and *leaving* the swarm. The schemes also allow swarms to be *merged, cloned* and *split*. We use the above schemes as a basis to implement an *oblivious* universal Turing machine, capable of evaluating dynamically specified functions.**

I. INTRODUCTION

There is a great interest in pervasive *ad hoc* and swarm computing [19], particularly in swarming Unmanned Aerial Vehicles (UAV) [14], [8]. A unit of UAVs that collaborate in a mission is more robust than a single UAV that has to complete a mission by itself. This is a known phenomenon in distributed computing where a single point of failure has to be avoided. Replicated memory and state machine abstractions are used as general techniques for capturing the strength of distributed systems in tolerating faults and dynamic changes.

Computing abstractions have been recently proposed to facilitate meaningful activities in a swarm of devices. In particular, the group communication abstraction [11] and the virtual automata abstraction (e.g., [7], [6], [5], [8], [9], [10], [18]), and the population protocol abstraction (e.g., [1], [4]) are prominent examples.

In this work we integrate security considerations to these abstractions, continuing the path of investigation proposed in [10]. We are interested in scenarios in which some of the swarm members are compromised and their secret shares are revealed. We would like the swarm members to execute a global swarm state transition *without knowing* the secret swarm state, before or after the transition. This computation may occur an unbounded number of times. We use secret sharing [17], and notions of secure multi-party computation (MPC) [20], [21], [12] as tools. We are specifically interested in computation that is secure in the information-theoretic sense [2], [3].

*Shlomi Dolev and Niv Gilboa are with the Department of Computer Science, Ben-Gurion University, dolev@cs.bgu.ac.il, niv.gilboa@gmail.com. Juan Garay is with AT&T Labs-Research, garay@research.att.com, and Vladimir Kolesnikov is with Bell Laboratories, kolesnikov@research.bell-labs.com. Partially supported by DIMACS, FRONTS EU Project, US Air Force European Office of Aerospace and Development, grant #FA8655-09-1-3016, the Rita Altura trust chair in computer sciences Deutche Telekom Laboratories at Ben-Gurion University of the Negev, and Lynne and William Frankel Center for Computer Sciences.

We investigate an extension of a threshold secret sharing scheme, in which the secret shares are modified over and over, on the fly, while maintaining the requirement that more than $t+1$ shares are required to reconstruct the secret while t or fewer shares provide no information on the secret. A secret is modified according to a previous (unrevealed) secret, an input (possibly distributively obtained) and a transition function.

The swarm entity supports secure *join* and *leave* operations of members. Whenever a new member requests to join the swarm, secret shares are sent to the new member such that a passive, honest-but-curious adversary that controls no more than t swarm members cannot learn the secret. The use of bivariate polynomials, a by now well established technique in information-theoretic MPC [2], allows us to support an unbounded number of *join* operations, which improves the *join* procedure suggested in [10]. Whenever a member leaves the swarm, proactive measures are taken to nullify the information which may be revealed by a share of the departing swarm member. A swarm can take such proactive actions periodically in order to cope with a dynamic adversary that compromises more than t parties over a long period of time, but no more than t concurrently. Such a dynamic adversary does not obtain any information on the state of the swarm as long as no more than t swarm members are compromised between any two successive proactive actions.

The swarm is also capable of adapting to different security levels, as a response, for example, to the hostility of the current environment. This is achieved by changing the threshold t of the number of shares needed to reconstruct a secret.

In addition to management operations on a single swarm, we propose possible operations among several swarms, including merging two swarms, splitting a swarm into two smaller swarms or cloning a swarm. Merging two swarms allows computing functions over the secrets of both swarms together.

Our contribution. Information-theoretically secure schemes for sharing and modifying a secret among a dynamic swarm of computing devices are presented. The schemes support basic swarm operations including elegant proactive *join* and *leave* operations, removing, in particular, the need for a safe *regain consistency* operation [10]. We present an information-theoretically secure implementation of a (strongly) oblivious universal Turing machine, that supports private processing of an unbounded number of inputs.

Paper organization. Definitions and notation appear in the next section. Procedures for supporting (a) secret distribution, (b) *join* and *leave* of members, (c) *merge* of swarm secrets, and (d) secret threshold *increase* and (a new technique for)

secret threshold *decrease* are detailed in Section III. Section IV presents an efficient private way to implement a (strongly oblivious and universal) Turing Machine based on the procedures presented in Section III. The implementation uses only one round of communication for each Turing Machine transition.

II. Definitions and Notation

Let $\mathcal{U}$ denote the set of all possible participants in a dynamic secret sharing scheme. Let $F = f_1, \ldots, f_{|F|}$ denote a finite field such that $|F| \geq |\mathcal{U}|$. Each participant, $u \in \mathcal{U}$ is identified by an element f_u in F. When the context is clear we sometimes identify the element with the participant and write u instead of f_u.

The following two definitions of a secret sharing scheme are due to Shamir [17].

Definition 1: An (n, t) secret-sharing scheme allows a dealer to distribute a secret $s \in F$ among n players, such that any set of at most t players obtains no information on s from their joint shares, and any set of at least $t + 1$ players can determine s uniquely from their shares.

Definition 2: To t-privately share a secret $s \in F$, the dealer chooses t random field elements $a_1, \ldots, a_t$ and defines a polynomial $P(x)$ of degree t by $P(x) = a_t x^t + \ldots a_1 x + s$. The dealer sends the value $P(f_i)$ to the i-th player, $1 \leq i \leq n$. Any $t + 1$ players can compute $P(x)$ by polynomial interpolation and thus obtain the secret s.

In our context, a swarm can be formally defined as follows.

Definition 3: At time T, a *swarm* is a set SW, such that $SW \subseteq \mathcal{U}$. SW is characterized by the tuple $(k, t, s_1, s_2, \ldots, s_k)$, where k is the number of secrets that members of SW share, $s_1, \ldots, s_k$ are the secrets, and they are all shared according to an $(|SW|, t)$-secret sharing scheme, where $|SW|$ is the size of the swarm.

An important difference between a swarm and a traditional set of players that share a secret is the dynamic nature of participation, secret sharing and computation in a swarm. The next two definitions describe changes in membership and secret sharing thresholds.

Definition 4: The following operations change membership in a swarm:

- *Join(SW, u, t),* in which a new player u joins the swarm SW and obtains a share of each of the swarm's secrets $s_1, \ldots, s_k$ that are all shared by an $(|SW|, t)$ secret sharing schemes;
- *Leave(SW, u),* in which a swarm member u leaves the swarm SW, possibly retaining its shares of the swarm's secrets;
- *Merge(SW_1, SW_2),* in which two swarms merge into a single swarm, redistributing their secrets so that all participants in the merged swarm obtain a share of all secrets of the original two swarms;
- *Clone(SW_1, SW_2),* which copies all information from SW_1 to SW_2, while ensuring that each swarm shares its secrets independently of the other swarm; and
- *Split($SW, u_1, \ldots, u_m$),* which divides the swarm SW into two different swarms, the first with members $u_1, \ldots, u_m$ and the second with all other members of SW. The participants in both new swarms retain their shares of the original swarm's secrets.

Definition 5: Increase(SW, t, t^*) and *Decrease(SW, t, t^*)* change the secret sharing parameters of a swarm. The swarm changes from a $(|SW|, t)$-secret sharing scheme, to a $(|SW|, t^*)$-secret sharing scheme; $t^* \geq t$ for *Increase* and $t^* < t$ for *Decrease*.

Since a swarm may exist indefinitely, it is certainly possible that an adversary might control different subsets of members over time. We are thus interested in refreshing swarm secrets periodically. In other words, we would like to have a *proactive* secret sharing scheme [15], [13].

Definition 6: Refresh(SW) updates the shares of every secret in the swarm. The secret remains unchanged while any set of at most t new shares reveal no information about the secret or other shares (old or new).

The main objective of swarm members is to compute certain functions in a distributed and private manner. Definition 7 determines the input and output of such private distributed computation in a swarm setting.

Definition 7: Compute($SW, f, s_{i_1}, \ldots, s_{i_m}, G, O$) causes the participants of swarm SW to perform the computation of function f on inputs $s_{i_1}, \ldots, s_{i_m}, G$. $s_{i_1}, \ldots, s_{i_m}$ are m of the shared swarm secrets, G is a *global* input that all swarm members have received in the last time interval before the *Compute* operation is invoked and O is the output, which is shared among swarm members as an additional secret.

We complete this section by defining the adversarial model we are considering.

Definition 8: An adversary of a swarm SW is a computationally unbounded entity that may control up to t members of the swarm between two successive invocations of the *refresh* or *leave* operations. The adversary may also control other players, former members of SW. The adversary's only information on the swarm is the information gathered by the corrupted members (i.e., the adversary cannot eavesdrop on the communication links between uncorrupted members). The adversary is *semi-honest* (also called "honest but curious") in that it tries to deduce information on the swarm secrets, but executes the algorithms and protocols in the prescribed manner.

III. Dynamic Participation in Secret Sharing

In this section we provide an implementation of the basic swarm operations. Each swarm is initiated by a "sharing" phase in which a trusted party, called the Dealer, distributes k secrets among swarm members. The Dealer knows the swarm members, the secrets $s_1, \ldots, s_k$ and the privacy threshold t. We assume that the links between the Dealer and every other player are private.

Each secret s_j is shared by a bivariate polynomial $P(x, y)$ of degree t in each variable over field F. $P(x, y)$ is a random polynomial with the sole restriction that $P(0, 0) = s_j$.

Each such polynomial is chosen independently of all other polynomials. In our case, the main advantage of using such a polynomial over standard Shamir secret sharing with a single polynomial is that it will make the Join operation simple and efficient. A participant u in the swarm SW can compute two degree-t single variable polynomials. These polynomials are $P_u(y) \stackrel{\triangle}{=} P(u,y)$ and $P^*_u(x) \stackrel{\triangle}{=} P(x,u)$. Each such polynomial is of degree t due to the definition of $P(x,y)$ and u receives the value of each polynomial at $t+1$ points during the $Deal$ algorithm, or if u joins the swarm later, during the Join operation. Hence, u can compute the polynomials (and any point on it) using Lagrange interpolation.

For readability, each of the following algorithms for *Deal*, *Join*, *Join*, *Increase* and *Decrease* is described for a single secret. It is easy to generalize each algorithm for k secrets by executing the sequence of instructions k times (independently for each execution).

Algorithm 1 $Deal(SW, s, t)$

The dealer chooses $P(x,y)$, a random bivariate polynomial of degree t in each variable over field F such that $P(0,0) = s$.
for all $u \in SW$ **do**
Let $a_1, \ldots, a_{t+1} \in F$ be distinct field elements.
The dealer sends $P_u(a_1), \ldots, P_u(a_{t+1})$ and $P^*_u(a_1), \ldots, P^*_u(a_{t+1})$ to u.

A. Swarm Membership

The following algorithms implement operations that change swarm membership. Each algorithm affects the secret sharing parameters of a swarm. For example, a $(|SW|, t)$ secret sharing scheme becomes a $(|SW|+1, t)$ secret sharing scheme when a new member joins the swarm.

Algorithm 2 $Join(SW, u, t)$

Let $u_1, \ldots, u_{t+1} \in SW$ be $t+1$ swarm members.
Let $a_1, \ldots, a_{t+1} \in F$ be $t+1$ distinct field elements.
Let $P(x,y)$ be the polynomial used to share s.
all $u_i, i = 1$ to $t+1$ **do**
u_i computes $P_{u_i}(u)$ and $P^*_{u_i}(u)$.
u_i sends $P_{u_i}(u)$ and $P^*_{u_i}(u)$ to u.

When a swarm participant leaves the swarm, the number of swarm members decreases by one, but the threshold actually remains the same. The reason is that the swarm member that left may be itself corrupted by the adversary while being outside the swarm. Thus, the Leave operation re-shares swarm secrets, ensuring that the shares that the ex-member holds are now meaningless (that is, independent of swarm secrets).

Remaining members (in fact, $t+1$ of them) re-share secrets by distributing shares of random t-degree polynomials $P_1(x,y), \ldots, P_{t+1}(x,y)$ with a free coefficient that equals zero. The polynomial $Q(x,y) = P(x,y) + \Sigma_{i=1}^{t+1} P_i(x,y)$ is a bivariate polynomial of degree t such that $Q(0,0) = P(0,0)$. $Q(x,y)$ also has the important property that for any $(i,j) \neq (0,0)$ the value of $Q(i,j)$ is distributed independently of the polynomial $P(x,y)$. Thus, the shares of the ex-member do not give any information on $Q(x,y)$.

To summarize, after a member leaves a swarm and secrets are re-shared, the former $(|SW|, t)$ secret sharing scheme becomes a $(|SW|-1, t)$ secret sharing scheme.

Algorithm 3 $Leave(SW, u)$

Let $u_1, \ldots, u_{t+1} \in SW$ be $t+1$ swarm members.
Let $a_1, \ldots, a_{t+1} \in F$ be $t+1$ distinct field elements.
Let $P(x,y)$ be the polynomial used to share s_j.
all $u_i, i = 1$ to $t+1$ **do**
u_i chooses a random polynomial $P_i(x,y)$ such that $P_i(0,0) = 0$.
for all $u \in SW$ **do**
u_i sends $P_i(a_1, u)$,...,$P_i(a_{t+1}, u)$ and $P_i(u, a_1)$,...,$P_i(u, a_{t+1})$ to u.
all $u \in SW$ **do**
for $i = 1$ to $t+1$ **do**
u sets $Q(a_i, u) \longleftarrow P(a_i, u) + \sum_{j=1}^{t+1} P_j(a_i, u)$.
u sets $Q(u, a_i) \longleftarrow P(u, a_i) + \sum_{j=1}^{t+1} P_j(u, a_i)$.

When two swarms merge all their secrets are shared among the members of both original swarms. We denote the secrets of the first swarm by $s_1, \ldots, s_k$ and the secrets of the second swarm by $s_{k+1}, \ldots, s_m$. We assume that the first swarm shared secrets according to a $(|SW_1|, t)$ secret sharing scheme and that the second shares secrets according to a $(|SW_2|, t')$ secret sharing scheme, where $t \geq t'$. After the merge, the larger threshold is used in the new swarm. That is, the combined swarm shares secrets in a $(|SW_1| + |SW_2|, t)$ secret sharing scheme.

Algorithm 4 $Merge(SW_1, SW_2)$

for all $u \in SW_1$ **do**
Join(SW_2, u, t).
u retains all previous shares in SW_1.
for all $v \in SW_2$ **do**
Join(SW_1, v, t).
v retains all previous shares in SW_2.

The final membership operation is when two swarms split. When the size of both new swarms is larger than t, there is no need to change any of the shares or secrets of swarm members. However, if one of the swarms has fewer than t members, the original swarm must adjust the threshold, so that the new swarm can reconstruct its secrets. The next subsection shows how to change the threshold.

B. Changing the Threshold and Refreshing Shares

Swarm members may decide to increase or reduce the threshold of their secret sharing schemes. Increasing the

threshold is vital when the adversary may corrupt more players than the original threshold. Reducing the threshold may be necessary to avoid a situation in which too many members leave and the number of swarm members drops below the threshold for secret reconstruction. Decreasing the threshold may also be desirable for improved performance, when the adversary is weaker than previously thought.

Increasing the threshold is carried out by simply adding random polynomials of a higher degree, with free coefficient equal to zero.

Algorithm 5 $Increase(SW, t, t^*)$

Let $u_0, \ldots, u_{t^*} \in SW$ be $t^* + 1$ swarm members.
Let $P(x, y)$ be the polynomial used to share s.
all $u_i, i = 0$ to t^* **do**
  u_i chooses a random polynomial $P_i(x, y)$ of degree t^* in each variable with $P_i(0, 0) = 0$.
  for all $u \in SW$ **do**
    u_i sends to u: $P_i(a_0, u), \ldots, P_i(a_{t^*}, u)$ and $P_i(u, a_0), \ldots, P_i(u, a_{t^*})$.
all $u \in SW$ **do**
  for $i = 0$ to t^* **do**
    u sets $P(a_i, u) \longleftarrow P(a_i, u) + \Sigma_{i=0}^{t^*} P_i(a_i, u)$.
    u sets $P(a_i, u) \longleftarrow P(u, a_i) + \Sigma_{i=0}^{t^*} P_i(u, a_i)$.

Swarm members reduce the degree of $P(x, y)$ by subtracting high degree monomials from it. One way to achieve that is the distributed degree-reduction method of [2]. We propose a slightly different approach.

A set of $t+1$ members $u_1, \ldots, u_{t+1}$ cooperate in computing a random bivariate polynomial $Q(x, y)$ of degree t. Q is a sum of the original secret sharing polynomial P and $t + 1$ random polynomials $Q^1, \ldots, Q^{t+1}$ each of degree t^* in both x and y. Consider a monomial of Q: $q_{\ell,m} x^i y^j$. If either $\ell > t^*$ or $m > t^*$ then $q_{\ell,m} = p_{\ell,m}$, where $p_{\ell,m}$ is the coefficient of $x^\ell y^m$ in P. However, if $\ell \leq t^*$ and $m \leq t*$ then $q_{\ell,m}$ is a sum of the coefficients of $x^\ell y^m$ in $P, Q^1, Q^2, \ldots, Q^{t+1}$.

IV. Private Distributed Computation in a Swarm

Swarm members refresh all their shares periodically. A swarm decides on the time gap between refreshing periods based on estimates of how long it takes the adversary to corrupt new swarm members. The method we propose for refreshing secrets is a simplification of [15] (which considered actively malicious adversaries). Periodic refreshing is identical to the refresh phase of the Leave operation in Algorithm 3. Every member chooses a random t-degree polynomial with the free coefficient equal to 0. Each member distributes shares of its polynomial to all members. The new share is a sum of the previous share and all the shares that were received by a member.

Refreshing secrets is effective against an adversary that controls parties for a short time, reads their shares and then loses control of the party. If an adversary is likely to retain control of a member once it is compromised then additional measures must be adopted, such as adding fresh members that are not compromised to the swarm, while forcing older members to leave.

Algorithm 6 $Decrease(SW, t, t^*)$

1: Let $u_1, \ldots, u_{t+1} \in SW$ be $t + 1$ swarm members.
2: Let $P(x, y)$ be the polynomial used to share s.
3: **all** $u_i, i = 1$ to $t + 1$ **do**
4: u_i chooses a random polynomial $Q^i(x, y)$ of degree t^* .
5: **for** $j = 1$ to $t + 1$ **do**
6: u_i sends $Q^i_{u_j}(x)$ and $Q^{i*}_{u_j}(y)$ to u_j.
7: u_i sends $P_{u_i}(x) + \sum_{j=1}^{t+1} Q^j_{u_i}(x)$ and $P^*_{u_i}(y) + \sum_{j=1}^{t+1} Q^{j*}_{u_i}(y)$ to u_1.
8: u_1 interpolates and computes a polynomial $Q(x, y) = P(x, y) + \sum_{j=1}^{t+1} Q^j(x, y)$.
9: **for all** $u \in SW$ **do**
10: u_1 sends to u every coefficient $q_{\ell,m}$ of Q such that either $\ell > t^*$ or $m > t^*$.
11: For each such coefficient $q_{\ell,m}$ of Q, u computes $P_u(x) \leftarrow P_u(x) - q_{\ell,m} x^\ell u^m$ and $P^*_u(y) \leftarrow P^*_u(y) - q_{\ell,m} u^\ell y^m$.

Thus far, we have considered how swarm members store secrets, share them and manage membership in the swarm. In this section, we discuss how a swarm performs computation based on its shared secrets and input that swarm members receive throughout their operation.

A. Requirements

As discussed in Section III, a swarm has two distinct phases, an initialization phase and an operational phase. During initialization, a dealer shares multiple secrets among swarm members. These secrets are part of the input for subsequent computation. The operational phase, which may continue indefinitely, is characterized by swarm members receiving additional input from external sources (such as sensors) and performing computation on this input and their secrets. The output of any computation can be revealed to a subset of swarm members, or stored as input for future computation.

There are several requirements on swarm computation. The first requirement is that values computed in a distributed manner by a swarm be always *correct*, in the sense that a large enough subset of swarm members can obtain together the same values that a single agent would have computed if it had all the necessary program descriptions, input and intermediate values.

The second requirement is that the computation may continue forever, based on input that swarm participants receive. Such input may prompt swarm members to share it with other members, compute some function or even change the programs that dictate the computation.

Informally, the third requirement is that any computation carried out by a swarm remain *t-private*. In other words, a

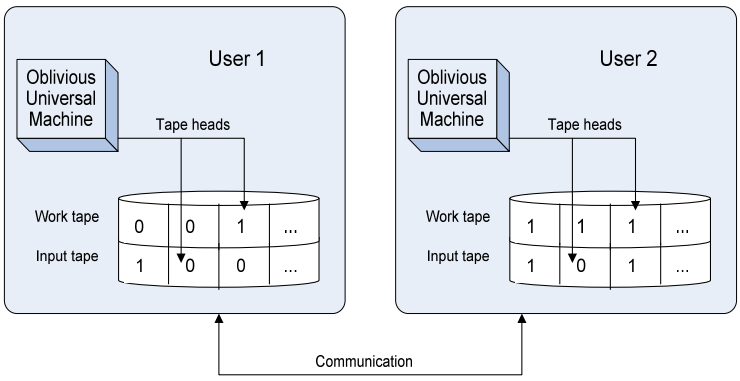

Fig. 1. Architecture of two swarm members

small subset of swarm members ($\leq t$) cannot obtain information on the secrets of other members or on output computed by other members. Swarm members are allowed to know only the space required by the Turing machine and nothing else (for example, what is being computed by the TM and its time complexity must not be revealed).

More formally, the requirements above can be readily derived from the definition of secure multi-party computation [12] for when the adversary is semi-honest (that is, honest but curious) and privacy is maintained in an information-theoretic sense.

B. Architecture

We now proceed to design a system that fulfills the requirements of Section IV-A. Figure 1 shows the architecture of a swarm member's system. Computation is carried out by an oblivious, universal Turing machine. The TM has two cyclic tapes: a work tape and an input tape on which the system sensors write input that they gather throughout the operational phase. Each swarm member has a (private) communication link with every other member.

We assume that the Turing machine is *universal*, and thus the actual computation it performs is encoded on the work tape. We further assume that the Turing machine is *oblivious*; in other words, the position of each tape head is a function only of the number of steps executed by the Turing machine, and not of the input. Pippenger and Fischer showed that every language L that is accepted by a Turing Machine with running time $T(n)$ is accepted by an oblivious Turing machine with running time $O(T(n)\log T(n))$ [16]. However, the method of [16] reveals information about the number of steps that the TM has taken so far; the movement of tape heads may change as a function of the number of steps the TM has performed. Since we would like to also hide information on the number of steps that the TM takes, we need a *strongly* oblivious Turing Machine. We define this notion and give a construction in Section IV-C.

The work tape is used to store the description of an *initial* Turing machine TM_I and its input, which the universal TM executes. During the initialization phase, the dealer secretly shares among swarm members both TM_I and its input, so a subset of at most t members does not have any information on TM_I or its input. The work tape is cyclic and swarm members can use it to hide the number of steps taken by TM_I as follows. The dealer shares a symbol for each cell in the cyclic tape. Unused tape cells have a special symbol that signifies that they are unused. The dealer begins writing the actual data (TM_I and input) at a random cell on the work tape. Swarm members may completely refresh the work tape by agreeing on a cyclic shift and moving all cells (and tape head) by that shift. For each cell i, swarm members use a $|SW|, t$ secret sharing scheme to share the value 0, independently of all other cells. Each member adds its share to the value in the i-th cell.

Thus, after a refresh, any subset of less than t tapes is distributed independently of its distribution prior to this phase. If an adversary does not control any member during such a refresh phase, then it does not have any direct information on the relation of the the work tapes before a refresh phase and the work tapes after a refresh phase. In particular, the adversary cannot directly determine how many computation steps were executed by swarm members since the last time that the adversary controlled one of them. Note that this does not preclude an adversary from learning information by indirect means, for example, by estimating the number of steps based on elapsed time rather than by direct observation of the computation.

The input tape is also cyclic and includes all input collected by swarm sensors during its operation. We assume that all swarm members receive the *same input* and store it in a shared manner on their input tapes as follows. The swarm members share independently for each cell of the tape the value 0 in a bivariate $(|SW|, t)$-secret sharing scheme as described in Section III. When all swarm members receive the same input, each one adds it to the next cell in the input tape, thus sharing the input in $(|SW|, t)$-secret sharing. Swarm members than re-share a zero for the next cell. This shared input can then be used in the computation on the work tape.

Receiving input during the operational phase enables the dealer to change TM_I on the fly. Since TM_I is encoded on the work tape, changing the "program" being computed is the

same as changing input for that program.

Each computational step of the universal Turing machine consists of the following. Let st be the current state of the TM finite automaton, and let γ be the combination of symbols on the input and work tape, that the tape heads point to. The TM reads γ from the tape cells, computes the transition function $\mu(st, \gamma) = (st', \gamma')$, sets the automaton state to st', writes γ' back to the tapes (in the same cells) and moves the tape heads. The movement of tape heads to a new location is determined uniquely because the TM is oblivious and moves the tape heads through the same sequence of cells, regardless of input.

Swarm members run the universal Turing machine in a distributed manner. Each member has its own Turing machine. Any subset of more than t members can reconstruct the state and tape contents of the original, universal Turing machine, while any subset of at most t members does not obtain any information about it. The members secretly share the state of the original Turing machine and the contents of each cell. However, their tape heads always point to the same cells. (see Figure 1). We now give a formal definition of a strongly oblivious TM.

C. A Strongly Oblivious Turing Machine

Definition 9: A *strongly* oblivious Turing Machine is a Turing Machine in which the movement of tape heads is a function only of the cell indices that the heads point to.

Not every oblivious TM is *strongly* oblivious, since the movement of tape heads may be a function of time, not only of space.

We now construct a strongly oblivious TM. Given any Turing Machine M we simulate it with a strongly oblivious TM OM. Assume that on input of length n the time complexity of M is $T(n)$ and its space complexity is $S(n)$. Assume further that $S(n)$ can be generated by a Turing Machine in $T'(n)$ time. Then, on input of length n, the time complexity of OM is $T'(n)+T(n)\cdot(S(n)+1)$ and its space complexity is $S(n)$.

Let M have a set of states, ST, a set of input symbols Γ and a transition function $\mu : ST \times \Gamma \longrightarrow ST \times \Gamma \times \{+, -\}$. To describe a step of M, let st be the current state, let the tape head point to cell i, let γ be the input symbol at cell i and let $\mu(st, \gamma) = \langle \overline{st}, \overline{\gamma}, dir \rangle$, where $dir \in \{+, -\}$. Then, M changes its state to $\overline{st}$, writes $\overline{\gamma}$ to cell i and moves the tape head. If $dir = +$ then the tape head is moved to $i + 1$, otherwise it is moved to $i - 1$.

OM description: OM simulates M over a cyclic tape of size $S(n)$ cells. For each step of M, the tape head of OM makes $S(n) + 1$ steps, shifting every cell to the adjacent cell. The tape head of OM always moves from cell i to cell $i + 1 \bmod S(n)$. The state of OM stores the state of M and its tape stores both the contents of M's tape and information on the current position of M's tape head.

In more detail, let OM have state set ST_O, a set of input symbols Γ_O and a transition function $\mu_O : ST_O \times \Gamma_O \longrightarrow ST_O \times \Gamma_O$. Define $\Gamma_O = \Gamma \times \{Yes, No\}$, where a symbol $\langle \gamma, Head \rangle$ includes γ which is an input symbol in M and $Head \in \{Yes, No\}$, which determines whether the head of M points to this cell in the simulation ($head = Yes$) or not. Define $ST_O = ST \times \Gamma \times \{0, 1, 2\}$, where a state $\langle st, \gamma_p, Pos \rangle$ is a tuple of st, a state in M, the symbol of M in the previous cell, γ_p and Pos, which is the number of cells left until the stored position of M's tape head must be changed. If $Pos = 0$ then $Head$ remains as it was in the same cell.

Let the state of OM be $st_O = \langle st, \gamma_p, Pos \rangle$, let the current OM cell store symbol $\gamma_O = \langle \gamma_c, Head \rangle$ and let $\mu(st, \gamma_c) = \langle \overline{st}, \overline{\gamma_c}, dir \rangle$. If $Head = No$ and $Pos = 0$ then:

$$\mu_O(st_O, \gamma_O) = \langle \langle \overline{st}, \overline{\gamma_c}, 0 \rangle, \langle \gamma_p, No \rangle \rangle.$$

If $Head = Yes$ and $dir = -$ then the position of the head is stored in the current cell (since the whole tape is being shifted by one cell, the current cell is analogous to moving in the "$-$" direction in M):

$$\mu_O(st_O, \gamma_O) = \langle \langle \overline{st}, \overline{\gamma_c}, 0 \rangle, \langle \gamma_p, Yes \rangle \rangle.$$

If $Head = Yes$ and $dir = +$ then the state of OM stores the information that the tape head should move two cells:

$$\mu_O(st_O, \gamma_O) = \langle \langle \overline{st}, \overline{\gamma_c}, 2 \rangle, \langle \gamma_p, No \rangle \rangle.$$

If $Head = No$ and $Pos = 2$ then the tape head is not stored in this cell, but in the next, so Pos is changed to 1:

$$\mu_O(st_O, \gamma_O) = \langle \langle \overline{st}, \overline{\gamma_c}, 1 \rangle, \langle \gamma_p, No \rangle \rangle.$$

If $Head = No$ and $Pos = 1$ then the tape head is stored in this cell and Pos is changed to 0:

$$\mu_O(st_O, \gamma_O) = \langle \langle \overline{st}, \overline{\gamma_c}, 0 \rangle, \langle \gamma_p, Yes \rangle \rangle.$$

D. Secretly Sharing Turing Machine Transitions

We now show how n players who jointly and secretly share the state of a Turing Machine, can continue to share an updated state after a single Turing Machine transition. Let the *state* of a Turing Machine be a combination of the state of its finite automaton, the contents of its tapes and the position of tape heads.

Theorem 1 constructs an (n, t) secret sharing scheme for a strongly oblivious Turing machine. This is a scheme to share the state of a strongly oblivious Turing machine among n parties so that any set of $t + 1$ players can reconstruct the current state and maintain a correct shared state after each transition of the TM. Each TM transition is carried out by the parties computing the transition function in a distributed manner on their shares. The result of this computation is shares of the new finite automaton state and shares of a new symbol for the current tape cell. Note that since our implementation of a strongly oblivious TM simulates adds to each cell either a Yes or a No, we consider each symbol on a tape to be a pair $\gamma = \langle c, D \rangle$, where c is a symbol for the original TM being simulated and $D \in \{Yes, No\}$.

Theorem 1: Let TM be a strongly oblivious Turing machine with a set of automaton states ST, a set of input symbols Γ and a transition function $\mu : ST \times \Gamma \longrightarrow ST \times \Gamma$. Let the required space of TM be m cells on its tape. Then, there exists an (n, t) secret sharing scheme of the machine's state, $n > 2t$,

such that the players can compute shares of the new state using $2nt \log n(|ST|+|\Gamma|)$ bits of communication. The size of each share is $O(|ST| \cdot |\Gamma| \cdot \log(|ST| \cdot |\Gamma|) + 2t(|ST| + m\,|\Gamma|) \log n)$. Computing each state transition requires $O(t^2 S[|ST| \cdot |\Gamma| + n(|ST| + |\Gamma|)])$ time by each player.

Proof: Let st be the secret, initial state of the automaton, let the initial contents of the tapes be $\gamma_1, \ldots, \gamma_m$ and let the initial position of the tape head be set as defined by a strongly oblivious Turing machine. Let F be a field, $|F| > n$, and let $f_1, \ldots, f_n \neq 0$ be n distinct field elements.

The secret state st is associated with the st-th unit vector of length $|ST|$. The value γ_k of cell k is associated with the γ_k-th unit vector of length $|\Gamma|$. The dealer associates st and γ_k with unit vectors by selecting $|ST| + m\,|\Gamma|$ bivariate polynomials of degree t in each variable over F:

$$\begin{array}{l} P_1(x,y), \ldots, P_{|ST|}(x,y), \\ Q_{1,1}(x,y), \ldots, Q_{1,|\Gamma|}(x,y), \\ \ldots, \\ Q_{m,1}(x,y), \ldots, Q_{m,|\Gamma|}(x,y). \end{array}$$

Each of these polynomials is random over F, except for the free coefficient. $P_{st}(0,0) = 1$, while the free coefficient of P_i for any $i \neq st$ is 0. $Q_{k,\gamma_k}(0,0) = 1$ for every $k = 1, \ldots, m$, while $Q_{k,\gamma_\lambda} = 0$, for any $\lambda \neq k$.

We first show how to compute a transition without considering privacy and then describe how to privately distribute that computation. Intuitively, we multiply each "state" polynomial with every "symbol" polynomial, obtaining $|ST| \cdot |\Gamma|$ product polynomials. The free coefficients of all product polynomials are 0, except for the product of the correct state and correct symbol. By carefully summing $|ST| \cdot |\Gamma|$ polynomials into $|ST|$ polynomials and reducing their degree we obtain correct new state polynomials. An analogous summation for Γ input symbols yields new symbol polynomials.

In more detail, let the transition be $\mu(st, \gamma_k) = \langle \overline{st}, \overline{\gamma_k} \rangle$. We compute $\overline{P}_1(x,y), \ldots, \overline{P}_{|ST|}(x,y)$ by

$$\overline{P}_\ell(x,y) = \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle \ell, \gamma' \rangle}} P_i(x,y) \cdot Q_{k,\gamma_\lambda}(x,y), \qquad (1)$$

where γ' is any possible tape symbol.

For every $\ell = 1, \ldots, |ST|$, the polynomial $\overline{P}_\ell(x,y)$ is of degree $2t$ in each variable. Furthermore, $\overline{P}_{\overline{st}}(0,0) = 1$, while $\overline{P}_\ell(0,0) = 0$ for any $\ell \neq \overline{st}$. The next step is to reduce every polynomial $\overline{P}_\ell(x,y)$, $\ell = 1, \ldots, |ST|$, to degree t in each variable by removing every monomial with powers of x or y greater than t. By setting $P_1(x,y), \ldots, P_{|ST|}(x,y)$ to be the t-degree reduced polynomials, we complete the computation of a new state (represented by $P_1(x,), \ldots, P_{|ST|}(x,y)$).

Computing the symbol $\overline{\gamma_k}$ to store in the k-th cell is similar to computing the state. Compute $\overline{Q}_{k,1}(x,y), \ldots, \overline{Q}_{k,|\Gamma|}(x,y)$ by

$$\overline{Q}_{k,\ell}(x,y) = \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle st', \gamma_\ell \rangle}} P_i(x,y) \cdot Q_{k,\gamma_\lambda}(x,y), \qquad (2)$$

where st' is any possible state.

$\overline{Q}_{k,\ell}(x,y)$ is of degree $2t$ in each variable, for every $\ell, 1 \leq \ell \leq |\Gamma|$ and $\overline{Q}_{k,\overline{\gamma_k}}(0,0) = 1$, while $\overline{Q}_{k,\ell}(0,0) = 0$ for any $\ell \neq \overline{\gamma_k}$. By setting $Q_{k,\ell}(x,y)$ to be the t degree reduced polynomial derived from $\overline{Q}_{k,\ell}(x,y)$, $1 \leq \ell \leq |\Gamma|$, we complete the computation of a tape symbol.

We now describe how to privately distribute the computation of a transition described above. In the dealing phase, the dealer provides the following to player number j, $j = 1, \ldots, n$:

- The transition function μ.
- The single variable polynomials $P_{1,f_j}(x), \ldots, P_{|ST|,f_j}(x)$ and $P^*_{1,f_j}(y), \ldots, P^*_{|ST|,f_j}(y)$, where for $i = 1, \ldots, |ST|$, $P_{i,f_j}(x) = P_i(x, f_j)$ and $P^*_{i,f_j}(y) = P_i(f_j, y)$.
- m tape cells, containing in cell k, $k = 1, \ldots, m$, the single variable polynomials $Q_{k,1,f_j}(x), \ldots, Q_{k,|\Gamma|,f_j}(x)$ and $Q^*_{k,1,f_j}(y), \ldots, Q^*_{k,|\Gamma|,f_j}(y)$, where $Q_{k,i,f_j}(x) = Q_{k,i}(x, f_j)$ and $Q^*_{k,i,f_j}(y) = Q_{k,i}(f_j, y)$.

μ is a table of $|ST| \cdot |\Gamma|$ entries, with each entry of size $\log(|ST| \cdot |\Gamma|)$. The polynomials $P_{i,f_j}(x)$ and $P^*_{i,f_j}(x)$ for $i = 1, \ldots, |ST|$ are of total size $2t\,|ST| \cdot \log n$. Each of the m tape cells contains the 2Γ polynomials, which require $O(2t\,|\Gamma| \log n)$ storage per cell. Therefore, the total size of a share is $O(|ST| \cdot |\Gamma| \cdot \log(|ST| \cdot |\Gamma|) + 2t(|ST| + m\,|\Gamma|) \log n)$.

Each player sets the tape heads to the same position. The state of the TM is shared, i.e., player j holds the polynomials $P_{1,f_j}(x), \ldots, P_{|ST|,f_j}(x)$ and $P^*_{1,f_j}(y), \ldots, P^*_{|ST|,f_j}(y)$ as its share of the state. The input symbol in cell k is also shared. Player j holds $Q_{k,1,f_j}(x), \ldots, Q_{k,|\Gamma|,f_j}(x)$ and $Q^*_{k,1,f_j}(y), \ldots, Q^*_{k,|\Gamma|,f_j}(y)$ as its share of the symbol.

Each player computes temporary variables $\overline{P}_{\ell,f_j}(x)$ and $\overline{P}^*_{\ell,f_j}(x)$, for $\ell = 1, \ldots, |ST|$, as follows:

$$\overline{P}_{\ell,f_j}(x) \longleftarrow \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle \ell, \gamma' \rangle}} P_{i,f_j}(x) \cdot Q_{k,\gamma_\lambda,f_j}(x) \qquad (3)$$

and

$$\overline{P}^*_{\ell,f_j}(y) \longleftarrow \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle \ell, \gamma' \rangle}} P^*_{i,f_j}(y) \cdot Q^*_{k,\gamma_\lambda,f_j}(y), \qquad (4)$$

where γ' is any possible symbol on the tape. Thus, $\overline{P}_{\ell,f_j}(x) = \overline{P}_\ell(x, f_j)$ and $\overline{P}^*_{\ell,f_j}(y) = \overline{P}_\ell(f_j, y)$, where $\overline{P}_\ell(x,y)$ is defined in Equation 1.

The next step is to correctly compute an output symbol. The player computes temporary variables $\overline{Q}_{k,\ell,f_j}(x)$ and $\overline{Q}^*_{k,\ell,f_j}(y)$, for $\lambda = 1, \ldots, |\Gamma|$, as follows:

$$\overline{Q}_{k,\ell,f_j}(x) = \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle st', \gamma_\ell \rangle}} P_{i,f_j}(x) \cdot Q_{k,\gamma_\lambda,f_j}(x)$$

and

$$\overline{Q}^*_{k,\ell,f_j}(y) = \sum_{\substack{i,\lambda \ s.t. \\ \mu(i,\gamma_\lambda) = \langle st', \gamma_\ell \rangle}} P^*_{i,f_j}(y) \cdot Q_{k,\gamma_\lambda,f_j}(y)$$

where st' is any possible state. $\overline{Q}_{k,\ell,f_j}(x) = \overline{Q}_{k,\ell}(x, f_j)$

and $\overline{Q}^{*}_{k,\ell,f_j}(y) = \overline{Q}_{k,\ell}(f_j, y)$, where $\overline{Q}_{k,\ell}(x, y)$ is defined in Equation 2.

The only problem left is that the polynomials representing the state and the output symbol are all of degree $2t$. The n players now utilize the degree reduction protocol described in Algorithm 6 to reduce their single variable polynomials to degree t in each variable. There are n players, $2|ST| + 2|\Gamma|$ polynomials, and the size of each share is $O(t \log n)$ which together yield communication complexity $2nt \log n(|ST|+|\Gamma|)$.

Each player now completes the state transition by storing the new shares of polynomials representing an input symbol in the current tape cell. ■

V. Conclusions

We have shown how a swarm can maintain security against an honest-but-curious adversary, while engaging in typical swarm activities over a long period of time. These activities include changing the membership of a swarm, changing the security levels of a swarm and performing distributed computation. Such computation is influenced by a swarm's state, external input that swarm sensors obtain and the "software" of a swarm. This software is itself kept secret and may change over time.

A great advantage in our context of using bivariate polynomials for our secret-sharing schemes is that joining a swarm is simple. Each existing member computes a value locally and sends it to the joining member. In contrast, in the single-polynomial case, each existing member would have to re-share its share among $t + 1$ other members before the new member can receive pieces of its share. The major disadvantage of using bivariate polynomials is that communication and storage for most operations, increases by a factor of t compared to the single-polynomial case, and local computation of Turing Machine transitions increases by a factor of t^2.

We have proposed to model swarm computation by a Turing Machine, instead of the more usual choice of circuits representing the function as in secure multi-party computation. Swarm computation is characterized by repeatedly receiving external input, computing a new state and storing it. In order to achieve the same functionality with circuits, it would be necessary to add storage to the circuit, enable external input and feedback the result of the circuit back into itself for another round of computation. Such modifications would essentially result in a similar model to the Turing Machine version we use in this paper.

A limitation of the TM version we propose is bounded space. On a practical level, this limitation is a more realistic model for real-world computation devices than an infinite tape. However, in case space constraints cannot be computed before the swarm begins its operational phase, then it is always possible for swarm members to increase their space on the fly (say, by doubling it) whenever required by the TM execution. Note that this increase in space does reveal some information to the adversary.

References

[1] Angluin, D., Aspnes, J., Diamad, Z., Fischer, M.J., and Peralta, R., "Computation in networks of passively mobile finite-state sensors," *Proc. of the 23^{rd} Annual ACM Symposium on Principles of Distributed Computing* (PODC), pp. 290-299, 2004.

[2] Ben-Or, M., Goldwasser, S., and Wigderson, A., "Completeness theorems for non-cryptographic fault-tolerant distributed computation" *Proc. of the 20th annual ACM symposium on Theory of computing* (STOC 1988), pp. 1-10, Chicago 1988.

[3] Chaum, D., Crépeau, C. and Damgård, I., "Multiparty unconditionally secure protocols(extended abstract)", *Proc. of the 20th annual ACM symposium on Theory of computing* (STOC 1988), pp. 11–19, 1988.

[4] Chatzigiannakis, I., Dolev, S., Fekete, S. P., Michail, O., and Spirakis, P., G., "Not All Fair Probabilistic Schedulers are Equivalent," *Proc. of the 2009 International Conference On Principles Of Distributed Systems*, (OPODIS), 2009.

[5] Dolev, S., Gilbert, S., Lahiani, L., Lynch, N., and Nolte, T., "Virtual Stationary Automata for Mobile Networks", *Proc. of the 2005 International Conference On Principles Of Distributed Systems*, (OPODIS), LNCS 3974, 2005. Also **invited paper** in *Forty-Third Annual Allerton Conference on Communication, Control, and Computing*.

[6] Dolev, S., Gilbert, S., Lynch, A. N., Schiller, E., Shvartsman, A., and Welch, J. "Virtual Mobile Nodes for Mobile Ad Hoc Networks", *18th annual Conference on Distributed Computing*, (DISC 2004), pp. 230-244, 2004.

[7] Dolev, S., Gilbert, S., Lynch, N. A., Shvartsman, A., Welch, J., "GeoQuorum: Implementing Atomic Memory in Ad Hoc Networks" *Distributed Computing*, special issue of selected paper from DISC 2003, Volume 18, Number 2, pp. 125-155, November 2005.

[8] Dolev, S., Gilbert, S., Schiller, E., Shvartsman, A., and Welch, J. "Autonomous Virtual Mobile Nodes" *Third ACM/SIGMOBILE Workshop on Foundations of Mobile Computing*, (DIALM/POMC), pp. 62-69, 2005. Brief announcement in *Proc. of the 17th International Conference on Parallelism in Algorithms and Architectures*, (SPAA 2005), pp. 215, 2005.

[9] Dolev, S., Lahiani, L., Lynch, N., and Nolte, T., "Self-Stabilizing Mobile Location Management and Message Routing", *Proc. of the 7th International Symposium on Self-Stabilizing Systems*, (SSS 2005), LNCS 3764, pp. 96-112, 2005.

[10] Dolev, S., Lahiani, L., Yung, M., "Secret Swarm Unit - Reactive k-Secret Sharing", *Proc. of the 8th International Conference on Cryptology*, LNCS 4859, (INDOCRYPT 2007), pp. 123-137, December 2007.

[11] Dolev, S., Schiller, M. E., and Welch, J., "Random Walk for Self-Stabilizing Group Communication in Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, 5(7) 893-905, 2006.

[12] Goldreich, O. "Foundations of Cryptography: Volume 2, Basic Applications", S Cambridge University Press, New York, NY, USA, ISBN 0521830842, 2004.

[13] Herzberg, A., Jarecki, S., Krawczyk, H. and Yung, M., "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage", *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, (CRYPTO '95), pp. 339–352, 1995.

[14] Kivelevich, E. and Gurfil, P., "UAV Flock Taxonomy and Mission Execution Performance", *Proc. of the 45th Israeli Conference on Aerospace Sciences*, 2005.

[15] Ostrovsky, R. and Yung, M. "How to withstand mobile virus attacks". *Proc. of the 10th Annual ACM Symp. on Principles of Distributed Computing*, (PODC 1991), pp. 51–59, 1991.

[16] Pippenger, N. and Fischer, M., "Relations Among Complexity Measures", *JACM*, volume 26, number 2, 361–381, 1979.

[17] Shamir, A., "How to Share a Secret", *CACM*, vol. 22, no. 11 (1979), pp. 612-613.

[18] Virtual Infrastructure Project http://groups.csail.mit.edu/tds/vi-project.

[19] Weiser, M., "The Computer for the 21th Century", *Scientific American*, September, 1991.

[20] Yao, A.C., "Protocols for Secure Computations (Extended Abstract)", *Twenty-third Annual IEEE Symposium on the Foundations of Computer Science (FOCS-82)*, pp. 160–164, 1982.

[21] Yao, A.C., "How to generate and exchange secrets", *Twenty-seventh Annual IEEE Symposium on the Foundations of Computer Science (FOCS-86)*, pp. 162–167, 1986.