Computing in Social Networks

Andrei Giurgiu[1], Rachid Guerraoui[1],
Kévin Huguenin[2], and Anne-Marie Kermarrec[3]

[1] EPFL
[2] Université de Rennes 1 / IRISA
[3] INRIA Rennes - Bretagne Atlantique

Abstract. This paper defines the problem of Scalable Secure Computing in a Social network: we call it the S^3 problem. In short, nodes, directly reflecting on associated users, need to compute a function $f : V \to U$ of their inputs in a set of constant size, in a *scalable* and *secure* way. Scalability means that the message and computational complexity of the distributed computation is at most $\mathcal{O}(\sqrt{n} \cdot \operatorname{polylog} n)$. Security encompasses (1) accuracy and (2) privacy: accuracy holds when the distance from the output to the ideal result is negligible with respect to the maximum distance between any two possible results; privacy is characterized by how the information disclosed by the computation helps faulty nodes infer inputs of non-faulty nodes.

We present AG-S3, a protocol that S^3-computes a class of aggregation functions, that is that can be expressed as a commutative monoid operation on U: $f(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$, assuming the number of faulty participants is at most $\sqrt{n}/\log^2 n$. Key to our protocol is a dedicated overlay structure that enables secret sharing and distributed verifications which leverage the social aspect of the network: nodes care about their reputation and do not want to be tagged as misbehaving.

1 Introduction

The past few years have witnessed an explosion of online social networks and the number of users of such networks is still growing regularly by the day, e.g. Facebook boasts by now more than 400 millions users. These networks constitute huge live platforms that are exploited in many ways, from conducting polls about political tendencies to gathering thousands of students around an evening drink. It is clearly appealing to perform large-scale general purpose computations on such platforms and one might be tempted to use a central authority for that, namely one provided by the company orchestrating the social network. Yet, this poses several privacy problems, besides scalability. For instance, there is no guarantee that Facebook will not make any commercial usage of the personal information of its users. In 2009, Facebook tried to change its privacy policy to impose new terms of use, granting the company a perpetual ownership of personal contents – even if the users decide to delete their account. The new policy was not adopted eventually, but highlighted the eagerness of such companies to use personal and sensitive information.

S. Dolev et al. (Eds.): SSS 2010, LNCS 6366, pp. 332–346, 2010.

We argue for a decentralized approach where the participants in the social network keep their own data and perform computations in a distributed fashion without any central authority. A natural question that arises then is what distributed computations can be performed in such a decentralized setting. Our primary contribution is to lay the ground for precisely expressing the question. We refer to the underlying problem as the S^3 problem: *Scalable Secure Computing in a Social network*. Whereas *scalability* characterizes the message and computational complexity of the computation, the *secure* aspect of S^3 encompasses accuracy and privacy. *Accuracy* refers to the robustness of the computation and aims at ensuring accurate results in the presence of dishonest participants. This is crucial in a distributed scheme where dishonest participants might, besides disrupting their own input, also disrupt any intermediary result for which they are responsible. The main challenge is to limit the amount of bias caused by dishonest participants. *Privacy* is characterized by the amount of information on the inputs disclosed to other nodes by the computation. Intuitively, achieving all three requirements seem impossible. Clearly, tolerating dishonest players and ensuring privacy calls for cryptographic primitives. Yet, cryptographic schemes, typically used for multi-party computations, involve too high a computation overhead and rely on higher mathematics and the intractability of certain computations [1,2,3]. Instead, we leverage users' concern for reputation using a information theoretical approach and alleviate the need for cryptographic primitives. A characteristic of the social network context is indeed that the nodes are in fact users who might not want to reveal their input, nor expose any misbehavior. This reputation concern determines the extent to which dishonest nodes act: up to the point that their misbehavior remains discrete enough not to be discovered.

Solving the S^3 problem is challenging, despite leveraging this reputation concern: to ensure privacy, an algorithm must ensure that even when all the nodes except one have the same inputs, the information obtained by the coalition of faulty nodes cannot know which non-faulty node had a different input. This requires the existence of two configurations of inputs that differ for two nodes, which with high probability lead to the same sequence of messages received by the faulty nodes. In turn, this boils down to *swapping* two nodes' inputs transparently (from the standpoint of the faulty nodes), which is challenging when the protocol needs to be also scalable and accurate. The scalability requirement (i.e., each node communicates with a limited number of nodes) makes it difficult to find a chain of messages that can be swapped transparently between two nodes in the system. The trade-off between privacy and accuracy can be illustrated by the following paradox: on the one hand verifying that nodes do not corrupt the messages they receive (without digital signature) requires the verifier to gather some information about what the verified node received; on the other hand the more the nodes know about the messages exchanged the more the privacy of the nodes is compromised.

Our contributions are twofold. Firstly, we define the Scalable Secure Computing problem in a Social network, namely the S^3 problem. Secondly, we present a distributed protocol, we call AG-S3 (i.e., S^3 for AGgregation), that solves the problem for a class of aggregation functions that derive from a monoid operation on U: $f(x_1, ..., x_n) = x_1 \oplus \cdots \oplus x_n$, under the assumption that the number of faulty nodes is upper-bounded by $\sqrt{n}/\log^2 n$. At the core of our protocol lie (1) a structured overlay where nodes are clustered into groups, (2) a secret sharing scheme that allows the nodes to obfuscate their inputs, and (3) a verification procedure which potentially tags the profiles of suspected nodes. Beyond these contributions, our paper can be viewed as a first step toward characterizing what can be computed in a large scale social network while accounting for the human nature of its users.

2 Problem

This section defines the problem of *Scalable Secure* Computing in a *Social network*: the S^3 problem. The problem involves a S^3 candidate, namely the function to be computed, and a set of nodes $\Pi = \{p_1, \ldots, p_n\}$.

2.1 Candidates

Definition 1 (S^3 candidate). *A S^3 candidate is a quadruple (f, V, U, d), where V is an arbitrary set, f is a function $f : V^* \to U$ such that $f(v_1, \ldots, v_n) = f(v_{\sigma(1)}, \ldots, v_{\sigma(n)})$ for any permutation σ of the inputs, and (U, d) is a metric space.*

Each node in Π has an input value in the set V, and a S^3 candidate maps the inputs of the nodes to a value in a metric space. The function f is assumed to be symmetric in the sense that the output depends on the multiset of inputs but not on their assignation to nodes. For example, a binary poll over Π can be modeled by the S^3 candidate $((v_1, v_2, \ldots, v_n) \mapsto v_1 + \cdots + v_n, \{-1, +1\}, \mathbb{Z}, (z_1, z_2) \mapsto |z_1 - z_2|)$. Consider also a component-wise addition on $U = \mathbb{Z}^d$, where V is the set of all vectors with exactly one nonzero component, which is either $+1$ or -1. The distance function is then just the L_1 (or Manhattan distance).

The nodes considered in the S^3 problem are users of a social network, able to (1) communicate with private message passing and (2) tag the public profile of each other. As such, every node directly reflects on the associated user. Nodes care about their privacy and their reputation: a user wants neither the private information contained in her input, nor her misbehavior, if any, to be disclosed. This reputation concern is crucial to make the problem tractable. To ensure security, part of the computation consists in checking the correctness of other nodes' behavior. The output of a node p is a value in U plus a set $\mathcal{F}_p$ of nodes that p detected as faulty. This information is eventually reported on the public profile of the involved nodes by means of tags of the form "p detected nodes in $\mathcal{F}_p$ as faulty".

Faulty nodes are considered rational: their goal is only to bias the output of the computation and infer the inputs of the users taking part in the computation. As such, their behavior is more restricted than that of Byzantine users [4]. To achieve their goal, faulty nodes may collude.

In our context, a distributed computation $\mathcal{D}$ on the set of nodes Π, is a sequence of message exchanges and local computations such that any non-faulty node p eventually outputs a value o_p. The content of the message and the nodes' outputs are random variables whose value is determined by the random choices made by the nodes during the computation. In the following, we define the desirable properties of a distributed computation in a social network, namely scalability and security, itself encompassing privacy and accuracy.

2.2 Scalability

Scalability means that the computation is able to handle a large number of inputs (i.e., large values of n): consequently, the properties are expressed in the form of asymptotic bounds.

Definition 2 ($\sqrt{\ }$-Scalability). *A distributed computation is said to be $\sqrt{\ }$-scalable if the message, spatial and computational complexities at each node are $\mathcal{O}(\sqrt{n} \cdot \operatorname{polylog} n)$.*

The intuition behind the logarithmic factor in the asymptotic bound is that operations with the nodes' identifiers and the memory needed to store such identifiers remain within $\mathcal{O}(\log n)$.

2.3 Accuracy

The definition of the accuracy of a computation relies on the metric space structure of the output space U: accuracy is given by the distance between the output of the computation and the actual value of the output of f. To render it meaningful, we normalize this distance by the diameter of $f(V^n)$ for a distributed computation over n nodes.

Definition 3 ($\sqrt{\ }$-Accuracy). *A distributed computation $\mathcal{D}$ is said to $\sqrt{\ }$-accurately compute a S^3 candidate (f, U, V, d) if:*

$$\frac{1}{\Delta(n)} \cdot \max_{p \text{ non-faulty}} d(o_p, f(v_1, \ldots, v_n)) = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right),$$

where v_i is the input of the i-th node and

$$\Delta(n) = \max_{\substack{(x_1, \ldots, x_n) \\ (y_1, \ldots, y_n)}} d(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)).$$

This definition highlights the importance of specifying the distance measure of a S^3 candidate: providing the output space with the coarse grain distance $d(x, y) = 0$ if $x = y$, and 1 otherwise, will restrict the class of S^3 computations to those that output the exact value of f. Meanwhile, for binary polling for instance Dpol [5], considering the natural distance on relative numbers includes computations for which the error on the tally is negligible when compared to the sample size n (i.e., $\Delta(n) = 2n$).

2.4 Privacy

Privacy characterizes how the information gained by curious nodes taking part in the distributed computation enables them to recover the input of a particular non-faulty node. Clearly, the cases where an input can be inferred from only the output and the inputs of the faulty nodes are ignored when looking at the privacy leaks of a computation. In a perfectly private distributed computation, a coalition of faulty nodes should be able to recover the input of a non-faulty node if and only if its input can be inferred from the output of the computation and the inputs of the faulty nodes. Such configurations of inputs are captured by the notion of *trivial* inputs. An example of such configuration of inputs is the case where all non-faulty nodes taking part in a binary poll have the same input, be it -1 or 1. Since S^3 candidates are symmetric by definition, a trivial input is a configuration where all nodes start with the same input.

Definition 4 (Trivial input). *An element v of V^* is said to be a trivial input for a coalition B if there is a node $p \notin B$ such that for all input configuration v' that coincides with v for all nodes in B, $f(v) = f(v')$ implies $v_p = v'_p$.*

We say in our context that a distributed computation is *private* if the probability of recovering the input of a particular non-faulty node (assuming that it cannot be inferred from the output alone, i.e., the configuration of inputs is non-trivial) decreases as $1/n^\alpha$ for some positive α. We capture this notion more formally through the notion of *probabilistic anonymity*, itself based on the very notion of *message trace.*

Definition 5 (Message trace). *A* message trace *(or* trace *for short) of a distributed computation is the collection of messages sent in a possible execution of a program. A trace is said to be* compatible *with an input configuration v if the trace can be obtained from v with a nonzero probability. We say that two traces are equivalent with respect to a coalition of faulty nodes B if each node in B receives the exact same messages in both traces.*

We are ready now to introduce the concept of probabilistic anonymity, which encapsulates the degree of privacy we require.

Definition 6 (Probabilistic anonymity). *A distributed computation $\mathcal{D}$ is said to be probabilistically anonymous if for any coalition of faulty nodes B, for any non-faulty node p, and for any trace D compatible with a non-trivial (w.r.t. B) input configuration v, there exists with high probability a trace D' compatible with an input configuration v' such that (1) D and D' are equivalent w.r.t. B and (2) v and v' differ on the input value of node p.*

The intuition behind this definition is that a coalition of faulty nodes cannot distinguish, with high probability, different executions of a computation in which non-faulty nodes had different inputs.

Definition 7 (S^3 computation). *A distributed computation is said to S^3-compute a S^3 candidate if it is $\sqrt{\ }$-scalable, $\sqrt{\ }$-accurate and probabilistically anonymous with respect to the candidate.*

3 Protocol

In this section, we focus on a class of aggregation functions and propose a protocol, namely AG-S3 (S^3 for AGgregation), which S^3-computes such functions for $|B| \leq \sqrt{n}/\log^2 n$ faulty nodes.

3.1 Assumptions

We consider S^3 candidates for which the function f is an aggregation function, i.e. deriving from an associative binary operation on U: $f(v_1 \ldots, v_n) = v_1 \oplus \cdots \oplus v_n$. Because a S^3 candidate must be symmetric, the '$\oplus$' operation is commutative. This induces a commutative monoid structure on $(U, \oplus)$ and it implies that V is a subset of U. We further assume that the '$\oplus$' operation is *compatible* with the distance measure d in the sense that

$$d(v_1 \oplus v_2, v_1' \oplus v_2') \leq d(v_1, v_1') + d(v_2, v_2') \ . \tag{1}$$

As an example, note that the S^3 candidate $((v_1, v_2, \ldots, v_n) \mapsto v_1 + \cdots + v_n, \{-1, +1\}, \mathbb{Z}, (z_1, z_2) \mapsto |z_1 - z_2|)$, introduced in the previous section, satisfies the compatibility condition described above. A simple example of S^3 candidate which cannot be expressed as an aggregation is the one given by the sum of products of pairs of inputs, i.e. $f(x_1, \ldots, x_n) = x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3 + \ldots$. This function is symmetric, and choosing $U = \mathbb{Z}$ turns this function into a valid S^3 candidate, but it is clearly not an aggregation function.

We assume the size of the set of possible inputs to be constant and the size of the output space to be polynomial in n implying that any input or output can be represented by $\mathcal{O}(\log n)$ bits. In addition, we assume that the diameter $\Delta(n)$ of the output space is $\Omega(n)$. Due to this assumption, bit operators do not fall into our definition. Finally, we assume that V is closed with respect to inverses: if v is in the input set V then $\ominus v$ is in V as well, where $\ominus v$ denotes the inverse of v with respect to the '$\oplus$' operation. We denote by δ_V the diameter of V: $\delta_V = \max_{v,v' \in V} d(v, v')$.

3.2 Design Rationale

The main challenge of S^3 computing is the trade-off between scalability and accuracy on the one hand and privacy on the other hand. We describe below this trade-off and how we address it before describing the protocol in details.

To ensure scalability, we cluster the nodes into groups of size $\sqrt{n}$, and require that a node sends messages only to other nodes in a small set of neighboring groups. We introduce two parameters of the protocol, κ and l. A non-faulty node p is allowed to send messages to any other node in its own group, and to exactly l nodes in each of κ other groups. For scalability, l and κ need to be low, since they are directly proportional to message complexity. The same for accuracy: intuitively, the larger l and κ, the more opportunities a node has to cheat (i.e., corrupt the unique pieces of information it receives before forwarding them), which entails a higher impact on the output. To preserve privacy (i.e. probabilistic anonymity), we need a mechanism which, for any node p, transforms any trace (i.e. input values and messages) into another trace, in such a manner that all messages received by the coalition of faulty nodes are preserved, and p has a different input in the two traces. This prevents the coalition from determining the input value of p. It will become apparent in our proof of privacy that both κ and l need to be large in order to obtain reasonable privacy requirements. To summarize, accuracy and scalability require the parameters κ and l to be small, whereas privacy requires them to be large. As a trade-off, we pick them both to be $\Theta(\log n)$, which reasonably ensure the S^3 requirements.

3.3 Protocol

We describe AG-S3 which computes general aggregation in a S^3 manner: the protocol is composed of two interleaved components: one computes the aggregation function while the other checks the behavior of users. The pseudo-code of all is given in Algorithms 1-4.

Structure. AG-S3 uses a special structure inspired from [6], where the n nodes are distributed into groups of size $\sqrt{n}$. Such an overlay can be obtained in a distributed fashion with strong guarantees on the randomness of nodes placement in the groups even in the presence of malicious users [7]. The groups (or *offices*) are placed in a ring, with nodes from a particular group sending messages to either nodes from the same office (called *officemates*) or to selected nodes from the next offices on the ring (called *proxies*). More specifically, a node is connected to its $\sqrt{n}$ officemates and to l proxies in each of the next κ groups on the ring. If a node p' is a proxy of p, then p is said to be a *client* of p'. The partitioning into groups and their placement on the ring are chosen uniformly at random. We further assume a perfect client-proxy matching that ensures that a proxy has exactly $\kappa \cdot l$ clients. For example, we can index the nodes inside each group and assign to the i-th node of a group the nodes $i+1, \ldots, i+l \mod \sqrt{n}$ as proxies in each of the next κ groups on the ring. We set $\kappa = 3/2 \cdot \lfloor \log n \rfloor$ and $l = 5 \cdot |V| \cdot \lfloor \log n \rfloor + 1$. These choices are motivated in the next section.

Aggregation. In the first phase, each participant splits its input into $\kappa \cdot l$ *shares* in V and sends them randomly to its assigned proxies. The randomized scheme ensures that the aggregate of the shares is the input value. The shares are generated as follows: $(\kappa \cdot l - 1)/2$ are chosen uniformly at random, $(\kappa \cdot l - 1)/2$ are the inverses of the randomly chosen shares, and one is the actual input of the node.

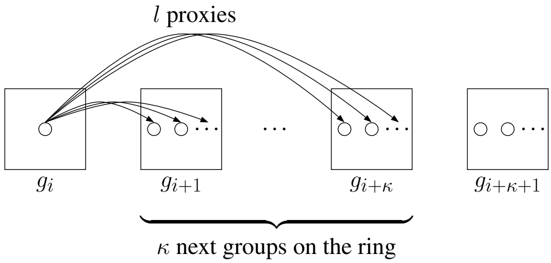


Fig. 1. Overview of the overlay

```
procedure share_input( $v$ );
  for $i \leftarrow 1$ to $(l \cdot \kappa - 1)/2$ do
      $s_i \leftarrow_{\text{rand}} V$   # random values in $V$;
      $s_{i+(l \cdot \kappa - 1)/2} \leftarrow \ominus s_i$;
  $s_{l \cdot \kappa} \leftarrow v$   # the actual input;
  $\sigma \leftarrow_{\text{rand}} \mathcal{S}_{l \cdot \kappa}$   # random permutation to distribute the shares;
  for $i_{\text{group}} \leftarrow 1$ to $\kappa$ do
      for $i_{\text{proxy}} \leftarrow 1$ to $l$ do
          send (SHARE, $p_{i_{\text{group}}, i_{\text{proxy}}}$, $s_{\sigma(i_{\text{group}} \cdot l + i_{\text{proxy}})}$);
```

Algorithm 1. Input sharing

In the counting phase, each proxy aggregates the shares received in the previous phase to obtain an *individual aggregate*. Each node then broadcasts its individual aggregate to all its officemates. Each node computes the aggregate of the individual aggregates of its officemates and obtains a *local aggregate*. If all nodes are non-faulty, then all local aggregates computed in an office are identical.

```
upon event receive (SHARE, $c$, $s$) do
  Verify $c$ is a client;
  Verify $s$ is a valid input in $V$   # $s \in V$;
  $u_{\text{ind}} = u_{\text{ind}} \oplus s$;
```

Algorithm 2. Individual aggregation

In the *forwarding phase*, the local aggregates are disseminated to other nodes thanks to tokens forwarded along the ring, as explained below. The forwarding phase is bootstrapped by a special group (that can be determined by the social networking infrastructure at random). The nodes in this special group send a token containing the local aggregate computed in their group to their proxies from the next group. The tokens are further forwarded along the ring. The first time a token reaches a node in a particular group, this node aggregates the local aggregate to the token and forwards it to its proxies in the next group. When a

```
procedure local_count();
  foreach officemate $o$ do
    send (INDIVIDUAL_AGG, $o$, $u_{\text{ind}}$);
```

Algorithm 3. Local aggregate broadcast

```
upon event receive (INDIVIDUAL_AGG, $o$, $u$) do
  Verify $u$ is a valid aggregate of $\kappa \cdot l$ shares;
    # $d(u, v_1 \oplus \cdots \oplus v_{\kappa \cdot l}) \leq \kappa \cdot l \cdot \delta_V$ where $v_1 \oplus \cdots \oplus v_{\kappa \cdot l}$ are random values in
  $V$;
  $u_{\text{local}} \leftarrow u_{\text{local}} \oplus u$ ;
```

Algorithm 4. Local aggregation

node receives a token for the second time, the node sets its own output to the value of the token and forwards it. The third time a node receives a token, it discards it.

Verifications. The purpose of verifications is to track nodes that deviate from the protocol. This is achieved by leveraging the value attached by the nodes to their reputation. The basic mechanism is that misbehaviors are reported by the participants who discover a faulty node and subsequently tag the latter's profile. The verifications are performed in each phase of the protocol. In the sharing phase, each proxy verifies that the shares received are valid input values. In the second phase, each node checks whether the distance between the individual aggregates sent and some random valid individual aggregate is at most $\kappa \cdot l \cdot \delta_V$. The reason for this is that due to the compatibility of the distance function with the monoid operation, for any $v_1, \ldots, v_k, v'_1, \ldots, v'_k \in V$, we have that

$$d(v_1 \oplus \cdots \oplus v_k, v'_1 \oplus \cdots \oplus v'_k) \leq d(v_1, v'_1) + \cdots + d(v_k, v'_k) \leq k \cdot \delta_V.$$

The verification in the third phase works as follows: if all the tokens received by a node in a given round (remember that tokens circulate up to three times around the ring) are not the same, then an alarm is raised and the profiles of the involved nodes are tagged. Otherwise, the node broadcasts the unique value of the tokens it received to its officemates. If it is not the case that all values broadcast are equal, again an alarm is raised.

3.4 Correctness

We prove here that AG-S3 satisfies the S^3 conditions for $|B| \leq \sqrt{n}/\log^2 n$.

Theorem 1 (Scalability). *The AG-S3 protocol is $\sqrt{\ }$-scalable.*

Proof. The nodes need to maintain a list of officemates, a list of proxies, and a list of clients. This amounts to $\mathcal{O}(\sqrt{n} \cdot \log n)$ space complexity as nodes' identifiers can be represented using $\mathcal{O}(\log n)$ bits. The message complexity is similarly $\mathcal{O}(\sqrt{n})$ arising from the following components: a node sends $\kappa \cdot l = \mathcal{O}(\log^2 n)$ shares during the sharing phase, $\mathcal{O}(\sqrt{n})$ copies of its individual aggregate in the counting phase, and $\mathcal{O}(\sqrt{n})$ in the forwarding phase. □

Theorem 2 (Accuracy). *The AG-S3 protocol is $\sqrt{}$-accurate.*

Proof. A faulty node can bias the output of the computation by either sending an invalid set of shares, changing the value of its individual aggregate, or corrupt the aggregate during the forwarding phase. However, a node never misbehaves in a way that this is exposed with certainty (by the verifications presented in the previous section).

Sharing: Not to be detected, a node must send shares in V. Therefore, the distance between the sum of a node's shares and a valid input is at most $\kappa \cdot l \cdot \delta_V$.

Counting: Suppose that a faulty node changes its individual aggregate from $v = v_1 \oplus \cdots \oplus v_{\kappa \cdot l}$ to some value u. When its officemates receive its individual aggregate u they compute the distance between this aggregate and an arbitrary aggregate $w = w_1 \oplus \cdots \oplus w_{\kappa \cdot l}$. If this distance is larger than $\kappa \cdot l \cdot \delta_V$ then the misbehavior is reported. If the distance is within the bound, the triangular inequality yields an upper-bound on the maximum impact: $d(u, v) \leq d(u, w) + d(w, v) \leq 2\kappa \cdot l \cdot \delta_V$.

Forwarding: To corrupt a token without being detected, the coalition of faulty nodes must *fool* (i.e., make a node decide and forward a corrupted token without raising an alarm) all the non-faulty nodes of a group. Otherwise the corruption is detected by the verification consisting in a node broadcasting the token received to its officemates. To fool a single non-faulty node, all the l tokens it received from its clients (remember that nodes forward tokens only to their proxies in the next group) must be equal. Since nodes have l proxies in the next group, f faulty nodes can fool up to f non-faulty nodes. Assuming that a group contains f non-faulty nodes (and $\sqrt{n} - f$ faulty nodes), then corrupting a token without being detected requires another f faulty nodes in preceding groups. That is a total of $\sqrt{n}$ faulty nodes which cannot happen under the assumption $|B| \leq \sqrt{n}/\log^2 n$. To conclude, the local aggregates cannot be corrupted during the forwarding phase.

The impact of a faulty node on the output of the computation is bounded by $3\kappa \cdot l \cdot \delta_v$. We have $|B| \leq \sqrt{n}/\log^2 n$, $\kappa = \mathcal{O}(\log n)$, $l = \mathcal{O}(\log n)$ and $\Delta(n) = \Omega(n)$. Putting everything together, we get that the accuracy of definition 3 is $\mathcal{O}(\sqrt{n}/\log^2 n \cdot \log n \cdot \log n/n) = \mathcal{O}(1/\sqrt{n})$, which concludes the proof. □

Theorem 3 (Probabilistic anonymity). *The AG-S3 protocol is probabilistically anonymous.*

Proof. We need to show that, with high probability, there exists a mechanism that for any node p, transforms any trace in such a way that the coalition of faulty nodes receives the same messages, but p has a different input. We first give an outline of the proof.

The transformation mechanism consists of changing the values transmitted between non-faulty nodes, in such a way that any subsequent message sent by non-faulty nodes to the nodes in the coalition does not change. As a result, the coalition receives the same information. The basic idea of this mechanism is to *swap* the inputs of two nodes p_1 and p_2, provided that there is a non-compromised group g (a group with no faulty nodes) that contains proxies of both p_1 and p_2. In this case, we can modify the shares sent by p_1 and p_2 to proxies in g, in such a way that the local aggregate of g is maintained. Since we assume that all nodes in g are non-faulty, the coalition does not have access to information exchanged in g during the counting phase. The coalition only sees what the nodes in g decide to broadcast in the forwarding phase, but that is identical to what is sent in the original trace. To modify the shares of p_1 and p_2, we assume that both send a share containing their own input to some proxies in g. Each of p_1 and p_2 has l proxies in g, so the larger l is, the larger the probability that our assumption is true. Then the aforementioned shares of p_1 and p_2 are swapped, resulting a consistent trace, where p_1 and p_2 swapped input values.

In case there is no such common non-compromised group g for p_1 and p_2, we may still find a chain of nodes with endpoints p_1 and p_2, such that two consecutive nodes in the chain can swap input values. The larger κ, the larger the probability that such a chain exists. Afterwards, the nodes can swap shares along the chain, resulting in a consistent configuration where p_1 has as input the old input value of p_2. The rest of the proof is concerned with making our outline description precise.

Let D be a trace of AG-S3 compatible with a non-trivial input v, B be a coalition of faulty nodes ($|B| \leq \sqrt{n}/\log^2 n$) and p be a non-faulty node. Since the input is non-trivial, there exists a node p' whose input is different from the input of p in v, and we prove that with high probability there exists a trace equivalent to D compatible with an input configuration v' which is the same as v, except that the inputs of p and p' have been swapped.

We say that a group compromised if it contains at least one faulty node. The coalition of faulty nodes knows the local aggregates of all the groups, the individual aggregates of the proxies in the compromised groups, the shares they received and their own inputs.

We first prove the following lemma.

Lemma 1. *The probability that in any sequence of $\kappa - 1$ consecutive groups there is at least one non-compromised group, is at least* $1 - \sqrt{n}\left(\frac{|B|}{\sqrt{n}}\right)^{\kappa-1}$.

Proof. This probability is minimized if no two faulty nodes lie in the same group, i.e. there are $|B|$ compromised groups. Fix $\kappa - 1$ consecutive groups. The number of configurations in which these groups are compromised is $\binom{\sqrt{n}-\kappa+1}{|B|-\kappa+1}$. The total number of configurations is $\binom{\sqrt{n}}{|B|}$, so the probability that all the fixed k consecutive groups are compromised is given by the ratio of the two binomial coefficients, which is upper-bounded by $(|B|/\sqrt{n})^{\kappa-1}$. We use the union bound to upper-bound the probability that there is at least one such sequence of $\kappa - 1$ consecutive compromised groups. There are $\sqrt{n}$ sequences of $\kappa - 1$ consecutive groups, which proves the lemma. □

Since $\kappa = 3/2 \cdot \lfloor \log n \rfloor$ and $|B| \leq \sqrt{n}/\log^2 n$, we get that the probability of having κ consecutive compromised groups is at most $1/n$.

Lemma 2. *Given $x \in V$, the probability that a node sends at least one share of value x to a proxy situated in a given group, assuming this node has proxies in that group, is at least $1 - 1/n^3$.*

Proof. The l shares sent to a group by a node are randomly picked from a set of $\kappa \cdot l$ shares in which $(\kappa \cdot l - 1)/2$ are random, $(\kappa \cdot l - 1)/2$ are the inverses of the random shares, and one is the actual input of the node. At least $(l-1)/2$ of them are independent, and drawn uniformly at random from V. Thus, the probability that a is not one of them is at most $(1 - 1/|V|)^{(l-1)/2}$. Since $(l - 1)/2 = 5/2 \cdot |V| \cdot \lfloor \log n \rfloor$, this probability is upper-bounded by $1/n^{5/2}$, which proves the lemma. □

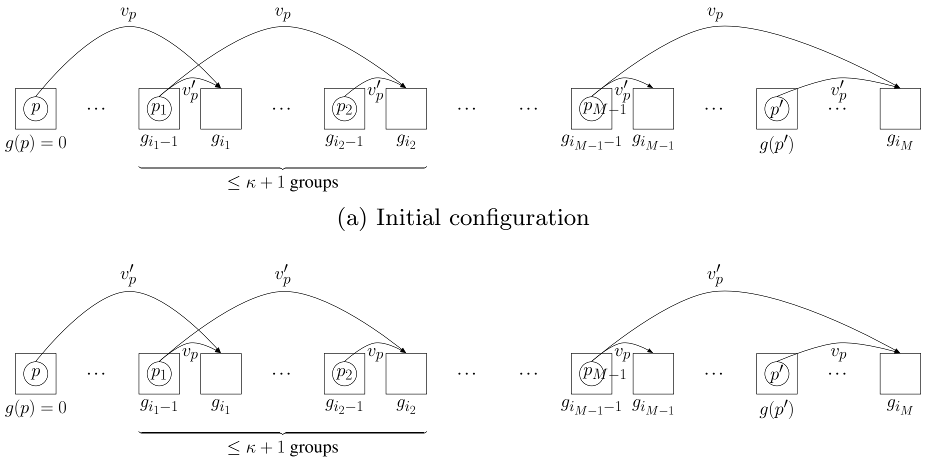
Let $g(\cdot)$ denote the index of a group in which a node lies. Without loss of generality, we assume that $g(p) = 0$. Since we assume that the input v is not trivial, let p' be a node such that its input v'_p is different from the input of p, i.e., v_p. Let $i_1, \ldots, i_M$ be a sequence of indexes such that: (1) group g_{i_m} is non-compromised for all m, (2) $0 < i_1 < \kappa$, (3) $0 < i_{m+1} - i_m < \kappa$ for all m, and (4) $0 < i_M - g(p') < \kappa$. Such a sequence exists with high probability according to Lemma 1. For all $1 \leq m < M$, we define p_m as an arbitrary non-faulty node in group $g_{i_m - 1}$. Additionally, we set $p_0 = p$ and $p_M = p'$. Since all nodes have proxies in the κ groups succeeding them, we have that for all $1 \leq m \leq M$, p_{m-1} and p_m both have proxies in g_{i_m} as depicted in Figure 2.

Using Lemma 2 and using an union bound on the $1 \leq m \leq M$, we get that the probability that for all $1 \leq m \leq M$, p_{m-1} sends a share of value v_p to a proxy in g_m and p_m sends a share of value v_p to a proxy in g_m, is at least $1 - 2M/n^{5/2}$. Since M is bounded by the number of groups, namely $\sqrt{n}$, this probability is lower-bounded by $1 - 2/n^2$.

Assuming that this event occurs, we exhibit a trace compatible with a configuration of inputs where the inputs of p and p' are swapped: for all $1 \leq m \leq M$, the v_p share sent by p_{m-1} to g_{i_m} is replaced by v'_p and the v'_p share sent by p_m to g_{i_m} is replaced by v_p, as illustrated in Figure 2. This trace is equivalent to D with respect to the coalition B as no share sent to a compromised group is changed and all local aggregates remain the same.

We complete the proof by showing that this trace is indeed compatible with the modified configuration of inputs. In the case of AG-S3, compatible means that the set of shares sent by a node is composed of $(\kappa \cdot l - 1)/2$ values of V, their inverses, and the actual input of the node. For p and p', we only change the value of one share equal to their inputs. Therefore, their set of shares remains compatible with their new inputs. For the other nodes p_m, $0 < m < M$, two of their shares are simply swapped.

We proved that the privacy of *a given* non-faulty node p is preserved with probability at least $1 - 2/n^2$, given that the event of Lemma 1 occurs. Since the probability of this event is large (according to Lemma 1), using Bayes rule it is clear that $1 - 3/n^2$ is an upper bound on the probability that privacy of a particular node is preserved. Using a union bound over the whole set of at most n non-faulty node nodes, we obtain that probabilistic anonymity as defined in Definition 6 is preserved with probability $1 - 2/n$. □



(a) Initial configuration

(b) Swapped configuration

Fig. 2. Illustration of the proof of privacy: pairs of shares sent in the same group can be swapped ((a) → (b)) leading to an equivalent trace compatible with a different configuration of inputs.

4 Related Work

Cryptographic primitives and secure multi-party computation [1,2,3] allow to compute aggregation functions in a secure way. This comes however at the price of non-scalability. Assuming trust relationships between users of a social network, Vu *et al.* [8] proposed an improved secret sharing scheme to protect privacy. In that scheme, the actual relationships between nodes are used to determine the trustworthy participants, and the shares are only distributed to those. In contrast, AG-S3 exploits solely the human nature of social networks without making any assumptions on the social relationships themselves.

The population protocol model of *et al.* [9] provides a theoretical framework of mobile devices with limited memory, which relates to the scalability requirement of the S^3 problem. The model however can only compute first order formulas in Presburger arithmetic [10] and can tolerate only a constant number of benign failures [11]. The community protocol model [12] relax the scalability requirements on the memory sizes of tiny agents which enables powerful computations and Byzantine fault-tolerance. Yet, the model breaks anonymity as agents are assigned unique ids. This illustrates the trade-off between the power and security of a model on one hand and privacy on the other hand. The problem of privacy in population protocols was also tackled in [13]. The sharing scheme of AG-S3 is inspired by the obfuscation mechanism proposed in that paper, namely adding unit noise (+1 or -1) to their inputs, upon a state exchange. Dpol [5], itself also inspired by [13], can be be viewed as a restricted form of AG-S3. Dpol is restricted to binary polling: it aggregates values in $\{-1, +1\}$ and it uses a rudimentary secret sharing scheme and overly structure that assume *(i)* a uniform distribution of inputs, and *(ii)* a built-in anonymous overlay: these are the two main difficulties of the privacy challenge as defined in the S^3 problem.

Differential privacy [14] and k-anonymity [15] are two common ways to express privacy in the context of distributed computations on sensitive databases. Contrary to AG-S3, where faulty nodes take part in the computation, those techniques aim at protecting the privacy of inputs from an external attacker that queries the database. Differential privacy characterizes the amount of information disclosed by the output by bounding the impact of a single input on the output. It is typically achieved by adding noise to the output. However, as pointed out in [16], differential privacy does not capture the cases of *rare input configurations* due to the multiplicative bounds in its formulation, which is precisely the difficult case we need to address in the S^3 problem, i.e., the case where everybody but one node have the same inputs. The obfuscating technique consisting in adding noise to intermediate results cannot be used in the context of S^3 computing. The granularity of noise may indeed by high if elements of V are far away. In addition, it gives more opportunities to faulty nodes to bias the output of the computation. On the other hand, k-anonymity guarantees that any input value maps to at least k input nodes. In the S^3 problem, privacy can be seen as 2-anonymity with high probability, expressed in a distributed setting. With AG-S3, faulty nodes cannot map any input to a restricted subset of nodes as any two nonfaulty nodes can swap their inputs transparently. It thus ensures $n - B$-anonymity with high probability.

5 Conclusion

Social networks constitute now huge platforms on which it is very tempting to perform large scale computations. Yet, such computations are challenging as one needs to ensure privacy, scalability and accuracy. We leverage the very fact that, in such platforms, behind every node lies a respectable user who cares about his reputation, in order to make the problem tractable. We define what the notion of computation means in that context and propose a protocol that computes a class of aggregation functions. This is a first step toward understanding what can be computed in a social network and many open questions are left open such as what is the maximum number of faulty nodes a S^3 protocol can tolerate and what else besides aggregation functions can be computed in a S^3 manner?

References

1. Benaloh, J.: Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 251–260. Springer, Heidelberg (1987)

2. Rivest, R., Shamir, A., Tauman, Y.: How to Share a Secret. CACM 22, 612–613 (1979)
3. Yao, A.C.: Protocols for Secure Computations. In: FOCS, pp. 160–164 (1982)
4. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. ACM TPLS 4(3), 382–401 (1982)
5. Guerraoui, R., Huguenin, K., Kermarrec, A.M., Monod, M.: Decentralized Polling with Respectable Participants. In: OPODIS, pp. 144–158 (2009)
6. Galil, Z., Yung, M.: Partitioned Encryption and Achieving Simultaneity by Partitioning. Information Processing Letters 26(2), 81–88 (1987)
7. Gupta, I., Birman, K., Linga, P., Demers, A., van Renesse, R.: Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 160–169. Springer, Heidelberg (2003)
8. Vu, L.H., Aberer, K., Buchegger, S., Datta, A.: Enabling secure secret sharing in distributed online social networks. In: ACSAC, pp. 419–428 (2009)
9. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in Networks of Passively Mobile Finite-state Sensors. Distributed Computing 4, 235–253 (2006)
10. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The Computational Power of Population Protocols. Distributed Computing 20, 279–304 (2007)
11. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When Birds Die: Making Population Protocols Fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006)
12. Guerraoui, R., Ruppert, E.: Names Trump Malice: Tiny Mobile Agents Can Tolerate Byzantine Failures. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 484–495. Springer, Heidelberg (2009)
13. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: Secretive Birds: Privacy in Population Protocols. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878, pp. 329–342. Springer, Heidelberg (2007)
14. Dwork, C.: Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
15. Samarati, P.: Protecting Respondents' Identities in Microdata Release. TKDE 13, 1010–1027 (2001)
16. Roy, I., Setty, S.T., Kilzer, A., Shmatikov, V., Witchel, E.: Airavat: Security and Privacy for MapReduce. In: NSDI (2010)