Scalable and Secure Polling in Dynamic Distributed Networks

Sébastien Gambs[1], Rachid Guerraoui[2], Hamza Harkous[2]
Florian Huc[2], Anne-Marie Kermarrec[3]
[1]Université de Rennes 1 – INRIA/IRISA
[2]École Polytechnique Fédérale de Lausanne (EPFL)
[3]INRIA Rennes Bretagne-Atlantique

***Abstract*—We consider the problem of securely conducting a poll in synchronous dynamic networks equipped with a Public Key Infrastructure (PKI). Whereas previous distributed solutions had a communication cost of $O(n^2)$ in an n nodes system, we present SPP (Secure and Private Polling), the first distributed polling protocol requiring only a communication complexity of $O(n \log^3 n)$, which we prove is near-optimal. Our protocol ensures perfect security against a computationally-bounded adversary, tolerates $(\frac{1}{2} - \epsilon)n$ Byzantine nodes for any constant $\frac{1}{2} > \epsilon > 0$ (not depending on n), and outputs the exact value of the poll with high probability. SPP is composed of two sub-protocols, which we believe to be interesting on their own: SPP-Overlay maintains a structured overlay when nodes leave or join the network, and SPP-Computation conducts the actual poll. We validate the practicality of our approach through experimental evaluations and describe briefly two possible applications of SPP: (1) an optimal Byzantine Agreement protocol whose communication complexity is $\Theta(n \log n)$ and (2) a protocol solving an open question of King and Saia in the context of aggregation functions, namely on the feasibility of performing multiparty secure aggregations with a communication complexity of $o(n^2)$.**

I. Introduction

We define the Polling problem as the one of providing to all the users the result of a poll that is conducted among themselves, thus yielding the number of votes for each candidate. This is an essential functionality of distributed systems that deal with the computation of global functions out of the local values held by the nodes. For instance, polling includes the problem of agreement (as defined in [1]) as a specific case. In general, polling can also be extended to compute any global function that is a linear combination of the local inputs. Such functions are particularly important in large-scale systems in which they are typically used to compute global properties of systems (*e.g.*, for monitoring purposes). While such computations may be achieved through a trusted central entity gathering all local inputs [2], distributed variants are appealing for *scalability* and *privacy* reasons.

In this paper, we address simultaneously *three* fundamental issues related to the distributed implementation of a polling in a dynamic network: namely *correctness*, *privacy* and *scalability*. The computation of a polling function is a specific instance of the much broader problem of secure multiparty computation. Therefore, generic constructions from this domain can be used [3], [4] to solve the problem while tolerating up to $n/2 - 1$ Byzantine (*i.e.*, malicious) nodes, where n is the number of nodes of the system. However, these constructions are often expensive with a global communication cost that is quadratic in n [5]. In addition, most of them assume the existence of a broadcast channel, which is rarely available in large scale networks. Simulating such a channel deterministically is possible but has a communication cost of $\Omega(n^2)$ [6], [7].

The main motivation of this work is to design a distributed polling protocol whose communication complexity is close to be linear in the number of nodes of the system. This is impossible to achieve with certainty (*i.e.*, through a deterministic algorithm) in a secure manner by the following argument: to be certain that a message sent by a node is not altered by a collusion of m Byzantine nodes, it needs to be sent at least $m + 1$ times (since all Byzantine nodes could simply drop the message), which induces $O(nm) = O(n^2)$ messages when $m = O(n)$. Therefore, instead of seeking to design a deterministic algorithm achieving correctness with certainty, we investigate probabilistic algorithms outputting the exact value of the polling *with high probability*.

Our first contribution (Section III) is to prove a lower bound stating that at least $\Omega(n \log n)$ messages are required to compute any multiparty function (not only polling) in an accurate way with high probability. This lower bound holds whenever the adversary controls a constant fraction of the nodes. The lower bound proof leverages on the strategy consisting of the adversary controlling all the neighbors through which a specific node transmits its vote while honest nodes select the nodes that will receive their inputs at random to make it difficult for the adversary to control them all.

Our second and main contribution (Section IV) is SPP (*Secure and Private Polling*), a near-optimal distributed polling protocol protecting the privacy of individual inputs and tolerating up to $(\frac{1}{2} - \epsilon)n$ Byzantine nodes, for any constant $\frac{1}{2} > \epsilon > 0$ independent of n. SPP is designed for dynamic networks and precisely leverages the churn induced by such a network (*i.e.*, when nodes leave and join the system) to maintain an overlay that can be used to compute any polling function. SPP outputs the exact outcome of the poll with high probability as n tends to infinity, has a global communication cost of $O(n \log^3 n)$ bits, and is balanced. In a nutshell, SPP is composed of two sub-protocols. SPP-Overlay is a novel distributed version of the protocol presented in [8] that builds and maintains

an overlay of clusters of size $O(\log n)$, such that each cluster contains a majority of honest nodes. The construction of such an overlay is a necessary condition to ensure the correctness of SPP. SPP-Computation relies on this overlay to conduct the poll, using existing cryptographic techniques [9] within clusters, which would otherwise be too expensive to run at the level of the entire network. Although we partially benefit from existing techniques in distributed systems and cryptography, to the best of our knowledge, we are the first to adapt and to reunite all of them in a dynamic distributed system to obtain a scalable and secure solution to polling.

We further describe two possible applications of SPP (Section V). First, we demonstrate how it can be adapted to give a positive answer to an open question by King and Saia [10] in the context of functions that consist in a linear combination of inputs. Namely, we show that it is possible to compute this type of functions in a dynamic network with a communication complexity of $o(n^2)$ bits, even in the presence of an adversary controlling a constant fraction of the nodes. Second, we propose an optimal Byzantine agreement protocol, with a complexity lower than the state-of-the-art lower bound of $n^{1/3}$ proved in the context of static networks [11]. This lower bound is circumvented by considering the context of dynamic networks, where it is possible that nodes join or leave. In that case, we benefit from our distributed construction of the nodes' overlay structure, which is further maintained dynamically.

Finally, we experimentally evaluate our approach through the implementation of a distributed polling protocol and of an agreement protocol on the Emulab testbed [12] (Section VI).

II. Model and Related Work

A. Related Work

1) Dynamic Overlays: Overlay networks organize nodes of a distributed system in a logical structure in order to allow efficient communications and to minimize the memory overhead. In this work, we are primarily interested in dynamic overlays tolerating the presence of Byzantine nodes controlled by an adversary. For instance, an early work of Scheideler [13] aims at partitioning the the nodes into clusters, each with a majority of honest nodes. Scheideler achieves this property by using a trusted central authority and introducing the concept of *k-rotation*, which organizes nodes along a ring so that any set of consecutive $O(\log n)$ nodes has an honest majority. In a followup work by Awerbuch and Scheideler [14], the *cuckoo rule* was introduced, allowing to maintain this property even when the Byzantine nodes can leave and rejoin the system at their convenience. Since our own approach is inspired by this work, *thereafter we summarize the system and adversary models that are also the ones we adopt in this paper*. The system considered by Awerbuch and Scheideler is composed of n nodes, among which a constant fraction (less than $1/2 - \epsilon$ for some constant $\epsilon > 0$) is controlled by an active, static adversary choosing the nodes it wants to corrupt at the time at which they join the network [15]. The nodes controlled by the adversary are called Byzantine and can behave arbitrarily, for instance by deviating from the protocol specification. The adversary has a complete knowledge of the system while honest nodes only have a local knowledge. Furthermore, the network is *dynamic*: at each time step and for a polynomial number of rounds, the adversary can choose any of the nodes it controls to make it perform a join-leave attack (*i.e.*, to make it leave and rejoin the system). However, the number honest nodes changes within a constant factor over time. In this setting, the cuckoo rule enables to organize the nodes along a ring of length one, so that any segment of length $O(\log n/n)$ contains $O(\log n)$ nodes, among which a majority are honest with high probability (i.e. with probability at least $1 - 1/n$). The validity of this rule was proved under a polylogarithmic number of leave-join attacks. To simplify their proofs, the authors use the extra assumption that at initialization, the system is exclusively composed of honest nodes. Later, this work was extended by the *cuckoo & flip* rule [16], which provides resilience against Denial-Of-Service (DOS) attacks at the expense of reducing the fraction of Byzantine nodes that can be tolerated to $\tau < \frac{1}{5} - \epsilon$.

Other types of adversaries have also been considered in the literature. For instance, the *Chameleon* system [17], which assumes an adversary capable of *blocking* a fraction of nodes, uses a randomized replication scheme to protect distributed servers against DOS attacks. *SHELL* [18], also due to Scheideler, is a dynamic distributed system resistant to *Sybil* attacks that is particularly tailored for heterogeneous systems. Finally, defense mechanisms working against an adversary that can lead nodes to crash were also considered [19]. In that work, the authors propose a distributed protocol to maintain an overlay when the number of crashes and joins is of the order of the maximum degree of the network, which they prove as being optimal. However, unlike our work, they assume that all nodes follow the protocol and that the adversary is *fail-stop*.

2) Secure Multiparty Computation: The main objective of *secure multiparty computation* is to allow participants to compute, in a distributed manner, a joint function over their inputs while at the same time protecting the privacy of these inputs and ensuring the correctness of the output. This problem was first introduced in the bipartite setting by Yao in 1982 [20] and has since become one of the most active fields of cryptography. Since the seminal paper of Yao, generic constructions have been developed for the multiparty setting, and a dichotomy appears depending on the proportion of nodes controlled by the adversary. On one hand, as long as the number of Byzantine nodes is strictly less than $n/2$, it is possible to securely compute (in the cryptographic sense) any distributed function [3]. If one requires unconditional security (in the information-theoretic sense), then the number of Byzantine nodes should be limited to $n/3$ [4], [21] unless a broadcast channel is available [22], [23], [24]. If the broadcast channel is not available as a resource, the bound of $n/2$ still applies. For some distributed functions, these

protocols were proven to be optimal with respect to the number of Byzantine nodes that can be tolerated [25], [26].

On the other hand, it is still possible to ensure some security guarantees when there is a majority of Byzantine nodes. For instance in [26], the authors have shown that the only functions that can be securely computed when a majority of nodes are Byzantine (up to $n-1$) are those that consist in a XOR-combination of n Boolean inputs. Nevertheless, a weaker notion of security can always be fulfilled when there is a majority of Byzantine nodes. This notion still provides privacy and correctness, yet it allows some nodes to abort the protocol while others obtain the (correct) output (thus sacrificing the robustness property). Protocols guaranteeing this notion of security for any number of corrupted parties can be constructed based on cryptographic assumptions [27], [3], [28]. Recent work has been dedicated to protocols achieving full security with an adversarial minority and the weaker notion of security with adversarial majority (see for instance [29] for some impossibility results as well as some positive ones).

In [2], a polling protocol (a specific case of secure multiparty computation) was proposed that computes the outcome of an electronic election while providing cryptographic security for a global communication cost of $O(n)$. However, contrary to our approach, this protocol requires the availability of a trusted entity during the whole computation. Our work is also to be compared to [30], in which the authors propose a protocol computing a $\sqrt{n}$-approximation of an aggregation function (i.e. functions that output a linear combination of the inputs). This is achieved even in the presence of $\sqrt{n}/\log n$ rational adversarial nodes (a weaker form of adversary than Byzantine nodes), with a global communication cost of $O(n^{3/2})$ and without relying on cryptographic assumptions. Finally, in [10], an open question was asked on whether secure multiparty computation can be achieved with a communication complexity of $o(n^2)$. In our work, we positively answer this question in the context of functions that output a linear combination of the inputs.

3) Byzantine Agreement: The problem of Byzantine Agreement (BA) was introduced originally by Lamport et al. [1] and consists in having all the honest nodes agreeing on a bit value among the input bits initially proposed by them. Clearly, this problem can be solved by running an election protocol between two candidates (named 0 and 1). Running a secure multiparty computation would give even more privacy guarantees as it also ensures that the adversary does not learn any additional information about the inputs of honest nodes. A lower bound of $\tilde{\Omega}(n^{1/3})$ was proved in the context of static networks. However, this lower bound does not hold in the dynamic case as shown by our work.

The most efficient BA protocols that can be found in the literature are probabilistic and have a communication complexity of $\tilde{O}(n^{3/2})$ [31]. More precisely, the protocols proposed in [32] and [31] have sub-quadratic communication complexity against a Byzantine adversary controlling less than $n/(6+\epsilon)$ (for some constant $\epsilon > 0$) and $n/3$ nodes respectively, in the full information model. However, these protocols assume that all nodes know the identities of all the other nodes in the system, which in itself hides an $\Omega(n^2)$ communication complexity to propagate this information.

4) Homomorphic encryption: In our work, we rely on a cryptographic primitive known as *homomorphic encryption*. This primitive allows to perform arithmetic operations (such as addition and/or multiplication) on encrypted values, thus protecting the privacy of the inputs of honest nodes. Paillier's cryptosystem [33] is an instance of a homomorphic encryption scheme. This cryptosystem is also *semantically secure* [15], which means that a computationally-bounded adversary cannot derive non-trivial information about the plain text m encrypted from the cipher text $\mathsf{Enc}(m)$ and the public key pk. In this paper, we use a *threshold* version of the Paillier's cryptosystem [9]. In this version, all nodes get the same public key, with which they can encrypt messages. However, each node gets a share of the private key and uses it to produce a share of the decrypted output. Only upon receiving $t-1$ other shares can a node compute the decrypted output.

B. Model

1) System and Adversary: We use the system and adversarial model of [8], [34] as summarized in Section II-A). Notice, however, that the assumption that, at the beginning, all the nodes are honest is only used to apply the results proved in [8], [34]. We do not need it for any other reason in this work. In particular, the initialization of the overlay works in the presence of the adversary.

2) Key Management: We assume that each node gets assigned a pair of private/public keys for digital signatures by a (trusted) Certification Authority (CA), which therefore corresponds to a form of Public Key Infrastructure (PKI). When a node joins the network, it receives its private key as well as a certificate signed by the CA containing its corresponding public key. The public key is assumed to be unique and chosen at random and thus can be considered as the node identifier. Note that this CA is dedicated to key management and is not used for any other task. Some decentralized implementations of such a CA are possible, but they are out of the scope of this paper. Moreover, these keys are different from those of the threshold cryptosystem detailed later in Section IV-D2. We also assume that all nodes in the system communicate via pairwise secure channels, which means that all the communications exchanged between two nodes are authenticated and confidential from the point of view of an external eavesdropper.

3) Network and Communication: Our protocols work in the synchronous model in which the communications between nodes proceed in rounds. We assume that a node can communicate with any other node as long as the identifier is known to it. We do not assume that the nodes know the identities of all the other nodes in the system, except during the initialization phase (Section IV-C). In particular, once the initialization phase has been completed, each node will only need to have a local knowledge of a polylogarithmic number of other nodes in

the system (the set of nodes a particular node knows varies as the protocol proceeds but the size of this set always remains bounded). Notice that ensuring full knowledge whenever a node joins the system would have a message complexity of n, which is not scalable as we consider a polynomial number of leave-join operations.

4) Polling Computation: Occasionally, the nodes within the system conduct a poll in a distributed manner. This poll is performed on the individual inputs $(x_1, \ldots, x_n)$ of the nodes, in which x_i is taken from a set of ℓ different possible candidates (*i.e.*, $x_i \in L = \{\nu_1, \ldots, \nu_\ell\}$). The poll result allows each node to know the number of votes received by each candidate. We restrict ourselves to the polling problem for ease of presentation, but our protocol can be extended to evaluate any property of the network that can be obtained by a linear combination of the local inputs. Examples of such functions include the byzantine agreement and the computation of the average (or the sum) of inputs (*i.e.*, $f(x_1, \ldots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i$). While our protocol maintains an overlay under churn, we assume that the network remains static during the computation of the polling function. Therefore, any join operation occurring during such a computation is postponed to the end of the protocol.

Due to the presence of Byzantine nodes, the computation needs to be achieved in a *secure* way, in the sense that it should offer some guarantees on the privacy (Definition 1) of local inputs of honest nodes and on the correctness of the output (Definition 2), and this against any actions that the adversary might do.

Definition 1 (Privacy [15])**.** *A distributed protocol is said to be* private with respect to an adversary *controlling a fraction τ of nodes if this adversary cannot learn (except with negligible probability) more information from the execution of the protocol than it could from its own input (*i.e.*, the inputs of the Byzantine nodes it controls) and the output of the protocol.*

In this work, we want to achieve privacy against a *computationally-bounded adversary* that coordinates the Byzantines nodes but does not have enough computing resources to break a cryptographic assumption on which the techniques are based (such as factorizing the product of two big prime numbers or solving the discrete logarithm problem). Furthermore, we aim at ensuring correctness (Definition 2), even if the Byzantine nodes controlled by the adversary misbehave.

Definition 2 (Correctness)**.** *A distributed protocol is said to be correct with respect to an adversary controlling a fraction τ of the nodes if the output of the protocol is guaranteed to be exact with high probability.*

Moreover, we are looking for *scalable* protocols, with low computational complexity, and with a global communication cost as close as possible to the lower bound $\Omega(n \log n)$ that we prove in Section III. In addition, we also aim at achieving a *balanced* protocol (Definition 3), in which each node receives and sends approximately the same quantity of information.

Definition 3 ((C_{in}, C_{out})-balanced)**.** *A distributed protocol among n nodes whose communication complexity is C_{total} is said to be* (C_{in}, C_{out})-balanced*, if each node sends $O(C_{in}C_{total}/n)$ and receives $O(C_{out}C_{total}/n)$ bits of information where C_{total} is the total number of bits sent by all the nodes.*

For instance, in a $(1, 1)$-balanced protocol, each node sends and receives the same number of bits (up to a constant factor) whereas, in an (n, n)-balanced protocol, a single node could do all the work. In this work, we will say that a protocol is *balanced* if it is $(Poly(\log n), Poly(\log n))$-balanced.

III. Lower Bound on Secure Multiparty Computation

In this section, we show that no balanced algorithm (with respect to Definition 3) can compute a polling function with a global communication complexity of $o(n \log n)$ provided that the number of Byzantine nodes is linear in the number of nodes in the system. The lower bound is obtained as a corollary of the following theorem and applies regardless of the privacy guarantees that the protocol seeks to achieve.

Theorem 1 (Lower bound on secure multiparty computation)**.** *Consider a distributed protocol that computes a function whose inputs are held by n nodes, among which ϵn are Byzantine for some positive constant $0 < \epsilon < 1$ (independent of n). Suppose that a fraction cn of the nodes sends no more than $\omega^+(n)$ messages (for some $\omega^+(n) = o(\log n)$ and constant $1 > c > 0$). Assume further that no node receives more than $\omega^-(n)$ messages (with $\omega^+(n)e^{\omega^+(n)}\omega^-(n) = o(n)$). These conditions imply that, with high probability (in n), there is a node whose messages are all intercepted by Byzantine nodes.*

Proof: We refer the reader to the long version of this paper [35] for the details of the proof. ■

Corollary 1. *Consider a $(Poly(\log n), n)$-balanced protocol computing with high probability the exact value of a function in a distributed manner, such that its inputs are held by n nodes among which ϵn are Byzantine (for some positive constant $\epsilon < 1$ independent of n). Then this protocol induces a total of $\Omega(n \log n)$ messages.*

IV. Secure and Private Polling (SPP)

A. SPP in a nutshell

We now provide a high-level view of our SPP (Secure and Private Polling) protocol that performs a poll in a decentralized setting (see also Figure 1). SPP relies on SPP-Overlay, a distributed version of the protocol in [8], described in Section IV-B. SPP-Overlay organizes the n nodes into g clusters, $C_0, \ldots, C_i \ldots, C_{g-1}$, of roughly the same size. The clusters are further arranged into both a Chord overlay [36] and a binary tree overlay. SPP-Overlay assumes that a threshold homomorphic cryptosystem [9] has been set up and that each node knows the public key pk and its share sk_i of the secret key of the threshold

cryptosystem. This is ensured by a precomputation phase detailed later.

SPP-Computation proceeds as follows:

1) After making its choice, each node does a preprocessing step (described below), transforming this choice into an input value that can be added with others. It then encrypts this new input using pk and securely broadcasts it within its cluster. Secure broadcast ensures that (1) all honest nodes receive an identical message, regardless of the actions of the sender and the Byzantine nodes (*consistency*), and (2) this output is the message of the sender, provided it is honest (*validity*) [7]. Hereafter, we will be using the secure broadcast protocol of Dolev and Reischuk [7] to emulate the broadcast channel.

2) The nodes in a given cluster agree in a distributed and secure manner on a common random string $rand$, which is the randomness injected into the homomorphic encryption (we refer the interested reader to [15], [9] for further details). Afterwards, each node adds the encrypted inputs it received from its own cluster using the addition operation of the homomorphic cryptosystem. The result of this addition is called the *local aggregate* and is the same for each honest node of the cluster.

3) Starting from the clusters at the leaves of the binary tree, the nodes of these clusters send their local aggregates to all the nodes of their parent clusters. The nodes of the latter add their own local aggregate with the two received ones from their children, thus forming the *partial aggregate*. A partial aggregate is adopted from the child cluster if and only if the same message was sent by a majority of nodes from this cluster. This majority decision rule is used to discard inconsistent messages that have been sent by Byzantine nodes. The partial aggregates are then propagated towards the root, by repeating this process $O(\log n)$ times. When the partial aggregates reach C_0, we say that the partial aggregate has become the *global aggregate*.

4) The nodes of the root cluster perform a threshold decryption of the global aggregate, thus revealing the output of the protocol, which is propagated down throughout the binary tree. Each node then processes the output (as described later) to discover the polling result.

B. SPP-Overlay

For the purpose of our polling protocol, nodes in SPP are organized into clusters $C_0, \ldots, C_{g-1}$, each of size $O(\log n)$. To achieve this, we assume that the nodes join the network according to a distributed version of the protocol presented in [8], where the original protocol relied on a trusted central authority (CA). Before showing how to avoid the CA, we first describe its functionality. Each node is assigned by the CA a random position on the segment $[0, 1)$. By inducing artificial churn when a node joins the system, the protocol of [8] is proven to ensure that each segment of size $c \log(n)/n$, for some specific constant c, contains a majority of honest nodes, under the condition that the adversary controls at most a fraction of $\frac{1}{2} - \epsilon$ of Byzantine nodes, for some constant $\frac{1}{2} > \epsilon > 0$, independent of n. The clusters $C_0, \ldots, C_{g-1}$ are composed of the nodes whose positions are in the respective segments $[0, c \log n/n), \ldots, [1 - c \log n/n, 1)$.

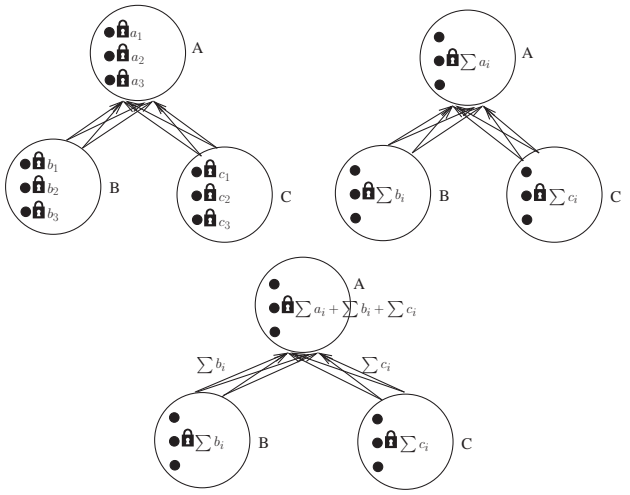


Fig. 1. Main idea of the algorithm. First, all nodes start by encrypting their inputs and broadcasting them to all the other nodes of the cluster. Within each cluster, each node computes the local aggregate ($\sum a_i$, $\sum b_i$ and $\sum c_i$), which is then propagated along the binary tree. After this, the nodes of cluster A know $\sum a_i + \sum b_i + \sum c_i$. The nodes of the last cluster (here A) collaborate to perform the threshold decryption and to output the polling result.

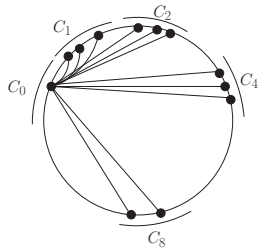


Fig. 2. Chord overlay.

In order to design an efficient distributed version of SPP, we first arrange the *clusters* in a Chord-like overlay [36]. The adapted join-leave protocol must further ensure that, for all $0 \leq i \leq g-1$, the nodes from cluster C_i know all the nodes from $C_{(i+2^{j-1}) \mod g}$, for all $1 \leq j \leq \log_2 g$, which results in $O(\log^2 n)$ connections per node (see Figure 2). Along with this Chord overlay, we consider a binary tree of depth $O(\log n)$ connecting the nodes of each cluster C_i to those of C_{2i+1} (for $2i+1 < g-1$) and C_{2i+2} (for $2i+2 < g-1$).

In [8] the authors assume that initially the network is composed of only honest nodes; instead, we propose a bootstrapping technique for the construction of the overlay in Section IV-C. We now describe how to adapt the join rule, called *cuckoo rule* [8] and how to maintain the overlay in a decentralized way. The leave rule of [16], which is required when the adversary can force any node to leave the network (for instance through a DOS attack), can be adapted similarly (the full details of this adaptation can be found in the long version of this paper [35]). More precisely, we rely on two following subroutines to distribute the cuckoo rule.

1) *Inter-cluster communication.* A node of cluster C receiving a message from a node of cluster C' accepts it if and only if C' is a neighbor of C in the Chord overlay or in the binary tree, and at least half of the nodes of C' have sent the same message.
2) *Random number generation (RNG).* This primitive is used to generate a number at random within some predefined range (typically between 0 and $\log^d n$ for some constant d). For fault-tolerant distributed random generation, notice that adding any set of numbers in a finite field $\mathbb{F}$ outputs a value taken uniformly at random as long as at least one element of this set is chosen uniformly at random, independently from the others. Hence, if each node commits to a random number without knowing what the other nodes have selected, the sum of these committed values is truly random if at least one of them is so. For this, we rely on the *Verifiable Secret Sharing (VSS)* notion introduced in [37]. In its distributed version, t-tolerant VSS allows a certain party (called the dealer) to spread shares of a secret s among m parties such that no collusion of t or less nodes can (1) infer any information about s or (2) prevent the reconstruction of s out of the shares. VSS has two phases: sharing and reconstruction. In the sharing phase, the nodes exchange messages in order to spread the dealer's shares. Next, in the reconstruction phase, the shares are combined by the nodes to either (1) recover the same secret s or (2) tag the dealer as cheater. In this work, we use the $(n-1)/2$-tolerant probabilistic VSS protocol of [38]. Accordingly, the *RNG* protocol in a set S of nodes proceeds as follows: (1) each node i in S chooses a random number $r_i \in \mathbb{F}$. (2) Each node i acts as a dealer and shares r_i using the VSS protocol. (3) For each node i, the reconstruction phase of the VSS protocol is triggered to recover r_i. If reconstruction fails, the zero element in $\mathbb{F}$ is used. (4) Each node i calculates the sum in $\mathbb{F}$ $r = \sum_j r_j$.

We now explain how the cuckoo rule (*i.e.*, join rule) is modified in order to avoid a CA. We assume that a node x joining the network is able to contact one of the nodes of the network which gives it the identity and composition of an arbitrary cluster (this cluster is chosen among the clusters that the contacted node knows). In turn, x contacts the whole cluster (*i.e.*, all its members) with a join request. Afterwards, this cluster starts to perform the cuckoo rule from [8], which corresponds to choosing a position p at random in $[0,1)$ for x. The nodes of the unique cluster containing p, to which we refer as C, are informed via messages routed using the Chord overlay that x is inserted at position p. At this point, extra churn is induced: for a constant $k > \frac{1}{\epsilon}$, C chooses a new random position for all the nodes of C whose positions are in a segment of length k/n containing p (see [8] for more details). Whenever a node x of a cluster C' is required to change its position to join a cluster C", all the nodes that were adjacent or become adjacent to this node, are informed of the change by messages sent by the nodes of C' and C" respectively. This step is crucial since all the nodes of a cluster C adjacent to C' in the Chord overlay or in the binary tree have to know the exact composition of C' in order to decide whether or not to accept a message from nodes of C' during inter-cluster communication. With high probability, *the communication overhead of the distributed version of this protocol is* $O(\log^3 n)$ *per join operation*: for each node to be moved, the overhead can be proved to be $O(log^3 n)$ for the random number generation and $O(logn) \times O(log^2 n)$ for propagating the information using the Chord overlay. Moreover, the expected number of nodes in a segment of size k/n is $O(k)$; thus the protocol needs to move a constant number of nodes with high probability. This protocol ensures that each cluster contains $\Omega(\log n)$ nodes, among which there is a majority of honest nodes as long as τ the fraction of Byzantine nodes is less that $1/2 - \epsilon$.

C. Initialization Phase

In [8], [13], [16], [34], the proofs are done assuming that initially the network is exclusively composed of honest nodes. However, we do not use this hypothesis to bootstrap our protocol as we explain below.

In [10], the authors proposed a protocol for a fully connected network composed of at least $1/2+\epsilon$ fraction of honest nodes, which ensures that all honest nodes agree on a small representative set C of nodes containing a majority of honest nodes with high probability. The communication cost of this algorithm is $\tilde{O}(n^{3/2})$. Once selected, this representative set can compute a random partition of the nodes into $g = O(n/\log n)$ sets of equal size $\{C_0, \ldots, C_{g-1}\}$, which are organized into a Chord overlay and a binary tree. This partitioning method has a communication cost of $\tilde{O}(n^{3/2})$ due to the use of the BA protocol. In addition, it incurs (1) a cost of $\tilde{O}(n)$ for the representative set to agree on the partition and (2) a cost of $\tilde{O}(n)$ to propagate the partition (*i.e.*, by having the nodes of C sending to each node x the composition of its cluster (C_i such that $x \in C_i$) and the composition of its neighboring clusters. Overall, this results in a global communication cost of $\tilde{O}(n^{3/2})$ for the initialization phase (note that this phase is performed only once at the construction of the overlay).

D. SPP-Computation

In this section, we present the second phase of the SPP protocol during which the polling result is effectively computed. This protocol (1) is optimal up to a logarithmic factor in terms of scalability (*i.e.*, communication and computational complexity), (2) has a round complexity of $O(\log n)$, (3) achieves perfect security (*i.e.*, privacy and correctness) against a computationally-bounded adversary controlling at most $(1/2 - \epsilon)n$ Byzantine nodes, for a constant $\frac{1}{2} > \epsilon > 0$ independent of n, and (4) is balanced.

1) Input pre-processing: We assume that each node of the network knows the following public parameters: ℓ (number of possible choices), L (list of possible choices), and s (taken as the length of the RSA modulus in the cryptosystem used). Choices are numbered from 0 to $\ell - 1$. We aim at mapping these choices to input values of an aggregation protocol and to be able to retrieve, from the aggregate, the number of votes for each choice. One way to do so is mapping choice i to the input

value 2^{ib}, where $b = s/\ell$. Any input outside the set $M = \{1 \dots 2^{ib} \dots 2^{(l-1)b}\}$ for $i \in \{0 \dots l-1\}$ is considered invalid.

2) Setting up the threshold cryptosystem: C_0 (that we refer thereafter as the *threshold cluster*) is in charge of setting up the threshold cryptosystem. This set-up phase requires all nodes of the threshold cluster to engage in a distributed key generation protocol [39] for a threshold homomorphic cryptosystem [9]. At the end of this key generation phase, all the nodes of the threshold cluster receive the same public key pk and each gets a share of the private secret key $(sk_1, \dots, sk_{k \log n})$. The threshold cryptosystem is such that any node can encrypt a value using the public key pk but that the decryption of a homomorphically encrypted value requires the active cooperation of at least t of the nodes. In our case, the parameter t is set to be at least $\frac{k \log n}{2}$ to ensure that there will be enough honest nodes to cooperate for the final decryption of the result at the end of the protocol. The public key pk is then communicated in the network cluster by cluster by following the structure of the binary tree. Thereafter, when we say that *a cluster communicates* a value *to the next cluster*, we mean that at least all the honest nodes in the current cluster communicate the same value to all the nodes in the next cluster using inter-cluster communication, which results in a communication cost of $O(\log^2 n)$. Once the previous round is over, the node decides, via a majority rule, on the final value for this particular round of inter-cluster communication. This value always corresponds to the unique value sent by all the honest nodes of the previous cluster, which are by construction a majority within this cluster (*cf.* inter-cluster communication).

3) Local aggregation: Each node within a cluster communicates its input encrypted using the public key pk to all the other nodes of its cluster through a secure broadcast channel along with a non-interactive zero-knowledge proof that this input is valid [40]. The complexity of constructing such a channel is polynomial in the number of nodes of the cluster, which is here $O(\log^2 n)$ and can be obtained for instance by running the broadcast protocol proposed by Dolev and Reischuk [7]. The main objective of the non-interactive zero-knowledge proof is to prevent an adversary from tampering with the output of the protocol by providing an invalid input. The privacy of the inputs is preserved by the semantic property of the cryptosystem [9] that we use. Once all nodes of the cluster have received the encrypted inputs from the other members, they add them using the additive property of the homomorphic cryptosystem. With respect to the randomness used in the addition operation of the homomorphic encryption, we assume that all the nodes have agreed on a common value $rand$.

4) Global aggregation and threshold decryption: The global protocol proceeds iteratively during $O(\log n)$ iterations. The nodes from leaf clusters in the binary tree send their local aggregate to their parent cluster. When a node receives the aggregate from the nodes of both of its child clusters, it adds its local aggregate and the two received ones, which gives a partial aggregate. This partial aggregate corresponds to the aggregation of the inputs of the nodes in the clusters in the subtree rooted at the current cluster; this aggregate is further transmitted to the nodes of the parent cluster. As mentioned previously, the encrypted value received from the previous cluster can be decided on by each node of the current cluster using a majority rule on the $O(\log n)$ messages received from the previous cluster. Once the threshold cluster has been reached (*e.g.*, the root of the binary tree), the members of this cluster add their local aggregated values to the partial aggregates received from the two children, producing an encryption of the sum of all the values. Finally, the members of the threshold cluster cooperate to decrypt this global aggregate by using their private key shares. Along with their decryption shares, the nodes send a non-interactive zero-knowledge proof showing that they have computed a valid decryption share of the final outcome [9]. As the number of nodes needed to decrypt successfully is $t = \frac{k \log n}{2}$ and that there is a majority of honest nodes in the cluster, this threshold decryption is guaranteed to be successful. The final output is forwarded cluster by cluster, following the binary tree structure of the overlay.

5) Output post-processing: Given the decryption result r, each node can know the number of voters for choice i by calculating $\lfloor (r \mod 2^{b(i+1)}) 2^{ib} \rfloor$, except for $i = l-1$, whose number of voters is $\lfloor r/2^{i(l-1)} \rfloor$. To see why this works, notice that $r = k_0 + \ldots + k_i 2^{ib} + k_{i+1} 2^{(i+1)b} + \ldots k_{l-1} 2^{(l-1)b}$, where k_i is the number of votes for choice i. Taking $r \mod 2^{b(i+1)}$ gives $k_0 + \ldots + k_i 2^{ib}$. Dividing by 2^{ib} produces $k_0/2^{ib} + \ldots + k_i$, whose floor is simply k_i. The special case of $i = l-1$ is easy to see. This method works as long each choice gets less than 2^b votes, where b equals s/l as described previously. Hence, $b = s/l$ should be chosen such that $2^b < n$, which is easy to guarantee with typical values of s.

E. Analysis of SPP-Computation

During an execution of the SPP-Computation protocol, the messages need to be long enough to encode all the possible outcomes of the polling. In particular, if the poll is among ℓ choices, the message need to have at least $\ell \log n$ bits. The following lemmas summarize the main properties of SPP-Computation protocol in terms of communication cost (i.e. number of messages sent by the honest nodes, messages that all are of identical size).

Lemma 1 (Communication cost)**.** *SPP-Computation has a global communication cost of $O(n \log^3 n)$. Furthermore, it is $(Poly(\log n), Poly(\log n))$-balanced, in the sense that no node sends or receives more than $Poly(\log n)$ bits of information, with an average of $O(\log^3 n)$ bits of information per node.*

Proof: During the setup of the threshold cryptosystem and the threshold decryption, only one cluster is involved and the communication cost of the primitives used (threshold cryptosystem setup, secure broadcast, and threshold decryption) is polynomial in the size of the cluster, which corresponds to a communication complexity

of $O(Poly(\log n))$. During the local aggregation, in each cluster, each node broadcasts its encrypted input and a broadcast induces a communication cost of $O(\log^3 n)$. As there are $O(n/\log n)$ clusters with $O(\log n)$ nodes in each cluster, it results in a communication cost of $O(n \log^3 n)$. Finally, the inter-cluster communication requires that all the n nodes send $O(\log n)$ messages, each of size $O(\log n)$, to the nodes of the parent cluster, resulting in a communication cost of $O(n \log^2 n)$. As a result, the protocol is dominated by the local aggregation part, which leads to a global communication cost of $O(n \log^3 n)$. Moreover, it is easy to see from the description of the protocol that it is balanced in the sense that it requires $O(Poly(\log n))$ communications from each node. ■

Furthermore, SPP-Computation is near-optimal as its complexity is $O(n \log^3 n)$ compared to the lower bound of $\Omega(n \log n)$.

Lemma 2 (Security)**.** *SPP-Computation ensures perfect security against a computationally-bounded adversary controlling up to* $(\frac{1}{2}-\epsilon)n$ *Byzantine nodes for any constant* $\frac{1}{2} > \epsilon > 0$ *(not depending on* n*) and outputs the exact value of the polling with high probability.*

Proof: The privacy of the inputs of individual nodes is protected by the use of a cryptosystem that is semantically secure and also by the fact that the adversary cannot decrypt the partial aggregate because it does not know the necessary t secret keys of the threshold cryptosystem to do so. The correctness is ensured by a combination of several techniques. First, the non-interactive zero-knowledge proof that each node issues along with the encrypted version of its value guarantees that the Byzantine nodes cannot cheat by choosing their values outside the range of the possible ones. Second, the secure broadcast ensures that honest nodes in each cluster have the same local aggregate. Third, the fact that the majority decision rule is used every time the nodes of a cluster communicate with the nodes of the next cluster along with the fact that there is a majority of honest nodes in each cluster (due to the construction of the structured overlay) ensures that the correctness of the partial aggregate will be preserved during the whole computation. Finally, the non-interactive zero-knowledge proof of the validity of the partial shares during the threshold decryption prevent the Byzantine nodes from altering the output during the last step of the protocol. ■

V. Applications of SPP

In this section, we outline how to adapt SPP to obtain SPP-BA, an optimal Byzantine Agreement protocol. Note that the guarantees provided by SPP-Computation are stronger that what is really needed for a BA protocol, as it also protect the privacy of inputs of the nodes.

A. Optimal Byzantine Agreement

We first run the protocol SPP-Overlay in order to construct a partition of the nodes into clusters of size $O(\log n)$ organized in a binary tree, each containing a majority of honest nodes. Afterwards, the protocol SPP-BA goes as follows:

1) The nodes from C_0 execute a Byzantine Agreement protocol (such as [41]) among themselves and agree on a common bit b.
2) Each node from cluster C_0 sends b to all the nodes of C_1 and C_2 according to the binary tree structure.
3) A node from cluster C_i receiving b from the nodes of its parent cluster uses a majority rule to select the correct value of b and forwards it to all the nodes of its cluster's children (step 3 is repeated cluster by cluster following the binary tree structure until the leaves are reached).

We now analyze the communication cost of this protocol. Step 1 has a communication cost of $O(\log^{3/2} n)$ as $|C_1| = O(\log n)$. Step 2, as well as each run of Step 3, has a communication cost of $O(\log^2 n)$. Therefore, as there are $O(n/\log n)$ clusters involved, *this results in a global communication cost of* $O(n \log n)$ *for SPP-BA, which matches the lower bound of Theorem 1.* Moreover, this protocol is (k, k)-balanced for some constant k and has a round complexity of $O(\log n)$ rounds.

B. Secure Multiparty Aggregation

In [10], King and Saia ask whether or not it is possible to perform secure multiparty computation using $o(n^2)$ bits of communication. For the specific case of functions computing a linear combination of inputs, SPP-Computation enables us to answer positively to this question. Indeed, it is possible to run the initialization phase of SPP-Overlay, which constructs the overlay from scratch, in the presence of Byzantine nodes, for a communication cost of $\tilde{O}(n^{3/2})$, as shown in Section IV-C . Moreover, as SPP-Computation can be easily extended to compute any linear combinations of the inputs of nodes (*cf.* Section IV-D), this gives us the claimed result.

VI. Experimental Evaluation

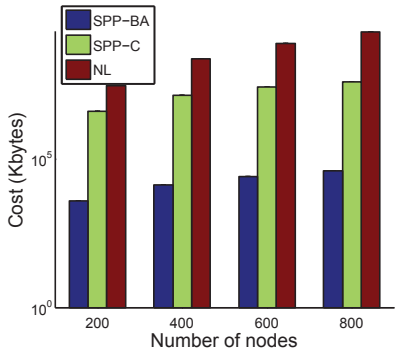
In this section, we evaluate the practical performances of (1) SPP-Computation, for a binary poll, which we compare against a protocol based on unstructured networks and (2) SPP-BA, our optimal Byzantine agreement protocol based on SPP. More precisely, we will evaluate and compare these protocols by measuring their communication and computational costs. The experiments were conducted on the Emulab platform [12], a distributed testbed allowing the user to choose a specific network topology using NS2 configuration file. In each experiment, we use up to 80 PC3000 machines, which correspond to Dell PowerEdge 2850s systems with a single 3 GHz Xeon processor, 2 GB of RAM, and 4 available network interfaces. Each machine runs Fedora 8 as its operating system and hosts 10 nodes at the same time. The nodes are connected to the router in a star topology, setting the maximum network bandwidth to 1000Mb, and the communication relies on UDP. We use a reliable broadcast protocol of [42] and the "Paillier Threshold Encryption Toolbox" [43] for threshold encryption. The nodes adjust their message sending rate to be uniformly distributed between 0 and 2 seconds. For each network size, we do five experimental runs, where each run involves a significant number of nodes over which the metrics are averaged. We use 95% confidence intervals,

displayed on all the figures. Due to the large number of nodes on which the averages are obtained, the intervals are very narrow and almost unnoticeable.

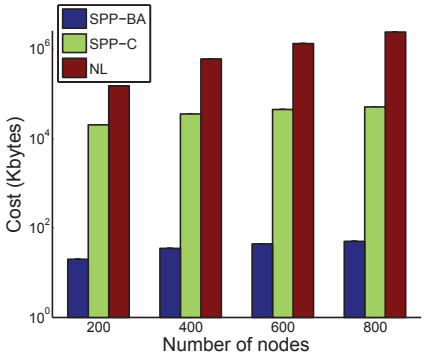
We perform the comparisons with the non-layout based protocol (NL) (i.e. that of unstructured networks), in which each node 1) securely broadcasts its encrypted input to all other nodes, 2) combines the values it receives, 3) securely broadcasts its decryption share to the others, and 4) combines the decryption shares to obtain the final output. Such a protocol provides privacy and correctness with certainty against an honest majority but requires a communication cost $O(n^3)$. Therefore, even for medium sized networks (*e.g.*, 200 to 800 nodes), the communication cost of the protocol is significant with hundreds of millions of messages being exchanged. In order to compare this protocol to ours, we quantify the complexity of single instances of a secure broadcast. Since the broadcast runs are assumed to be run sequentially to avoid congestion, we can add up the cost of each. For ease of implementation, we use a centralized key generation authority to set up the threshold version of the Paillier's cryptosystem, whose modulus is fixed to 1024 bits. Moreover, we set the cluster size to be $20 * \log n$, which we found empirically to be adequate for an adversary controlling a fraction of $\frac{3}{10}$ nodes.

1) Communication Cost: This cost is quantified by the total number of megabytes sent during the execution of the protocol. Figure 3.a depicts the global communication cost using a semi-log scale, with a varying network size. We observe that the polling function SPP-Computation is much more efficient than NL in terms of communication cost. For instance, in a network composed of 200 nodes, NL has a global communication cost of approximately 30GB whereas SPP-Computation communication cost is approximately 4GB. Figure 3.b further details the communication cost per node. While it keeps increasing with NL, SPP-Computation generates a lower cost per node which is almost independent of the network size. Moreover, the cost of the Byzantine Agreement protocol (SPP-BA) is significantly lower than the one of SPP-Computation because most broadcasts are avoided in SPP-BA.

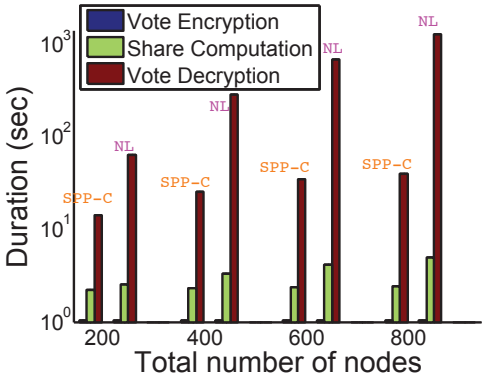
2) Computational Cost: In this experiment, for each size of the network, we average over all the nodes the computational cost of the steps in SPP-Computation. Figure 3.c illustrates the breakdown in terms of computational time. For each network size (*i.e.*, 200, 400, 600 and 800 nodes), the durations of the different steps for SPP-Computation can be seen on the left while those of NL are on the right. We plot the durations of the vote encryption, share computation, and vote decryption (*i.e.*, share combination). This figure highlights the efficiency of our protocol and indicates that decryption is the most expensive step in both protocols, thus supporting our approach of delegating this task to clusters of small size.



(a) Global communication cost



(b) Communication cost per node



(c) Computational complexity of each phases for 4 different network size. Given a network size, the three bars to the left correspond to the duration of the three steps for SPP while the three bars to the right correspond to the duration of the three steps for NL

Fig. 3. Experimental evaluation of our protocol (SPP) and of the Byzantine Agreement protocol (BA) against a non-layout based one (NL).

VII. Conclusion

In this paper, we have proposed SPP, a scalable and secure distributed protocol to conduct polls in a dynamic network. Its complexity is drastically lower than those of previously known algorithms and within a factor $\log^2 n$ of the optimal. The experimental evaluation illustrates that SPP-Computation significantly outperforms classical protocols in unstructured networks. Furthermore, the implementation shows that the proposed solution is efficient whereas protocols in unstructured networks cannot be used in practice in large networks due to scalability issues. We have also presented two applications of SPP: (1) an optimal Byzantine agreement protocol for dynamic networks as well as (2) a protocol for computing functions consisting in linear combination of inputs whose complexity is $\tilde{O}(n^{3/2})$. We leave as future work the design of a protocol closing the gap between the lower bound

of $\Omega(n \log n)$ and the upper bound $O(n \log^3 n)$ currently achieved by SPP as well as the possibility of extending our algorithms to other secure multiparty computations.

REFERENCES

[1] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", *ACM Transactions on Programming Languages and Systems (TOPLAS'82)*, vol. 4, no. 3, pp. 382–401, 1982.

[2] O. Baudron, P.A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical Multi-Candidate Election System", in *20th annual ACM symposium on Principles of distributed computing (PODC'01)*, 2001, pp. 274–283.

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game", in *19th annual ACM symposium on Theory of computing (STOC'87)*, 1987, pp. 218–229.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation", in *20th annual ACM symposium on Theory of computing (STOC'88)*, 1988, pp. 1–10.

[5] Z. Beerliová-Trubiniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity", in *5th conference on Theory of cryptography (TCC'08)*, 2008, pp. 213–230.

[6] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement", *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[7] Danny Dolev and Rüdiger Reischuk, "Bounds on information exchange for byzantine agreement", *J. ACM*, vol. 32, no. 1, pp. 191–204, Jan. 1985.

[8] B. Awerbuch and C. Scheideler, "Towards a Scalable and Robust DHT", *Theory of Computing Systems*, vol. 45, no. 2, pp. 234–260, 2009.

[9] I. Damgård and M. Jurik, "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System", in *Public Key Cryptography (PKC'01)*, 2001, pp. 119–136.

[10] V. King and J. Saia, "From almost everywhere to everywhere: Byzantine agreement with $\tilde{o}(n^{3/2})$ bits", *Distributed Computing*, pp. 464–478, 2009.

[11] D. Holtby, B.M. Kapron, and V. King, "Lower bound for scalable byzantine agreement", *Distributed Computing*, vol. 21, no. 4, pp. 239–248, 2008.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks", in *5th Symposium on Operating Systems Dessign and Implementation (OSDI'02)*, 2002, pp. 255–270.

[13] Christian Scheideler, "How to spread adversarial nodes? rotate!", in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, New York, NY, USA, 2005, STOC'05, pp. 704–713, ACM.

[14] Baruch Awerbuch and Christian Scheideler, "Towards a scalable and robust dht", in *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, New York, NY, USA, 2006, SPAA'06, pp. 318–327, ACM.

[15] O. Goldreich, *Foundations of cryptography: Basic tools*, Cambridge Univ Press, 2001.

[16] B. Awerbuch and C. Scheideler, "Towards scalable and robust overlay networks", in *6th International Workshop on Peer-To-Peer Systems (IPTPS'07)*, 2007.

[17] Matthias Baumgart, Christian Scheideler, and Stefan Schmid, "A dos-resilient information system for dynamic data management", in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, New York, NY, USA, 2009, SPAA '09, pp. 300–309, ACM.

[18] Christian Scheideler and Stefan Schmid, "A distributed and oblivious heap", in *Proceedings of the 36th Internatilonal Colloquium on Automata, Languages and Programming: Part II*, Berlin, Heidelberg, 2009, ICALP'09, pp. 571–582, Springer-Verlag.

[19] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer, "Towards worst-case churn resistant peer-to-peer systems", *Distributed Computing Journal (DC)*, vol. 22, no. 4, pp. 249–267, May 2010.

[20] A-C. Yao, "Protocols for Secure Computations", *23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, pp. 160–164, 1982.

[21] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty Unconditionally Secure Protocols", *20th annual ACM symposium on Theory of computing (STOC'88)*, pp. 11–19, 1988.

[22] D. Beaver, "Multiparty protocols tolerating half faulty processors", in *Advances in Cryptology (CRYPTO'89)*, 1990, vol. 435 of *Lecture Notes in Computer Science*, pp. 560–572.

[23] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority", in *21rst annual ACM symposium on Theory of computing (STOC'89)*, 1989, pp. 73–85.

[24] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin, "Efficient multiparty computations secure against an adaptive adversary", in *Advances in Cryptology (Eurocrypt'99)*, 1999, pp. 311–326.

[25] R. Cleve, "Limits on the security of coin flips when half the processors are faulty", in *18th annual ACM symposium on Theory of computing (STOC'86)*, 1986, pp. 364–369.

[26] B. Chor and E. Kushilevitz, "A Zero-One Law for Boolean Privacy", *SIAM Journal on Discrete Mathematics*, vol. 4, no. 1, pp. 36–47, 1991.

[27] D. Beaver and S. Goldwasser, "Multiparty computation with faulty majority", in *Advances in Cryptology (CRYPTO'89)*, 1990, pp. 589–590.

[28] S. Goldwasser and Y. Lindell, "Secure computation without agreement", *Distributed Computing*, pp. 17–32, 2002.

[29] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank, "On achieving the best of both worlds in secure multiparty computation", *SIAM journal on computing*, vol. 40, no. 1, pp. 122–141, 2011.

[30] A. Giurgiu, R. Guerraoui, K. Huguenin, and A-M. Kermarrec, "Computing in Social Networks", *Lecture Notes in Computer Science*, vol. 6366, pp. 332–346, 2010.

[31] V. King and J. Saia, "Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary", in *Proceeding of the 29th ACM symposium on Principles of distributed computing (PODC'10)*, 2010, pp. 420–429.

[32] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani, "Fast asynchronous byzantine agreement and leader election with full information", in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA'08)*, 2008, pp. 1038–1047.

[33] P. Paillier, "Public-key Cryptosystems based on Composite Degree Residuosity Classes", *Advances in Cryptology (EUROCRYPT'99)*, 1999.

[34] B. Awerbuch and C. Scheideler, "Robust random number generation for peer-to-peer systems", *Principles of Distributed Systems*, pp. 275–289, 2006.

[35] S. Gambs, R. Guerraoui, H. Harkous, F. Huc, and A. Kermarrec, "Scalable and Secure Aggregation in Distributed Networks", *Arxiv preprint 1107.5419*, 2011.

[36] M. Young, A. Kate, I. Goldberg, and M. Karsten, "Practical Robust Communication in DHTs Tolerating a Byzantine Adversary", in *International Conference on Distributed Computing Systems (ICDCS'10)*, 2010, pp. 263–272.

[37] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults", in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1985, SFCS '85, pp. 383–395, IEEE Computer Society.

[38] Arpita Patra Michael Backes, Aniket Kate, "Computational verifiable secret sharing revisited", in *ASIACRYPT*, Dong Hoon Lee and Xiaoyun Wang, Eds. 2011, vol. 7073 of *Lecture Notes in Computer Science*, pp. 590–609, Springer.

[39] T. Nishide and K. Sakurai, "Distributed Paillier Cryptosystem without Trusted Dealer", *Information Security Applications*, pp. 44–60, 2011.

[40] T. Yuen, Q. Huang, Y. Mu, W. Susilo, D. Wong, and G. Yang, "Efficient non-interactive range proof", in *15th Annual International Conference on Computing and Combinatorics (COCOON'09)*, 2009, pp. 138–147.

[41] V. King, S. Lonargan, J. Saia, and A. Trehan, "Load balanced scalable byzantine agreement through quorum building, with full information", *Distributed Computing and Networking*, pp. 203–214, 2011.

[42] G. Bracha, "An asynchronous [(n - 1)/3]-resilient consensus protocol", in *Proceedings of the third annual ACM symposium on Principles of distributedcomputing (PODC'84)*, 1984, pp. 154–162.

[43] Data and Privacy Lab, "Paillier threshold encryption toolbox", July 2011.